

# Polyphonic Hybrid Synthesizer: *NepTune*

Summer 2025 – Fall 2025

Group 16



## Authors:

Robert Rosario      Dina Suleiman  
(EE)                    (EE)

Alexander Barrass    Anthony De Rosa    Jordan Khan  
(EE)                    (EE)                    (CpE)

## Reviewers:

Prof. Mark Maddox  
Dr. Justin Phelps  
Dr. Saleem Sahawneh  
Dr. Kalpathy B. Sundaram

# Acknowledgements

## Major Acknowledgements

Before starting there are several individuals the team would like to express our gratitude towards. The amount of help ranges from something as small as bit of advice the team found helpful, to acts that saved the entire project.

Firstly, the team would like to express our thanks to Dr. Chung-Yong Chan, and Dr. Arthur Weeks. Without the advice of Dr. Chan we would not have made it through Senior Design I. In similar fashion, without the technical help of Dr. Weeks, we would not have made it through Senior Design II. Many a time we sought help from these professors, and they were able to provide guidance the team needed.

Next the team would like to thank Professor Maddox for providing advice many times throughout Senior Design I. He provided much useful information on the difficulties surrounding mixed signal design and provided advice on what types of ICs (integrated circuits) to look for. He always made time for us to meet and for that we are thankful.

Next, the team would like to thank Emilio Armas, a fellow undergraduate who stepped in during the late hours of the night to help our team when we needed it the most. He, being familiar with the STM32 processors, helped resolve a major problem regarding communication protocols. Without him, it's safe to say the project would not have been fully completed.

The team would also like to thank our entire review committee: Dr. Saleem Sahawneh, Professor Mark Maddox, Dr. Kalpathy Sundaram, and Dr. Justin Phelps. The project would not be possible without their support and advice through the project. We also thank them for remaining patient on presentation day when our project stopped working thirty minutes before the scheduled presentation.

## Minor Acknowledgements

The team would also like to thank several individuals who provided help at some point throughout the project. Thank you, Joshua Yu, for your support with the STM32. Thank you, Bradley Spencer, for providing tips on PCB design. Thank you, Jacob Riesterer, for your help with the output amplifier design. Thank you, Vera Fodor, for helping with troubleshooting the STM32. Thank you, Ibrahim Khater, for your help with physical construction. Thank you, Riley Newport and Nicholas Miguel, for your help with the showcase poster. The team appreciates all of you!

# Table of Contents

Chapter 1 – Executive Summary .....	1
Chapter 2 – Project Description.....	2
2.1 Project Background and Motivation .....	2
2.2 Short History of Synthesizers .....	3
2.3 Existing Products .....	4
2.4 Goals and Objectives .....	5
2.4.1 Basic Goals: .....	5
2.4.2 Advanced Goals:.....	6
2.4.3 Stretch goals:.....	7
2.5 Table of Key Engineering Specifications with Quantitative Measures .....	9
2.6 House of quality .....	11
2.7 Hardware Block Diagram .....	12
2.8 Software Block Diagram.....	13
2.9 Prototype illustration/blueprint .....	14
Chapter 3 – Synthesizer Fundamentals.....	15
3.1 Oscillators .....	15
3.2 Voices .....	15
3.3 Filters .....	16
3.4 ADSR (Attack Decay Sustain Release) Envelope Generator .....	16
3.5 (VCA) Voltage Controlled Amplifier.....	18
3.6 Low-Frequency Oscillators.....	18
3.7 Control Voltages .....	18
3.8 MIDI .....	19
3.8.1 MIDI Controllers .....	19
3.8.2 MIDI Connectors .....	20
Chapter 4 – Research and Investigation.....	22
4.1 Synthesizer Design Methodology .....	22
4.1.1 Types of Synthesis .....	22
4.1.2 Analog vs Digital Circuitry.....	25
4.1.3 Discrete Circuits vs Integrated Circuits .....	29
4.2 Hardware Parts and Technology .....	31
4.2.1 Digital Processing System .....	31
4.2.2 Flash Memory .....	37
4.2.3 Digital to Analog Conversion .....	40
4.2.4 MIDI Keyboard Interface .....	47
4.2.5 User Interface .....	52

4.2.6 Audio Signal Filtering.....	54
4.2.7 Low Frequency Oscillators.....	61
4.2.8 Signal Amplification.....	67
4.2.9 Envelope Generation.....	71
4.2.10 Audio Mixer.....	75
4.2.11 Audio Output Connections .....	78
4.2.12 Audio Output Amplifier.....	79
4.2.13 Power Supply .....	84
4.3 Serial Communication Protocols (SCP).....	92
4.3.1 I2C.....	93
4.3.2 SPI.....	94
4.3.3 I2C vs. SPI – PCM3060.....	94
4.3.4 PCM3060 HW mode.....	96
4.3.5 UART over MIDI Communication.....	97
4.4 Software Parts and Technology .....	98
4.4.1 Embedded Language Comparison .....	99
4.4.2 Software Audio Library Comparison.....	103
4.4.3 Development Board Final Selection .....	109
Chapter 5 – Standards and Design Constraints.....	116
5.1 Standards.....	116
5.1.1 IPC Standard: IPC-2221A.....	117
5.1.2 UL Standard: UL-60065 .....	119
5.1.3 MIDI Standards and Implementation.....	122
5.2 Constraints .....	128
5.2.1 Major Constraints.....	128
5.2.2 Minor Constraints .....	130
Chapter 6 – Application of LLM Platforms.....	134
6.1 Comparison of Models.....	134
6.1.1 OpenAI – ChatGPT.....	134
6.1.2 DeepSeek – DeepSeek R1 .....	135
Chapter 7 – Hardware Design.....	138
7.1 Overall Schematic .....	138
7.2 MCU Circuit .....	139
7.3 Memory Circuit.....	140
7.5 User Interface Circuit.....	141
7.x CODEC Circuit.....	141
7.6 VCF Circuit.....	143
7.7 LFO Circuit.....	143
7.8 VCA, Envelope Generator, and Mixer Circuit .....	144

7.4 Keyboard Circuit.....	149
7.9 Output Amplifier Circuit.....	149
7.9.1 Headphone Driver.....	150
7.9.2 Line-Level Output Driver .....	152
7.10 Power Circuit .....	152
7.10.1 Analog Board Power Circuit.....	153
7.10.1 Digital Board Power Circuit .....	154
Chapter 8 – Software Design .....	<b>Error! Bookmark not defined.</b>
Chapter 9 – System Fabrication & Prototype Construction.....	160
9.1 Printed Circuit Board (PCB) Layouts .....	162
9.1.1 VCF PCB .....	162
9.1.2 LFO PCB .....	163
9.1.4 VCA, Envelope Generator, and Mixer PCB .....	164
9.1.5 Audio Output PCB.....	165
9.1.6 Keyboard PCB .....	166
9.1.7 Analog Power PCB .....	166
9.1.6 Digital Power PCB.....	167
Chapter 10 – System Testing and Evaluation .....	168
10.1 Hardware Testing.....	168
10.1.1 Daisy Seed .....	168
10.1.2 SSI2140.....	168
10.1.3 SSI2164.....	168
10.1.4 SSI2190.....	168
10.1.5 ENVGEN 8D .....	169
10.1.6 VCLFO 10 .....	169
10.1.7 Audio Output Amplifier.....	169
10.2 Software Testing .....	170
Chapter 11 – Administrative Content .....	171
11.1 Budget and Financing .....	171
11.2 Project milestones for SD1 and SD2 .....	172
11.3 Division of Labor .....	174
11.4 Team Organization.....	175
Chapter 12 – Conclusion.....	176
Appendices.....	i
Appendix A – References .....	i
Appendix B – copyright permission .....	vi
Appendix C – data sheet .....	viii
Appendix D – Software Coding and Debugging .....	ix
Appendix E – LLM prompts and outcomes.....	xxv

# List of Tables

<b>Table 2.1</b> Engineering Specifications .....	9
<b>Table 2.2</b> Marketing Requirements.....	10
<b>Table 4.1:</b> Pros and Cons for Analog and Digital Circuitry.....	28
<b>Table 4.2:</b> Pros and Cons for Topologies Involving Discrete or Integrated Circuits.....	31
<b>Table 4.3:</b> Pros and Cons for DSP System. ....	34
<b>Table 4.4:</b> Pros and Cons for Various MCUs. ....	36
<b>Table 4.5:</b> Pros and Cons for Memory Technology .....	38
<b>Table 4.6:</b> Pros and Cons for Memory Options .....	40
<b>Table 4.7:</b> Pros and Cons for DAC Technology.....	44
<b>Table 4.8:</b> Pros and Cons for Various DACs .....	46
<b>Table 4.9:</b> Pros and Cons for Various Keyboard Technologies.....	49
<b>Table 4.10:</b> Pros and Cons for Various Keyboard Switches .....	51
<b>Table 4.11:</b> Pros and Cons for Various Multiplexers.....	53
<b>Table 4.12:</b> Pros and Cons for Various Filter Technologies .....	57
<b>Table 4.13:</b> Pros and Cons for Various Filter ICs. ....	60
<b>Table 4.14:</b> LFO Technology.....	63
<b>Table 4.15:</b> Pros and Cons for Various LFO ICs.....	66
<b>Table 4.16:</b> Pros and Cons for Various VCA Technologies.....	68
<b>Table 4.17:</b> Pros and Cons for Various VCA ICs.....	70
<b>Table 4.18:</b> Pros and Cons for Various ADSR Technologies.....	72
<b>Table 4.19:</b> Pros and Cons for Various ADSR ICs. ....	75
<b>Table 4.20:</b> Pros and Cons for Various Audio Mixer ICs .....	77
<b>Table 4.21:</b> Pros and Cons for Types of Operational Amplifiers.....	81
<b>Table 4.22:</b> Pros and Cons for Stage 1 (I-V) Operational Amplifiers.....	82
<b>Table 4.23:</b> Pros and Cons for Stage 2 (Output Driver) Operational Amplifiers.....	82
<b>Table 4.24:</b> List of Parts with Power Requirements.....	87
<b>Table 4.25:</b> Pros and Cons for Various Communication Medium .....	96
<b>Table 4.26:</b> Pros and Cons of Languages Used by Teensy, Daisy Seed, and Bela .....	102
<b>Table 4.27:</b> Pros and Cons of Various Libraries .....	108
<b>Table 4.28:</b> Pros and Cons of Various Platforms.....	109
<b>Table 7.1:</b> Sensitivity, Impedance, Peak Voltage, and Peak Current of Headphones (Permission has been requested for this table).....	150
<b>Table 7.2:</b> Headphone Driver Design Criteria.....	151

<b>Table 11.1:</b> Budget distribution.....	171
<b>Table 11.2:</b> Senior Design I Project and Team Milestones .....	172
<b>Table 11.3:</b> Senior Design I Deliverable Deadlines .....	173
<b>Table 11.4:</b> Senior Design II Milestones .....	174
<b>Table 11.5:</b> Division of Labor .....	175

## List of Figures

<b>Figure 2.1</b> An enriching journey often begins with simple curiosity. (Courtesy of Anabella Suluh).....	Error! Bookmark not defined.
<b>Figure 2.2:</b> House of Quality Diagram .....	11
<b>Figure 2.3:</b> Hardware Block Diagram.....	12
<b>Figure 2.4:</b> Software Block Diagram.....	13
<b>Figure 2.5:</b> Renders for the final product. ....	14
<b>Figure 3.1:</b> Left to right: LEKATO Mini MIDI Keyboard, Akai MPCStudio2 Studio Drum Pad, Jamstik Standard MIDI Guitar, and Yamaha WX5 MIDI Wind Controller.....	20
<b>Figure 3.2:</b> A representative MIDI setup example of a 5-DIN connection from a MIDI keyboard controller to multiple MIDI controllers via daisy chaining and a USB MIDI connection between the keyboard and a computer [18]. .....	21
<b>Figure 4.1:</b> Additive synthesis adds multiple sine waves to create one complex waveform. ....	23
<b>Figure 4.2:</b> FM Modulation consists of modulator and carrier waveforms. ....	24
<b>Figure 4.3:</b> Wavetable synthesis consists of a table of waveforms to morph between....	24
<b>Figure 4.4:</b> Binary Weighted DAC layout. ....	41
<b>Figure 4.5:</b> R-2R ladder DAC layout. ....	42
<b>Figure 4.6:</b> Sigma-Delta DAC diagram. ....	43
<b>Figure 4.7</b> Capacitive touch keyboard. ....	48
<b>Figure 4.8:</b> Mechanical switch diagram .....	49
<b>Figure 4.9:</b> Sallen and Key filter configurations.....	55
<b>Figure 4.10:</b> State variable filter configuration of SSI2140.....	56
<b>Figure 4.11:</b> Cauer-Elliptical Filter Configuration of SSI2140. ....	56
<b>Figure 4.:</b> Frame diagram of an I2C message [12].....	93
<b>Figure 5.1:</b> Topics covered under IPC-2221A .....	117
<b>Figure 5.2:</b> UL standards for individual temperature of parts. ....	121
<b>Figure 5.3:</b> Binary format of MIDI message.....	122
<b>Figure 7.1:</b> Overall schematic of synthesizer shown in hierachal format. Smaller subsystems are represented by blocks in the top portion of the image. ....	138
<b>Figure 7.2:</b> MCU Schematic.....	139
<b>Figure 7.3:</b> Schematic for flash memory. ....	140

<b>Figure 7.4:</b> Schematic for interface controls. These are the buttons and knobs on the face of the board that the user will interact with.....	141
<b>Figure 7.5:</b> Schematic for both CODECs.....	142
<b>Figure 7.6:</b> Schematic for voltage-controlled low-pass filter with output gain Q-compensation .....	143
<b>Figure 7.7:</b> Schematic for LFO circuit including the VCLFO10 IC and output filters..	144
<b>Figure 7.8:</b> Schematic for ADSR including ENVGEN8. ....	145
<b>Figure 7.9:</b> Schematic of ADSR Envelope Generators (ENVGEN8) and Quad VCA IC (SSI2164) and the Mixer IC (SSI2190). .....	148
<b>Figure 7.10:</b> Schematic for keyboard matrix. This will be the MIDI keyboard on which the user plays. ....	149
<b>Figure 7.11:</b> Headphone driver circuit shown in upper half of image. ....	151
<b>Figure 7.12:</b> Line-level driver circuit.....	152
<b>Figure 7.13:</b> +12V and -12V Input Voltage Regulator.....	153
<b>Figure 7.14:</b> +5V and -5V Regulator.....	153
<b>Figure 7.15:</b> Power circuit schematic.....	154
<b>Figure 8.1</b> .....	160
<b>Figure 9.1:</b> PCB layout for four VCF ICs. Each voice has its own VCF.....	162
<b>Figure 9.2:</b> PCB layout containing the LFO circuit. U1 is Electric Druid's VCLFO10, and U2 is the TL072 which filters the output of the LFO. ....	163
<b>Figure 9.:</b> PCB layout containing the VCA and ADSR envelope generators for each of the four voices. The mixer which combines the four voices into one signal is also included in this PCB layout. ....	164
<b>Figure 9.: Output amplifier circuit. This layout contains the amplifier circuits for both the line-level and headphone outputs.</b> .....	165
<b>Figure 9.: MIDI Keyboard PCB. This is the layout for the keyboard the user will play on.</b> .....	166
<b>Figure 9.6:</b> -12V Analog Power to be connected to ICs .....	166
<b>Figure 9.7:</b> -5V Analog Power to be connected to ICs .....	167
<b>Figure 9.8:</b> -5V Analog Power not connected to ICs .....	167
<b>Figure 10.1:</b> LTspice simulation schematic of audio output amplifier. (a) Headphone amplifier circuit. (b) Line-level output circuit. ....	170

# Chapter 1 – Executive Summary

*NepTune* is a portable, hybrid analog/digital music synthesizer developed to deliver rich, professional sound in an accessible, fun, and modern package. In the landscape of electronic music, the synthesizer defines the limits of creation. Users will find themselves between the steep barrier of entrance of professional-grade synthesizers, or the limiting confines of entry-level devices; however, our team of electrical and computer engineers from the University of Central Florida have put our minds together to bridge the gap between the two worlds without sacrificing depth or tonal character.

At its core, *NepTune* aims to be a journey into subtractive synthesis, providing rich classic waveforms such as sine, square, sawtooth, and triangle waves to be the launchpad for the musician's vision. The instrument has been designed for immediate interaction by forgoing the need for menu-diving and providing a tactile, knob-per-function interface and mechanical keys that provide a direct and tangible dialogue between the musician and the machine. This hands-on philosophy is paired with a portable chassis that frees the user from the confines of their studio, allowing the music to be brought with them anywhere they go.

The sonic identity of *NepTune* is derived from the combination of both analog and digital domains. The digital architecture lends precision, stability, and versatile control, while the warmth of the analog signal modulation brings a richness and nuance to the sound that a digital synthesizer may lack. This pairing ensures that the instrument is both forgiving for a novice and rewarding for an experienced sound designer, offering a wide range of possibilities for everyone to enjoy.

The following document is a record of every decision that the team has made regarding technology pursued, parts implemented, goals and testing methods, as well as thought processes behind *Neptune*'s design. It also documents the rigorous iterations that NepTune has taken as the design process has progressed along with how these decisions were made in regard to environmental conditions and available scope of the project. Contributions from each member lay side by side to formulate the complete NepTune design which will be delivered as reality once manufacturing is complete, and prototyping concludes.

Ultimately, *NepTune* represents the culmination of a dedicated engineering effort to create a cohesive and capable instrument. It stands as a testament to the idea that thoughtful design can make powerful creative technology approachable, reliable, and resonant for a new generation of artists and engineers alike.

# Chapter 2 – Project Description

## 2.1 Project Background and Motivation

It is fair to say that for most people, music is an integral part of daily life. This statement is made even more true in the modern day. Thanks to technological advancements, music has become more accessible and pervasive than ever. It serves as a powerful medium for expression, communication, and connection across societies. This senior design project is rooted in an appreciation for this positive impact. The team hopes to spread this positive influence even further, by creating a tool that lowers the barrier of entry to music-making. NepTune synthesizer is intended to be approachable, intuitive, and appealing to curious individuals who may be new to musical synthesis. The team hopes to spark curiosity in others by offering an accessible entry point into this field. Often, what begins as a simple interest can evolve into a deep “rabbit hole” filled with discovery, experimentation, and personal expression. The goal is to make that first step inviting, so that users feel encouraged to explore further and find the same inspiration and joy that the team has experienced through engaging with synthesis. Ultimately, the team wishes to use the team’s newly developed skills to create something that can shine a bit of light on someone’s day, whether they are a musician or a listener.

Despite technological advancements, many synthesizers remain costly. This creates a high barrier of entry for those interested in synthesizers. Of course, there is the software solution, where most synthesizers can be replicated via a digital audio workstation (DAW), but the team wanted to offer something physical. Therefore, the goal of this project is to create a more affordable solution to synthesizer hardware.

In addition, a musical synthesizer presents an ideal technical challenge for senior design students. Synthesizers generate and manipulate audio waveforms in real time, requiring multiple systems to work in harmony with one another. This design will draw upon concepts learned in courses on circuits, electronics, signal processing, and embedded systems, making it extremely relevant to the team’s academic background and engineering interests.

An anticipated question may be: Why choose to include analog circuitry in the system rather than focus on a purely digital design? There are a few answers to this. Firstly, the project was defined to match the skills that the team possessed. As a team of four Electrical Engineers and one Computer Engineer, the decision was made to offload a lot of the signal processing to specialized analog circuitry. In addition, offloading the signal processing frees up the MCU to perform more process-intensive effects such as reverb, delay, or more complex oscillation. This means that more effects can be added via software if project goals are met early. Finally, this product is developed with the electronics-hobbyist community in mind. The idea is that those coming from the synthesizer world who are interested in electronics might find analog circuitry a bit easier to understand.

## 2.2 Short History of Synthesizers

The history of analog synthesizers began during the mid-20th century with the groundbreaking innovations pioneered by RCA engineers Harry Olson and Herbert Belar. As the first programmable electronic instrument, the 1957 RCA Mark II represented one of the greatest leaps in music technology. It used vacuum-tube oscillators for precise pitch control, alongside noise generators and resonant filters to shape complex tones. Its paper-tape sequencer allowed composers to program automated music by punching holes into the tape, an early form of digital control, while envelope generators and a switchboard modulation matrix enabled dynamic amplitude shaping and signal routing. The Mark II granted unprecedented tonal precision and self-playing capability, yet its impractical scale (7 feet tall, 20 feet wide) and cost (equivalent to ~\$1.8 million today) made it inaccessible outside institutional settings.

By the 1960s and 1970s, Robert Moog and Don Buchla—working independently on opposite U.S. coasts—revolutionized electronic music through divergent philosophies. Moog's designs prioritized familiarity, integrating voltage-controlled oscillators, filters, and envelopes with a piano-style keyboard to democratize synthesis. His Modular System, using transistors instead of vacuum tubes, compacted synthesis into tabletop units and popularized subtractive synthesis via the iconic "warm" ladder filter. Buchla, conversely, rejected Western musical conventions entirely. His 100 Series replaced keyboards with capacitive touch-plates and eschewed traditional waveforms for complex wavefolding, framing the synthesizer as a laboratory for experimental signal routing and modulation.

The Minimoog Model D (1970) marked a pivotal shift in synthesizer design by proving that an analog synthesizer could be both portable and performative. Its fixed signal path and compact keyboard interface proved that accessibility does not need to compromise the tonal characteristics of a synthesizer. Yet, while the Minimoog excelled at subtractive synthesis, its analog oscillators suffered from tuning drift—a limitation that plagued synthesizers until the development of synths such as the PPG Wave 2 (1981).

The PPG Wave introduced digitally stabilized oscillators (8-bit wavetables) paired with analog filters, eliminating tuning instability while retaining nonlinear analog warmth. This hybrid architecture proved that digital precision could coexist with analog signal paths, yielding complex sounds that were both stable across octaves and rich in texture, without being limited to the simple waveform of analog synthesizers before it. However, the Wave's aliasing artifacts and reliance on expensive peripherals underscored the era's trade-offs - challenges that modern components and design approaches now allow us to overcome.

Three decades of technological progress have transformed early hybrid concepts into practical realities. Today's DSP delivers pristine digital oscillators without aliasing, while modern analog components offer stability in compact form factors. Instruments like the Dave Smith Evolver (2002) and Korg Minilogue XD (2019) proved hybrid architectures could be both affordable and roadworthy - yet many still compromise where it matters most.

Buried menus sacrifice immediacy; proprietary components hinder repair; overly integrated designs prevent meaningful servicing.

Building on these foundations, this synthesizer merges immediate tactile control with digital stability while solving their historical limitations. The team preserves analog signal integrity through modern implementations of classic filter and amplifier designs, but with rock-solid digital oscillators that eliminate tuning drift. Neptune's digital voice architecture leverages current DSP technology to provide polyphony and waveform flexibility without the aliasing artifacts or peripheral dependencies of early hybrids. And crucially, where others use proprietary or difficult to repair construction, the design team has prioritized modular, serviceable components. The result is an instrument that combines playability and sonic potential in a reliable, affordable package designed for today's musicians.

## 2.3 Existing Products

To determine the parameters of the synthesizer and what was hoped to achieve, the team have conducted thorough background research into various synthesizers currently available for commercial use. The first product that was looked at is the Behringer MS-1 / MS-101 MKII [2]. This specific synthesizer stands out because it is a monophonic analog synthesizer utilizing subtractive synthesis. It has four oscillators providing a rich sound to its single voice. It has a great layout, fitting 57 knobs and sliders into an 85 x 569 x 267mm package, making it a decent size to be carried. The only drawback is it's a bit heavy, weighing around 10lbs. However, it boasts 32 semi-weighted full-sized keys. This became part of NepTune's inspiration as ergonomics utilizing both simple layout and plenty of keys was something the team wanted to implement. Finally, this synthesizer remains on the affordable scale reaching up to \$259, making it a great option for price sensitivity.

Next, Arturia's Microfreak [1] was examined. It is a hybrid synthesizer, utilizing both digital and analog components, similar to what the team will implement. This synthesizer is unique in the fact that it utilizes digital oscillators with analog filters, while maintaining a small design and packing in lot of features. It comes in with a weight of 2lbs and a size of 55 x 311 x 233mm making it very portable and is priced at \$405. The sound of this hybrid synth is similar to what the team desires out of *NepTune*. In fact, after hearing it, the decision was made to implement polyphony instead of mono- or paraphony. However, a common complaint from customers about the Microfreak is that there are too many controls crammed into the small interface. This is something the team plans to avoid in this *NepTune*.

Another notable product is a previous senior design project called "Synthia." This project was a fully analog monophonic synthesizer that generated sounds based on a MIDI input. The team found the design very interesting and took some inspiration to create a hybrid synthesizer with multiple voices. This would challenge us to expand the teams' skills in analog design. The synth will also offer MIDI capabilities as well, making it compatible with other instruments, synths, and computers.

## 2.4 Goals and Objectives

Exploring the wide variety of synthesizers available on the market revealed the amount of creative potential this project holds. While it is tempting to build an elaborate synthesizer with many features, such a product would be unrealistic given the time, cost, and knowledge constraints. After considering factors such as schedule, experience, budget, and personal interests, the team agreed that the most valuable approach would be to design a simple synthesizer that functioned well. The most productive use of this opportunity provided by Senior Design would be to focus on creating a synthesizer that functions like a finished, marketable product.

This direction was chosen because it aligns with the team's capabilities. No one on the team has ever built a synthesizer before, and some have not had much hands-on experience with electronics outside of the classroom. With this in mind, there was an agreement to define a project with an adequate balance between attainability and complexity. This would empower the team to make the most out of Senior Design and have fun along the journey.

Listed below are the goals and features the team hopes to achieve in *NepTune*. They are broken down into basic, advanced, and stretch goals. The basic and advanced goals are to be expected in the final demonstration in Senior Design II. These are fundamental aspects of a synthesizer and cannot be negotiated. On the other hand, the stretch goals are not guaranteed in the final demonstration. Instead, these are goals that the team hopes to achieve if everything works out as planned. This is because they are more difficult and will require some extra time to plan, design, and prototype. If everything works as expected, however, the team plans to reach all goals: basic, advanced, and stretch.

Some of the terminology used in the following sections are specific to synthesizers and may cause some confusion. To avoid this, the team has provided an additional chapter, Chapter 3, in which the basic terminology and explanation of various parts of a synthesizer are explained in more detail. Therefore, it may be beneficial to refer to that chapter, should any confusion arise from reading the following goals and objectives.

### 2.4.1 Basic Goals:

Listed below are the basic goals to be achieved in *NepTune*. These goals include the fundamental parts of a synthesizer and as a result are required in the final product. Because of this, they are expected to be demonstrated at the end of Senior Design II.

- 2 Oscillators per voice
- Voltage controlled filters (VCF)
- Voltage controlled amplifiers (VCA)
- Envelope generators
- Portability

**2 oscillators per voice.** To fit within the project constraints, the team has chosen to implement 2 oscillators per voice. This configuration provides a sound that has some richness to it but also yields a simple design. A simpler design enhances scalability,

allowing the two-oscillator voice architecture to be duplicated and used to realize polyphony (multiple voices).

**Voltage controlled filters (VCF).** Filters are an important component to any synthesizer. Filters enable control over which frequencies pass through. This affects the timbre of the sound and controls whether it sounds dull or sharp. Filters may be arranged in high-pass, low-pass, or band-pass configurations. Low-pass filters are the most common because they produce a nicer sound. For NepTune, expect to have at least a low-pass filter. If time allows, the team will incorporate more types of filters. Thus, at least one low-pass filter per voice will be included in the final product.

**Voltage controlled amplifiers (VCA).** Like filters, amplifiers are an integral part of a synthesizer. Each voice must have its own amplifier to allow for amplitude shaping (covered in the next goal). This is the key to realizing a polyphonic synthesizer and will be a requirement in the final product.

**Envelope generators.** Envelope generators are responsible for shaping the volume of every keypress. This feature is what enables synthesizers to mimic the behavior of a classical piano. When a key is pressed, a hammer strikes the string, producing a sharp initial sound. If the key is held, the volume is sustained at a reduced level. When the key is released, the sound immediately fades. This is the type of behavior an envelope generator typically seeks to replicate. To accomplish this, an envelope generator modulates the volume of the voltage-controlled amplifier (VCA). Additionally, since there is one VCA per voice in a polyphonic system, there must also be one envelope generator per voice.

**Portability.** This is a key feature the team plans to implement for NepTune. With modern advancements in software and hardware enabling significant reductions in circuit size, it is reasonable to consider portability a baseline expectation for a compact synthesizer. The team has decided that the final product should fit inside a standard backpack. As such, the design will need to prioritize a rugged and durable form factor. This size constraint may limit the types of features or analog circuitry that can be included. However, as the project progresses and the team gains more experience, the feasibility of various design elements will become clearer.

#### 2.4.2 Advanced Goals:

The advanced goals outlined below are intended to be part of the final product. Given their complexity, they are categorized separately because they will require greater effort to achieve than the basic goals. These features are also to be expected in the final demonstration of Senior Design II.

- 2 Voices
- 24 Keys (2 octaves) with pitch up and down capabilities.
- Low Frequency Oscillator (LFO) applied to all low-pass VCF.

**2 Voices.** Even though it is not uncommon nowadays to find polyphonic synthesizers with upwards of five or six voices, NepTune will incorporate some level of analog hardware which makes achieving a higher voice count more difficult. For instance, each voice will

require its own “voice card” consisting of a VCF, VCA, and envelope generator. These components require extensive care in design and manufacturing to ensure a consistent sound across all voices. As the number of voices increases, the level of hardware necessary can multiple dramatically.

Although support for more than two voices is a desirable goal, the most important aspect of NepTune is to be a quality product. As a result, to remain within project constraints, the design will initially support two voices with the possibility of expanding voice count if development progresses smoothly.

**24 keys.** Determining the number of keys was a significant design decision. The choice of 24 keys hinges on the balance between portability and performance, as 24 keys are just enough to cover two octaves. While a larger keybed spanning more octaves would be ideal, several constraints influence the final decision. These include ongoing considerations regarding key size and type, as well as the requirement for the final product to remain portable and fit within a standard backpack. Opting for smaller keys would allow more to fit on the board at the cost of ergonomics.

However, to ensure that the user is not limited to two octaves, shift buttons will be included that will allow the user to shift the pitch up or down in octaves. This will extend the playable range beyond two octaves and enable access to a broader pitch range despite the limited number of physical keys.

**LFO applied to VCF.** Low-frequency oscillators are a staple for any subtractive synthesizer and must be included in *NepTune*'s design. LFOs introduce dynamic variation and can be used to modulate any of the audio processing circuits in the signal path. They can be applied to the oscillators making up the voice, they can be applied to the amplifiers, and they can be applied to the filters. In *NepTune*, the LFOs will be used specifically to modulate the low-pass filters.

#### 2.4.3 Stretch goals:

With the basic and advanced goals defined, the last set of goals to be examined are the stretch goals. These goals are not guaranteed to be in the final product at the Senior Design II demonstration. Although the team plans on attaining these goals, some of them may not be included due to unforeseen delays or complications. This is understandable given the tight project constraints (time, money, and experience). Nevertheless, the team will try their best to achieve these in the final design.

- 6 voices
- Presets
- Arpeggiator
- Screen
- Sound system

**6 Voices.** As previously mentioned, many of the synthesizers on the market today have upwards of five or six voices. To make *NepTune* competitive, increasing the number of supported voices is an important consideration. These additional voices will challenge the design constraints, due to the increase in hardware within the limited physical space. Expanding from two to six voices would require six voice cards instead of two. However, if two voices can be successfully implemented, it will provide the necessary insight into whether the number of voices can be scaled up within the space constraints.

**Presets.** In the modern digital age, the ability to save presets is often taken for granted. If a musician were to tune all the parameters such as filters, envelopes and LFOs, to craft a desired sound, they expect to save that configuration as a preset for easy recall. This is not possible to do with analog circuitry. Once knobs and sliders are moved, the previous configuration is lost forever unless the musician is able to recall their positions manually. Thus, to facilitate preset storage within *NepTune*, some solution will need to be found.

**Arpeggiator.** Arpeggiators are a common feature on many modern synthesizers. To remain competitive, *NepTune* should include some type of arpeggiator. Several different patterns of arpeggiator can be employed. A decision has not been made on what type of arpeggiator to employ, but the team is determined to include this feature in some form.

**Screen.** The synthesizer may have a small screen on the board. It will primarily serve to display the live waveform of the output, similar to an oscilloscope. This will give the user a visual aid when tuning and adjusting parameters. If time allows, the screen can also be used to display menus such as parameter values, presets, and more.

**Sound System.** The synthesizer may have a small sound system. A decision has not been made on whether the sound system will be an internal or separate external part. If there is enough space within the synthesizer, the team will try to implement an internal one. However, it is still early in the design phase and difficult to determine whether an internal sound system is possible as part sizes have not been finalized. Because it is a stretch goal, the priority for this decision is low.

## 2.5 Table of Key Engineering Specifications with Quantitative Measures

**Table 2.1** Engineering Specifications

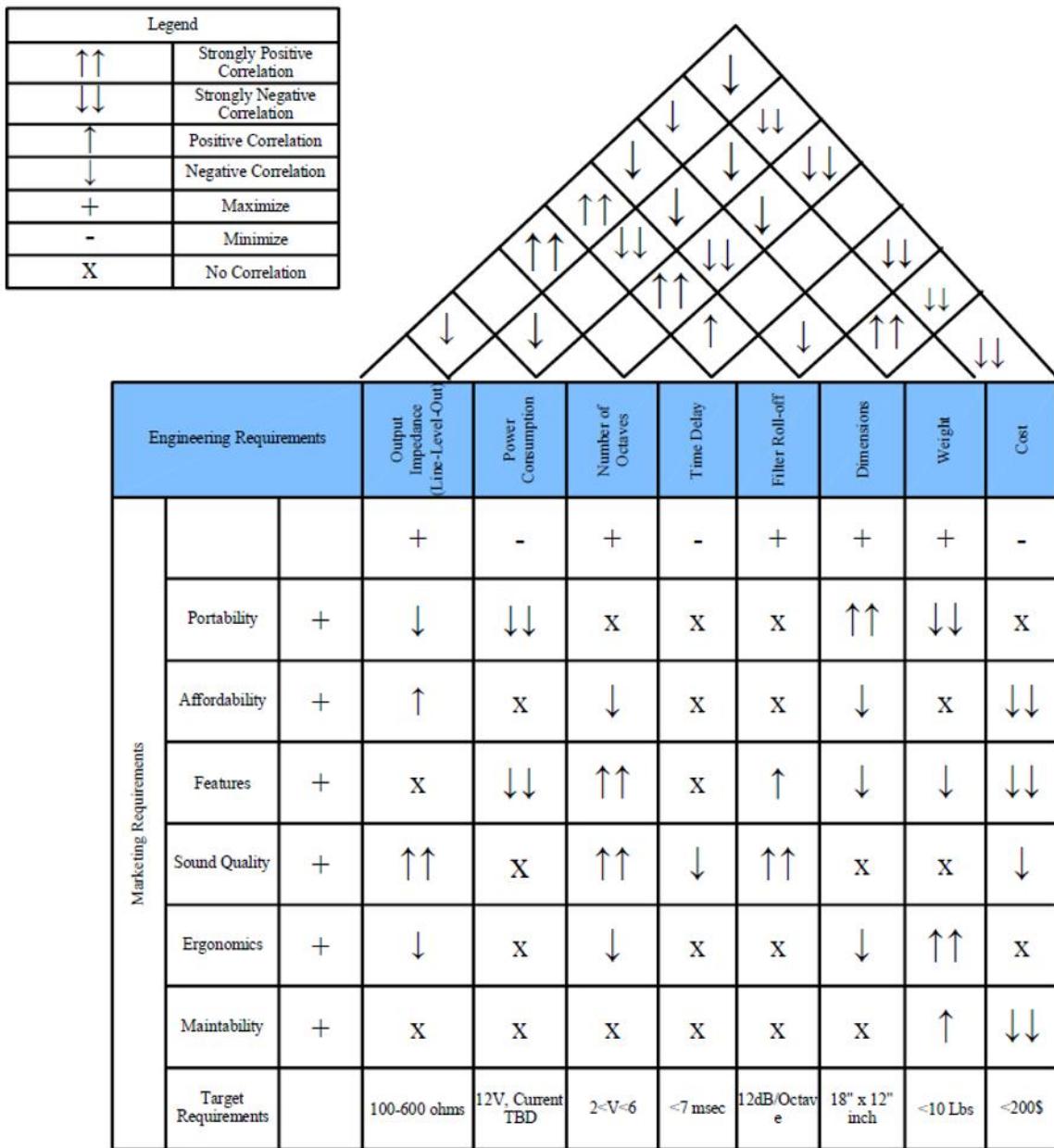
Parameters	Values	Marketing Requirement	Description
Max Dimensions	18" x 12"	1	L x W. System must be smaller than these dimensions.
Weight	< 8 lbs.	1	Weight of system.
Cost	< \$500	2	The total cost of the synth to the end user.
Time Delay	< 7ms	4, 5	Response time of system from user input.
SNR	> 40dB	4	The signal to noise ratio (SNR) should be within an acceptable range for entry-level hobbyists.
THD	< 2%	4	The ratio of total undesired harmonics to the fundamental harmonic should be in an acceptable range.
Filter Roll-off	12dB/octave	3, 4	Rate of roll off for configurable filters with key-tracking
Number of Octaves	8	3	Audible range of notes
Dynamic Range	~ 60dB	4	The ratio between the loudest sound and the noise floor should be in an acceptable range.
Frequency Response	30Hz – 7kHz	4	The range of frequencies the synthesizer can play.
Headphone output voltage	~2.5 V <sub>rms</sub>	3, 4	Voltage output for headphone jack in root-mean-squared voltage (V <sub>rms</sub> )

Headphone Output Impedance	16-50 Ωs	3, 4	Output impedance for headphone jack.
Line-Level Output Impedance	100-600 Ωs	3, 4	Voltage output for line-level out in Vrms.
Line-level output voltage	1-2 Vrms	3, 4	Output impedance for line-level out.
Power Consumption	< 25W	1, 2	Synth uses 12V for components and Neptune wants to limit to 25W maximum draw.

*Table 2.2 Marketing Requirements*

Requirement	Description
1. Portability	The system should be easily portable and small enough to fit inside a backpack.
2. Affordability	The system should be low-cost and targeted at entry-level enthusiasts.
3. Features	The feature should have the necessary features to get an enthusiast started.
4. Sound Quality	The system should sound well on its own.
5. Ergonomics	The system knobs should be easy to handle and not hinder the user's creativity in any way.
6. Maintainability	The internal circuitry and boards should be easily disassembled and accessed. It should also be made from parts in active production.

## 2.6 House of quality



**Figure 2.1: House of Quality Diagram.**

## 2.7 Hardware Block Diagram

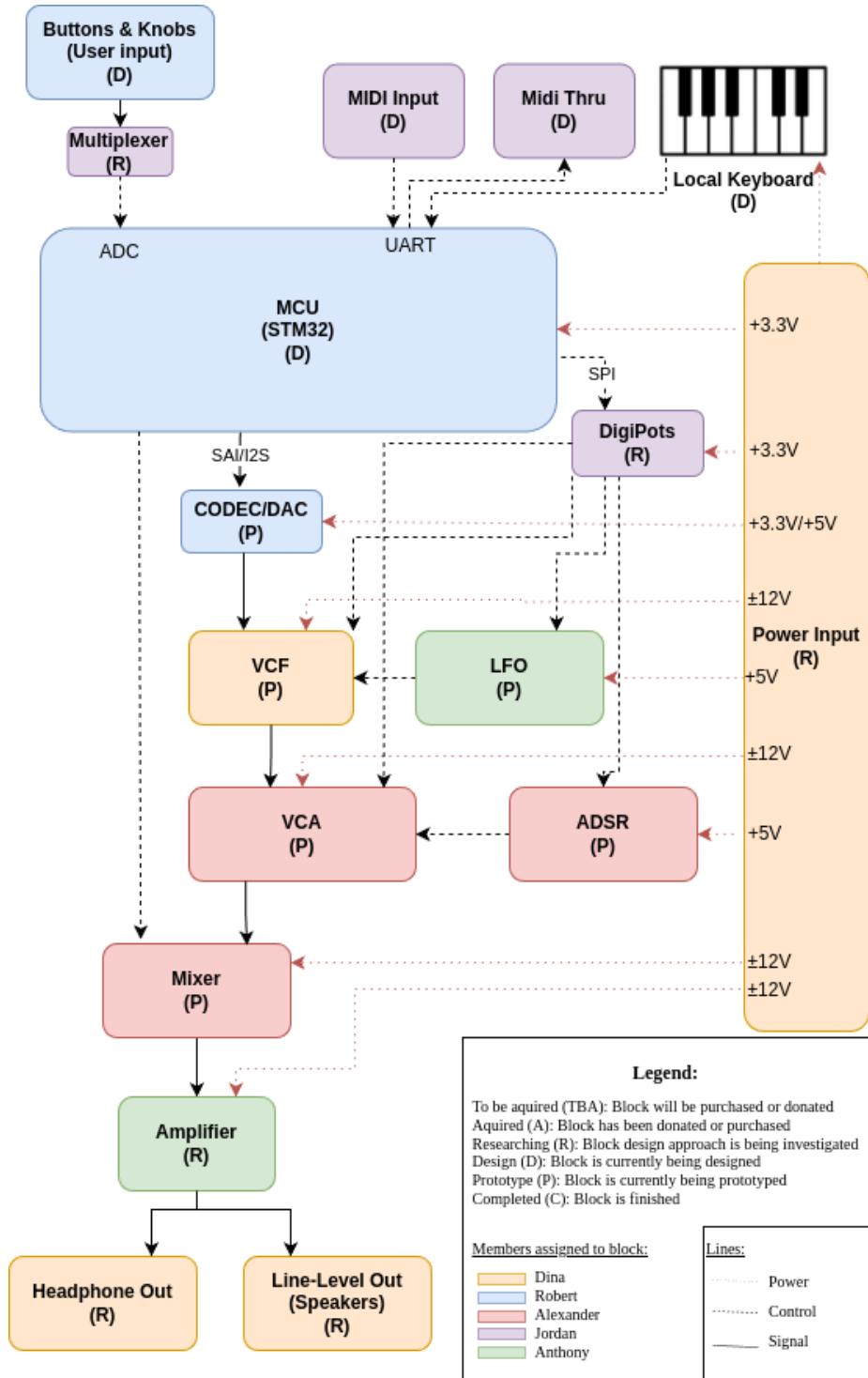


Figure 2.2: Hardware Block Diagram.

## 2.8 Senior Design II Hardware Block Diagram

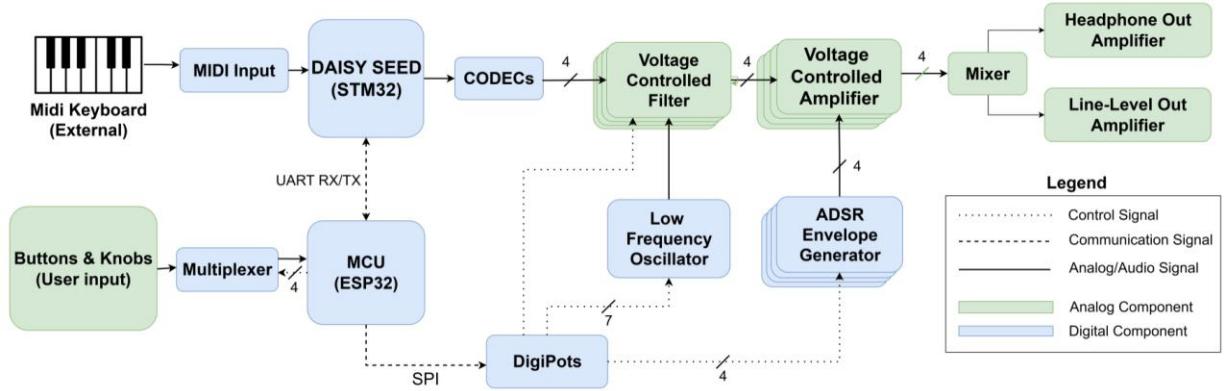


Figure 2.4: Finalized Hardware Block Diagram from SD2 with same power input

## 2.9 Software Block Diagram

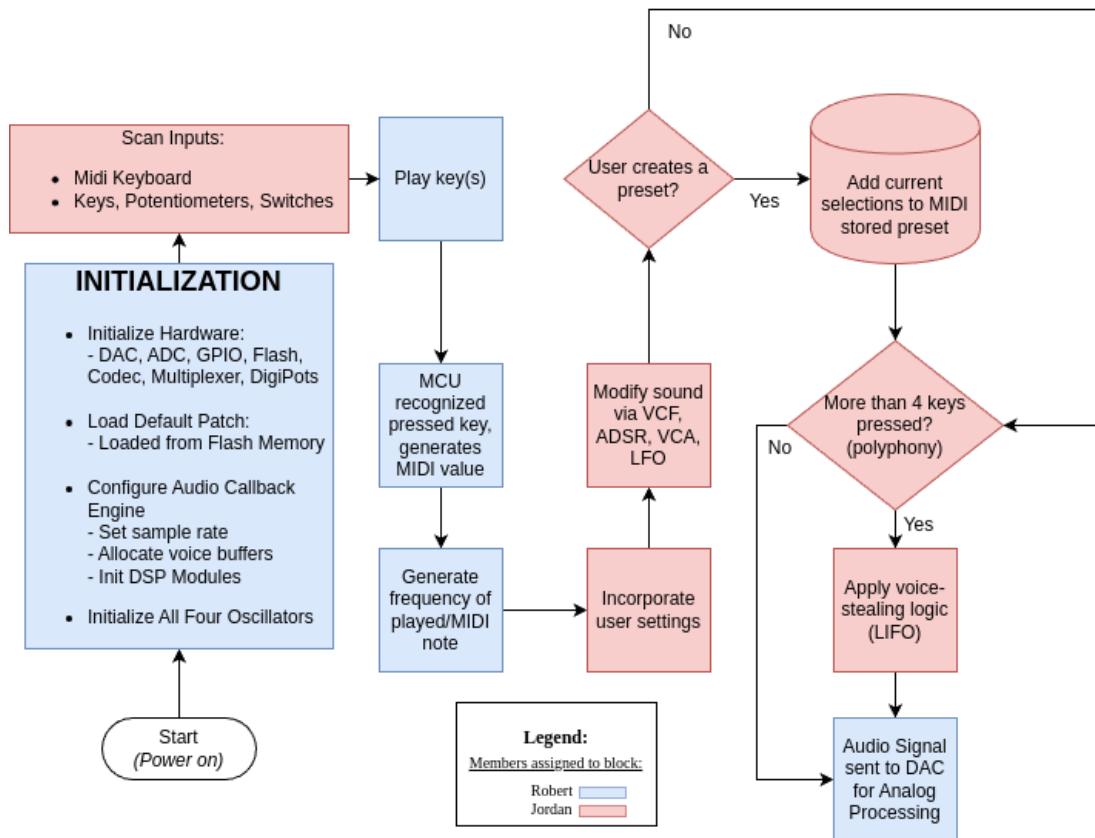


Figure 2.5: Software Block Diagram.

## 2.10 Prototype illustration/blueprint



**Figure 2.6:** Renders for the final product.

# Chapter 3 – Synthesizer Fundamentals

## 3.1 Oscillators

[3] Oscillators are the core component of any synthesizer, and typically the very first thing found in the signal chain. They are responsible for generating the tone heard by the user when they play a note. The sound heard can vary greatly depending on the architecture of the oscillators, what signal is chosen, and how it is being modulated. Oscillators are typically defined by two parameters: frequency and shape.

Frequency can be easily described as the pitch of the sound heard; that is to say, a high frequency can yield a sharp, shrill sounds, while a lower frequency can yield deep, rumbling bass notes. The frequency of the oscillators in a synthesizer is typically modulable by two interfaces: one with a coarse pitch that spans octaves, and a fine pitch that gives users a more granular approach to their tuning. This altogether grants the end user the ability to sweep through the entire audible frequency range, somewhere around 20Hz to 8KHz.

The oscillator's shape defines the timbre and harmonic characteristics of the sound. The most common waveforms found in synthesizers are the sine, triangle, square (or pulse), and sawtooth waveforms: each carrying with them their own tonal characteristics. A sine wave is the purest form, consisting of a single harmonic with no overtones, resulting in a smooth, rounded sound. Triangle waves are slightly richer, containing odd harmonics at lower amplitudes, which gives them a soft, flute-like quality. Square waves are harmonically complex, made up of only odd harmonics, producing a hollow, woody tone. Narrower pulse widths introduce even harmonics, creating a more nasal, biting texture. Meanwhile, sawtooth waves are the brightest and most aggressive, containing both even and odd harmonics, making them ideal for brassy leads or sharp bass sounds.

Our synthesizer will include two oscillators per voice, creating a wider and more complex range of sounds for the user to enjoy.

## 3.2 Voices

In synthesis, a voice refers to the number of individual notes or sounds a synthesizer can produce simultaneously. A monophonic synthesizer can only play one note at a time, where note priority determines which pitch is heard at the time of playing. Paraphonic synths, on the other hand, allow multiple notes to be played at once; however, they share a single amplifier and filter envelope. This means that while you get chordal playability, all effects modulating the signal (such as your envelope generation) only apply to the first note played.

Polyphonic synthesizers in contrast provide independent voices with each note played having its own dedicated signal path – this means that for each note played, it will go through its own filter, amplifier, and have its own envelope modifying it. This allows for richer, complex chords where every note responds dynamically.

### 3.3 Filters

Audio Signal filtering provides the user with another method of manipulating the sound of the standard function. In a synthesizer, what allows each waveform to produce its own unique sound is the timber of the function. The timber that dictates the sound is produced by the waveform's harmonics. Harmonics are best described as integer multiples of the fundamental frequency, and the fundamental frequency is chosen by the note which the user chooses to play. In the team's case, the lowest fundamental frequency is 30Hz, and the harmonics generated are based on the function chosen. For instance, a sawtooth waveform harmonic would be 30, 60, 90, 120 Hertz and so forth. These harmonics combine to form the sound of the waveform.

Therefore, when you apply a filter to the waveform chosen, the sound can change because the frequency range that is allowed is changed depending on the type of filter used. There are many types of filters used when filtering audio. Voltage controlled filters (VCF) are integrated circuits that contain multiple operational amplifiers(op-amps) that can be configured to create different wave filters.

### 3.4 ADSR (Attack Decay Sustain Release) Envelope Generator

The genesis of an envelope is a staple technique in music synthesis as it creates the opportunity for the user to customize up to 4 different values, increasing the range of possibilities for producible sound. Envelope signals are used to modulate the sound signal without altering the sound of the notes. The four values that NepTune can alter are Attack, Decay, Sustain, and Release. In concise language, this particular envelope generator is referred to as ADSR, an acronym for each of the four values presented.

An explanation of the ADSR's elements is required for understanding. Attack is a value that slows down the rate at which a sound reaches its peak amplitude, creating a volume effect. In analog iterations of ADSR, the attack is accomplished by charging a capacitor through a variable resistance. This means that higher resistance options (higher attack on user end) lead to a lower current charging the capacitor, which slows the rate at which peak amplitude is achieved. This can be represented in an example equation:

$$V(t) = V_{max} * (1 - e^{-\frac{t}{RC}})$$

As for digital ADSR envelope generators, another equation is used, because no discrete element is required to change the voltage entering the VCA, the only element that is variable is the voltage controlled alpha value ( $0 < \alpha < 1$ )

$$env_{next} = env_{current} + (Goal - env_{current}) * \alpha$$

Here you can see that a microprocessor can simply add the remaining amount of amplitude multiplied by its alpha value, which is what the user decides. This digital implementation

allows for a larger range of reliable rates of attack while also avoiding implementing extra discrete components, further simplifying the design.

Decay is the next value which oversees determining the rate at which the signal drops from its max amplitude down to the next value “Sustain”. Lower levels of decay will drop the signal significantly faster than a high decay value. So, if the team wants the sound to “jump” from no sound, up to its maximum, and then back down to a specified value quickly, the user should set low Attack and Decay. Low attack and decay combined can create a sharp percussion style sound due to the velocity of its change in amplitude while a combination of higher values creates a slow, ambient sound that also finds use in music synthesis.

The mathematical expressions for decay can be split up between linear and exponential as well, with RC components, the linear representation is as follows:

$$V(t) = V_{peak} - \left( \frac{V_{peak} - V_{sustain}}{T_{decay}} \right) * t$$

This linear configuration provides a straight-line descent to the sustain amplitude which is common with digital ADSRs however for analog synths like ours, NepTune can use exponential decay which can be represented for a suitable sustain level with the following:

$$V(t) = V_{sustain} + (V_{peak} - V_{sustain}) \cdot e^{-\frac{t}{\tau}}$$

The next parameter is far simpler as it is not time based like the other values but rather configured to the desired level. This value is Sustain, and with what the user sets its level to, is the amplitude that it will remain at after the decay stage until the key is no longer held down. With a Sustain of 0.5 or 50%, NepTune can expect the note being played to be half of its peak amplitude that was achieved by the attack and decayed into the sustain. This level will remain until the user either resets the ADSR by pressing a new key or until the user lets go of all keys being played.

The fourth integral value of the ADSR is Release; this parameter determines the time required for the signals amplitude to fall from the Sustain level down to 0% strength. Release has the special role of imitating reverb and acoustics sounds that are seen in string and pianos often. With an exponential, analog-style release the team can stick to the following equation which will show the user’s selection of value and its impact on the amplitude of the signal:

$$V(t) = V_{release \text{ (begin point)}} * e^{-\frac{t}{\tau}}$$

### 3.5 (VCA) Voltage Controlled Amplifier

Signal amplification will be the final gateway to the finished sound; amplification is often conducted immediately after the envelope is finished and the signal of the audio is combined with the envelope at this stage and modulated by the VCA (voltage-controlled amplifier). These amplifiers come in a few variations when concerning popular synth integration. Linear, Exponential and Dual/Quad VCAs are the iterations that are relevant to the design and will be explored later in the technology section of this component.

Generally, VCAs operate as volume modulators because their capability of amplitude modulation can quickly raise or drop the output strength of the sound signal. This function is more accurately described as “Gating” which mutes the output entirely. Connecting the envelope to the operational amplifier within the VCA will allow the shape of the envelope to determine the volume of the output signal according to how the user configures it using the four ADSR values. This would typically be the final step in the signal path; however, because the design implements multiple voices, NepTune will need to combine all the VCAs in the audio mixer.

### 3.6 Low-Frequency Oscillators

LFOs differentiate from the voices of the synthesizer by only operating at sub-audible frequencies, typically never surpassing 20Hz. What LFOs are useful for is modulation of the audible signals, and at different parts of the signal’s regular synthesis path, they can offer different functionalities. The number of possibilities just by modulating with LFOs is far too great to list all here, but some significant and relevant uses will be discussed.

Generally, the more that the LFO is connected to, the more features that will be available to the user to take advantage of. There are plenty of modulation effects necessary for detailed and intricate production of art and as a result, the more that are available, the more versatile the market will be for NepTune. One that will be included is Vibrato, which is a shaking sound that is produced as a result of connecting the LFO to the control voltage of pitch control. Another unique sound created is by connecting the LFO to the control voltage of the volume control, which synthesizes Tremolo.

The controllable parameters of the LFO that increases the depth of possible sounds is the waveform, selectable between sine, triangle, square and more similar to the voice oscillators. Frequency, while limited in range, does have a dramatic effect between the extremes of that scope (1 – 20Hz). Other more complicated customization takes the form of tempo-syncing and system defined intensity.

### 3.7 Control Voltages

Control Voltage is best described as the “language” of the synthesizer. IC’s or complete circuits will have lines that will configure parameters of operation based solely on the voltage present on a pin or line. This allows the designer to instruct the system to operate according to a variable voltage which can be inputted by a variety of ways. One such

example of this “language” is the keys of the synthesizer, Gate and trigger lines will interact with the keys to either send “on” or “off” signals based on connecting a voltage source to the control voltage input, either sending 0V or 5V. Other methods are also used, for example, the ADSR envelope generator present in NepTune’s design reads input voltage across four lines to determine the rate at which the amplitude of the resulting wave is increased or decreased. It can determine this by reading the voltage between 0-5V and internal components are designed to operate in a spectrum based on these values.

Control voltages are fantastic tools for controlling parameters with a simple tool being just voltage on a line which often can be manipulated using simple potentiometers. Especially in synthesizers, control voltage is the main way to interact with electronics and additionally can be controlled via digital chips like the Daisy Seed MCU integrated in NepTune.

## 3.8 MIDI

In the early 1980s, manufacturers wanted a simple way to standardize the growing amount of digital music hardware. Roland founder Ikutaro Kakehashi proposed the idea of a standard instrument language and attempted to provide a way to communicate all the features of a musical performance digitally [14]. Finally in 1982, MIDI was first developed as a digital language that controls and synchronizes electronic musical instruments and stood for Musical Instrument Digital Interface.

Essentially, MIDI works between pieces of hardware. The two devices transmit to one another which includes instructions on how to play a particular sound and other parameters such as notes, tempo, dynamics and effects [15]. It's also important to remember that these instructions are not audio; they're data that triggers sound.

Since its inception, the MIDI standard has gained the ability to control a wide variety of devices and it is important to understand what these devices control, and the types of MIDI connections that facilitate their communication.

### 3.8.1 MIDI Controllers

There are many types of MIDI controllers available that both send and receive MIDI messages. These controllers are essentially electronic musical instruments that emulate the sounds of the acoustic instruments they are physically based off of. The MIDI instruments take the form of piano keyboards, percussion/drum pads to emulate drum sets, guitars, and even wind instruments. Examples shown below:



**Figure 3.1:** Left to right: LEKATO Mini MIDI Keyboard, Akai MPCStudio2 Studio Drum Pad, Jamstik Standard MIDI Guitar, and Yamaha WX5 MIDI Wind Controller

On top of being able to send MIDI messages, MIDI controller can control parameters such as volume, panning, effects, and tempo. Further designing can effectively turn a controller into a full synthesizer; capable of real-time manipulation of sound using knobs, faders, buttons, and other hardware. However, most controllers with that level of capability are usually MIDI keyboards. While being just one type of MIDI controller, they are often preferred by many musicians and producers due to their unique combination of features and advantages.

### 3.8.2 MIDI Connectors

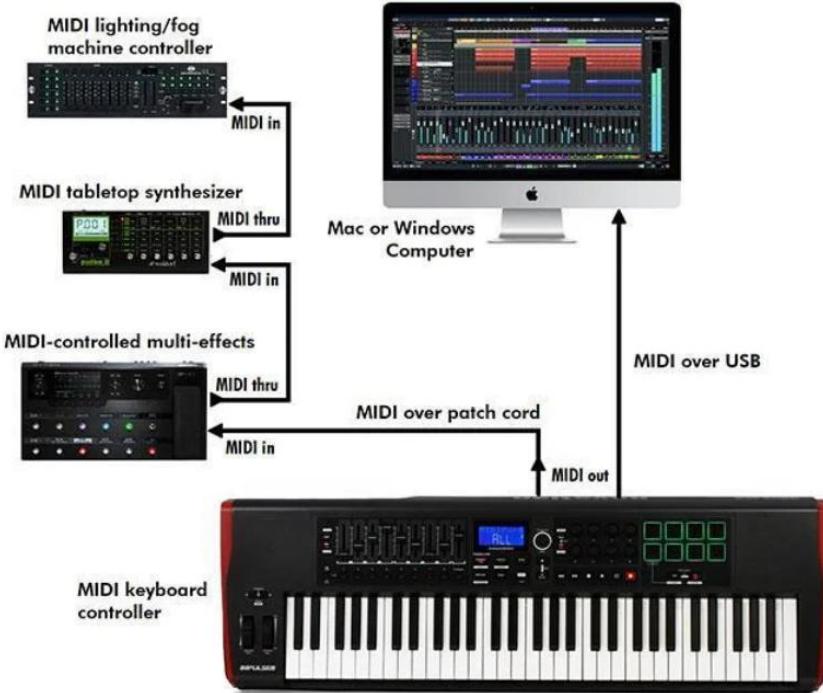
When it comes to MIDI messages, it all depends on the type of hardware used to facilitate the connection. There are a few ways to send MIDI messages, but the connections most common among synthesizers are 5-DIN MIDI connections and USB MIDI connections. Both send MIDI messages, but each is used in a different way and connects to different technologies.

#### 5-DIN MIDI

In the early days of its development, the MIDI protocol actually relied on standard 1/4" cables; the same type commonly used for audio signals. As the technology matured, however, it became clear that a more distinctive connection was needed. To help differentiate MIDI from audio and establish its own identity, the protocol adopted the now-familiar 5-pin DIN connector.

Most synthesizers typically include three 5-pin MIDI ports labeled In, Out, and Thru. These ports support one-way communication, with each serving a clear role: the Out port sends MIDI data such as notes and control messages, while the In port receives that data from another device. The Thru port passes incoming data from the In port to yet another device, allowing control of multiple instruments in sequence.

MIDI was developed before the widespread use of personal computers, and its biggest appeal at the time was the ability to control one instrument from across the studio using another. Connecting two synths is fairly straightforward, but as your setup grows, the complexity of routing MIDI signals increases. Additionally, while MIDI transmits data quickly, it's not instantaneous, and so chaining multiple devices through Thru ports can introduce noticeable latency between pressing a key and hearing a sound [17].



**Figure 3.2:** A representative MIDI setup example of a 5-DIN connection from a MIDI keyboard controller to multiple MIDI controllers via daisy chaining and a USB MIDI connection between the keyboard and a computer [18].

## USB MIDI

USB MIDI is a modern implementation of the MIDI protocol that allows digital musical instruments, controllers, and computers to communicate over a standard USB connection. Unlike traditional 5-pin DIN MIDI, which uses a dedicated serial hardware interface, USB MIDI transmits the same MIDI messages (note on/off, control changes, program changes, etc.) over USB using a specific class-compliant protocol. This allows USB MIDI devices to be recognized by operating systems without the need for custom drivers, provided they follow the USB Audio Device Class specification with MIDI support.

USB MIDI works by encapsulating MIDI messages into USB packets, which are sent over a USB bus between a host (typically a computer) and a peripheral device (like a MIDI keyboard or synthesizer). Because USB supports much higher data rates than legacy MIDI, USB MIDI can support multiple MIDI ports (called "virtual cables") over a single connection and allows for much lower latency and higher bandwidth communication.

USB MIDI is widely used in computer-based music production environments, embedded synthesizer projects, MIDI controllers, drum pads, and digital audio workstations (DAWs). It's especially common in compact or modern gear where USB ports are more practical than traditional 5-pin MIDI ports. Despite lacking the electrical isolation of DIN MIDI, USB MIDI is favored for its plug-and-play convenience, ease of use, and ability to integrate tightly with modern computing systems [17].

# Chapter 4 – Research and Investigation

The following chapter illustrates the research behind decisions made for this project. It is broken down into two sections. The first section covers design choices affecting the overall project, such as the type of musical synthesis to be implemented. The second section explains the more specific choices, such as those pertaining to the filters.

In each of these sections, research will be conducted on some of the available technologies. The advantages and disadvantages of each technology will be evaluated as it applies to *NepTune*. In addition, the second section also includes a portion where specific parts (with part numbers) are evaluated after the technology is chosen. These parts are compared with one another so that the best part can be utilized in this project.

## 4.1 Synthesizer Design Methodology

In this first section, the team will first establish a few foundational design choices before diving into specific subsystems. These include: the type of synthesis to employ, whether the project will utilize analog or digital circuitry, and whether to focus on custom circuit design or utilize available integrated circuits (ICs). Making these decisions sooner will narrow down the research scope, align team efforts, and reduce delays during the rest of the research phase.

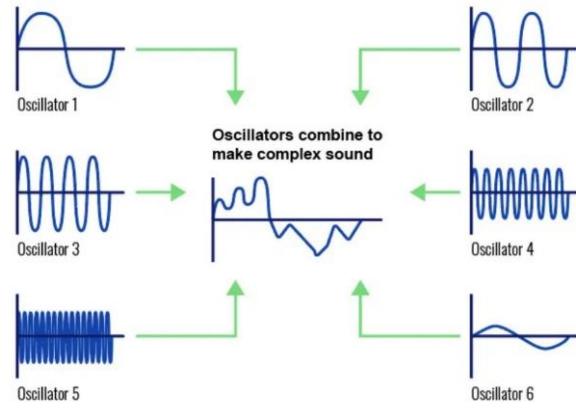
### 4.1.1 Types of Synthesis

The first and most important decision to make is the type of musical synthesis to employ in *NepTune*. To make the best decision it is important to understand the various types of synthesis. The type of synthesis implemented affects the entire design of the synthesizer as well as the resulting sound. It is imperative to choose a type of synthesis that provides an optimal balance between musical capability, design feasibility, and realistic value within the project scope.

#### Additive Synthesis

Additive synthesis, as the name implies, mathematically combines pure sine waves, each with independently controllable frequency, amplitude, and phase, to create harmonically rich and complex sounds. This approach of synthesis implements concepts from Fourier's theorem, which states that any sound can be built up of sine components. While additive synthesis is theoretically capable of replicating any acoustic instrument or sound, the practical implementation of the circuitry requires dozens to possibly hundreds of precisely tuned oscillators and complex modulation systems. This makes additive synthesis computationally intensive and circuit-prohibitive for hardware projects with cost and

timeline constraints, as each oscillator demands dedicated analog circuitry or significant digital processing power.



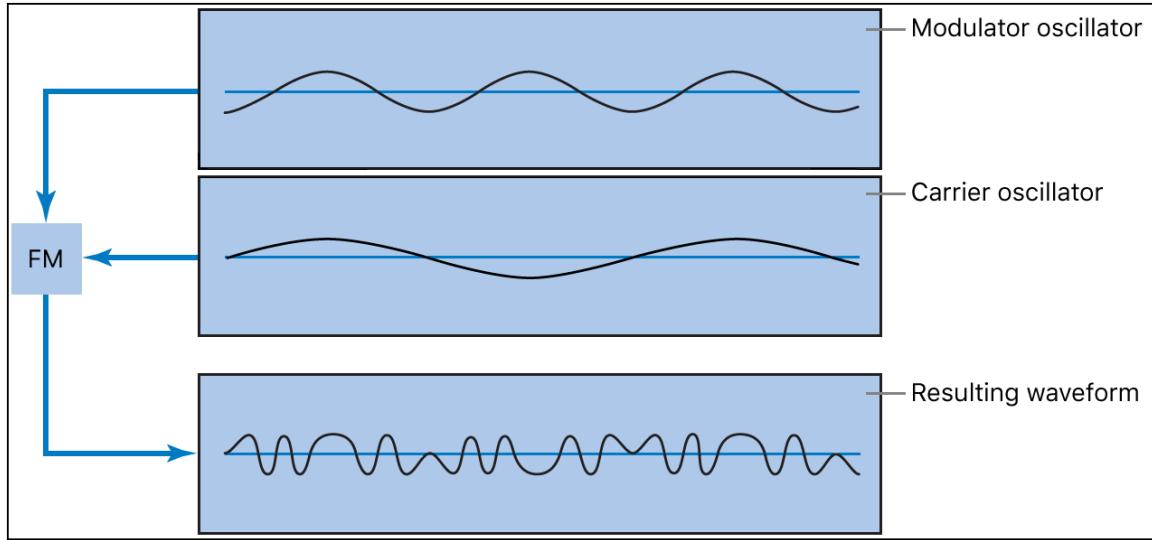
**Figure 4.1:** Additive synthesis adds multiple sine waves to create one complex waveform.

### Frequency Modulation (FM) Synthesis

FM synthesis generates complex sounds by using one audio signal (the *modulator*) to rapidly alter the frequency of another signal (the *carrier*), governed by the equation:

$$\text{Output} = A \sin(2\pi Ct + D \sin(2\pi Mt))$$

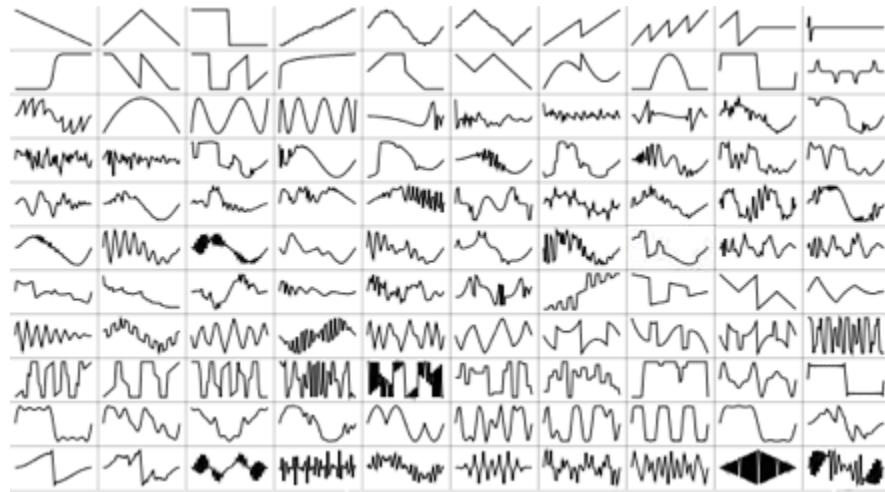
This equation is derived from the formula for a simple sine oscillator and adjusted to accommodate for a time-varying frequency. In this formula, C is the *carrier*, D is *depth of modulation*, and M is the *frequency of modulation*.  $I = \frac{D}{M}$  is the *index of modulation* [26]. This form of synthesis is perfect for creating metallic, bell-like, and gritty tones through sideband frequencies that are not harmonically related to the original pitch. While FM synthesis allows for a diverse range of complex sounds, it is impractical to implement on an analogue level due to tuning instability and temperature sensitivity, relegating it to only being practical for fully digital synthesis. Digital implementations also require real-time DSP algorithms, adding complexity beyond the scope.



**Figure 4.2:** FM Modulation consists of modulator and carrier waveforms.

### Wavetable Synthesis

Wavetable synthesis cycles through snapshots of an array of different waveforms (“wavetables”) stored in memory, often morphing smoothly between them. This technique excels at creating animated, evolving sounds, but entirely relies on DSP to facilitate the transformations [27]. Wavetable synthesis requires high-speed memory and interpolation logic, which draws focus away from the analog design of the synthesizer; however, wavetable synthesis does have the benefit that it can adapt to other paradigms. Wavetable synthesis is compatible with subtractive synthesis, which is ultimately the form that the team has decided to represent the project.



**Figure 4.3:** Wavetable synthesis consists of a table of waveforms to morph between.

## Subtractive Synthesis

The basis of subtractive synthesis involves the selection of harmonically rich soundwaves, such as a sawtooth, square, or pulse waves, and selectively removing frequencies from them through the use of filters and morphing them with envelope generators. Subtractive synthesis involves the least manipulation of the original signal generated relative to the other forms of synthesis, giving it a key edge in the simplicity involved for its implementation. Subtractive synthesis can be easily modeled both digitally and analogue, providing it another win in its ease of availability. While subtractive synthesis can be particularly dull and uninteresting on its own, it benefits greatly from the ability to be modulated and tuned with ease. Layering voices and other complex waves grants it incredible potential. The many upsides and its relatively few downsides were the ultimate factors that led us to decide on subtractive synthesis for the modeling of choice.

### 4.1.2 Analog vs Digital Circuitry

The second foundational design choice for *NepTune* is to decide whether to implement the synthesizer architecture using analog circuitry, digital circuitry, or a hybrid of the two. This design decision affects every subsystem of the synthesizer and has a direct impact on sound quality, user experience, and development feasibility. Nowadays, digital technology supersedes analog technology in many ways. Almost everything that can be done via analog circuitry can be replicated digitally. Following natural course of reasoning one might ask “why implement analog circuitry into *NepTune*?”

Before this question can be answered, it is important to examine and compare analog and digital circuitry. Only then will the balance of this hybrid synthesizer make sense. The criteria used to weigh the effects of design options is listed below:

- Polyphony: Design must support multiple voices.
- Sound Variety: Support variety of waveforms and features for creative expression.
- Ergonomic/ Ease of use: System must be simple and easy to use
- Low cost: Final product should be < \$500.
- Small size: Final product must fit inside a backpack.
- Maintainability: Components should be in active production.
- Ease of design: Final product should be reached in < 5 months

When choosing which system will be analog or digital, the criteria above will be kept in mind.

### Analog Circuitry

Like most other types of technology, early synthesizers were constructed entirely with analog circuitry. This circuitry utilized multiple analog components such as resistors, capacitors, transistors and op-amps, with very early synthesizers even using vacuum tubes. These analog systems formed the foundation of electronic sound synthesis, and many of their design principles are still in use today.

For *NepTune*, analog circuitry offers several benefits. A key advantage to analog systems is low latency. These systems operate in real-time without the need for digitization and instruction processing. Additionally, analog signals are inherently smooth and continuous. This results in a waveform free of artifacts caused by quantization and aliasing. Analog design also involves topics covered in coursework such as Linear Circuits I & II and Electronics I & II. These classes provided the team with the hands-on experience necessary for this area of design.

Unfortunately, there are many difficulties that make analog synthesis less practical for *NepTune*. First, there are general challenges affecting all analog designs and specific challenges that may only be encountered when designing synthesizers.

The general challenges are more well-known. The first of these is slow prototyping. Design of such circuits can take time as they often must be simulated or physically laid out and tested. In some cases, the condition of test equipment must be taken into consideration, as broken or malfunctioning equipment can cause delays if not properly maintained. Additionally, analog circuitry requires a lot of hardware and can quickly use up board space. For a design which must be compact and portable, this is less than ideal.

The functionality of a synthesizer can also be limited by the complexity of analog design. As more features are implemented, the difficulty of circuit design can quickly rise beyond what the team is prepared for. Combine this with other issues such as noise and temperature sensitivity, and the final product may be un-attainable.

Alongside these broader challenges, there exist challenges that may be particular to synthesizer design. Among these is the difficulty of scaling up the number of voices. Since polyphony is a primary feature of *NepTune*, the hardware that makes up one voice must be scaled up to reach the required voice count. As more voices are implemented, more hardware is required. While duplicating circuitry may seem straightforward, achieving consistent performance across multiple voices is quite complicated. This is because the system is a musical instrument and has tight tolerances. The components that make up each voice circuit, or voice card, must be almost identical to the other voice cards. Any variation can lead to voices sounding out of tune with one another. This is especially important for bipolar junction transistors (BJTs) which can vary wildly in gain from one another. To mitigate this, the components on each voice card must be hand-measured and binned accordingly. While possible, this procedure is tedious and time consuming as every component must go through this process.

Analog oscillators are also prone to pitch instability due to temperature, humidity, and power supply fluctuations. While some variation is tolerable, and even desirable in some musical contexts, excessive drift can render an instrument unusable. To combat this, many analog synthesizers use temperature compensation circuits, which attempt to correct pitch shifts dynamically. However, these add further complexity and require careful design.

Analog circuits are also susceptible to electrical noise, particularly when placed near digital systems. High-frequency switching from microcontrollers, DACs, or logic ICs can couple into sensitive analog paths, causing clicks, hums, or high-frequency artifacts. Proper

filtering, shielding, and PCB layout are necessary to mitigate these issues. However, this may increase the time required to realize such a design.

Finally, analog synthesis does not inherently support the use of presets, which is a core feature of many modern synthesizers. This means that when a musician finds a board or parameter configuration they may like, they have no way to save it for later. Once the positions of knobs or sliders are changed, the setup is lost forever unless the musician is able to write and recall the position of every knob. This limits usability and performance flexibility.

## Digital Circuitry

With the advent of the microcontroller, electronic circuitry has developed at an incredible rate. Digital circuits operate using discrete, quantized signals. Quantization is among the key advantages of digital circuitry, supporting systems such as logic and memory. While this technology poses its own set of challenges for musical synthesis, its advantages may be worth the effort.

One of the primary benefits of digital circuitry is its ability to support a wide variety of synthesis methods. These include physical modeling in which each waveform is meticulously calculated, often with fundamental sine and cosine functions. Another example is wavetable synthesis where two waveforms are stored in memory via a lookup table and smoothly morphed between. A third method is direct digital synthesis (DDS), in which a waveform is stored in memory and iterated through at varying rates to produce different frequencies.

Like most other technologies, musical synthesis can greatly benefit from the characteristics of digital circuitry. Besides supporting more complex, programmable waveforms and modulation schemes, the oscillators in digital circuitry do not face issues with tuning drift or instability. These oscillators often rely on quartz crystals known for exceptional stability. In addition, prototyping is rapid, and low-cost, since all that is needed is a firmware flash to realize different logic. Digital circuitry also offers smaller board size, since most of the signal processing and calculations can be done on the microprocessor.

Perhaps the most significant advantage of digital circuitry in musical synthesis is the ease in which the number of voices can be scaled. It is not uncommon for digital synthesizers to have upwards of 5 or 6 voices. Because the oscillators in digital synthesis are stable, and rely on quantized values, component matching across voices is not required like it is for analog circuitry. This drastically cuts down on development time, making this one of the most appealing benefits of digital synthesis.

Despite the plethora of benefits, digital circuitry still has some challenges to face if implemented in *NepTune*. First off, software development is not a core strength of the team, so the benefits of digital synthesis rely on the availability of software libraries and documentation. While this will alleviate the amount of custom code that is required to be developed, it also retrains us to the capabilities of existing libraries.

Despite being relatively self-contained, microcontrollers may need an external digital-to-analog converter (DAC) or a low-pass filter for pulse-width-modulation (PWM) to

generate an analog signal. This can drive up costs and increase complexity. Additionally, PWM-based output can introduce switching noise, and filtering may not fully eliminate artifacts resulting from switching or quantization. Digital circuitry also introduces latency from instruction processes, communication protocols, and signal conversion overhead. Furthermore, while rapid prototyping is a notable advantage, debugging can be more challenging, particularly when dealing with logic errors, timing issues, or input/output (I/O) misbehavior.

### Decision on Type of Circuitry to Employ:

After assessing the numerous advantages and disadvantages of analog and digital synthesis, the team ultimately came to the decision on a hybrid system. There are a number of reasons that led to this decision. The table below provides a visual aid of the logic behind this choice.

*Table 4.1: Pros and Cons for Analog and Digital Circuitry*

	Analog Circuitry	Digital Circuitry
Polyphonic capability	Poor	Good
Sound variety	Limited	Good
Noise sensitivity	Poor	Good
Oscillator stability	Poor	Good
Team capability	Good	Limited
Prototype difficulty	Manageable	Manageable
Small size	Manageable	Good

After considering the team's capabilities, project timeline, and budget, a hybrid synthesizer architecture was selected as the most practical and effective solution for *NepTune*. This decision was primarily driven by the team's desire to support polyphony, as well as the need to work within the team's existing technical experience.

Polyphonic synthesis needs to be a defining feature for this project not only because of its rich sound, but also to differentiate *NepTune* from the previous Senior Design project, Synthia, which had already explored monophonic synthesis. Additionally, given the condensed Summer-Fall schedule, the team decided that digital circuitry was the most feasible way to attain polyphony.

After weighing in all the options, it was decided that the best way to implement polyphonic capabilities was to utilize the performance overhead of the microcontroller to generate physically modeled digital oscillators rather than using analog voltage-controlled oscillators. This decision was made primarily due to the difficulty that would arise with

ensuring the stability of analog oscillators. These digitally generated, mathematically controlled voices will have no issues with tuning or stability and can be scaled upwards depending on the scope of the project. The variety of waveforms that *NepTune* would be able to generate would be essentially limitless, and software updates to its generation algorithm would allow us to add interesting capabilities to it in the future.

At the same time, the team emphasized preserving an analog signal path beyond the oscillator stage. This would include the filters, amplifiers, and signal mixers. This choice supports two major objectives. First, it provides the team with valuable experience in analog circuit design. Second, it breathes some character into *NepTune*, showcasing the capabilities of traditional analog signal processing.

#### 4.1.3 Discrete Circuits vs Integrated Circuits

With the decision to pursue a digital-analog hybrid architecture firmly established, the final overarching design consideration remains: Should the team focus on developing custom circuitry with discrete components for the analog signal path or utilize integrated circuit (IC) solutions that consolidate these functions into ready-made components? This choice has significant effects on the complexity and flexibility of *NepTune*. Custom circuits with discrete components offer opportunities for deeper learning and finer control over performance, but integrated circuits can reduce design time, save board space, and simplify debugging.

Custom analog circuitry relies on discrete components such as resistors, capacitors, transistors, and operational amplifiers (op-amps). While each component is individually small, assembling them into functional circuits takes up more space on a printed circuit board (PCB) and increases routing complexity. In contrast, integrated circuits (ICs) utilize solid-state technology to recreate these discrete analog components on a circuit contained on a single silicon die. As a result, an IC can often replace a circuit that would otherwise require several discrete components, leading to significant reductions in board size and component count.

Each option has its own benefits and drawbacks. In the context of *NepTune*, several factors will be taken into consideration. These include design feasibility, noise sensitivity, temperature stability, and component availability.

Designs with discrete components inherently offer more control and flexibility. The designer is free to modify the circuit according to his or her specifications. On the other hand, ICs are often designed with a specific application in mind. While this can be beneficial if the designer's intended application aligns with a user's target application, it can also sometimes be difficult to find such an IC. As a result, the user is often limited to the functions of ICs available on the market.

In the context of Senior Design, custom designs with discrete components offer a great learning opportunity, as it would enable the team to gain a deep understanding of analog circuit design. However, since the analog signal path consists of several voice cards in

parallel with one another, component matching becomes a major issue [49]. If a design utilizing discrete components is chosen, each voice card must be identical to the next and will require components to be measured and matched. As a result, not only will designing custom circuitry be difficult, but scaling up the number of voices will also pose a challenge.

In contrast, IC circuitry will alleviate the need for component matching, since components are constructed on the same die. This streamlines the manufacturing process and may be worth sacrificing the flexibility offered by designing custom circuitry.

Noise sensitivity is also an important consideration in this design. Because it is a musical instrument, components must be resilient against noise. Unfortunately, discrete components offer little resistance against noise. Large part sizes and long trace lengths are susceptible to external electromagnetic interference (EMI) and high frequency switching noises from nearby digital circuitry (e.g. the microcontroller). IC designs, however, may offer better noise resilience due to their smaller trace lengths and overall size [45].

While analog components are inherently sensitive to temperature changes, discrete components and integrated circuits respond to temperature in different ways. Discrete components typically dissipate heat more effectively due to their large footprint and distance from one another. However, these circuits can experience uneven temperature gradients, leading to inconsistent behavior from part to part. This can produce negative effects on the voice cards where components must be matched with one another. In contrast, integrated circuits often suffer from poor heat dissipation. However, since components are densely packed within an IC, the temperatures can rise consistently throughout it [44]. This may be a benefit, since uniform temperature rise can allow designers to predict thermal drift and inform users of its effects on performance. Although thermal drift can have drastic effects, a small amount is acceptable and is a defining characteristic of analog synthesizers.

Lastly, component availability is a major concern. While there is never a shortage of discrete analog components, integrated circuits are more specialized, and availability is often limited. This is true especially for niche designs such as synthesizers. When deciding on parts, the team must take into consideration whether the IC is suitable for audio applications. In contrast, an analog circuit made up of discrete components can be designed to meet these audio application requirements.

### **Final Decision on Analog Signal Path Construction**

After assessing the advantages and disadvantages of each, the team ultimately decided on implementing IC technology where possible. A table is provided below to aid in understanding this reasoning.

**Table 4.2: Pros and Cons for Topologies Involving Discrete or Integrated Circuits**

	Discrete Components	Integrated Circuits (ICs)
<b>Design feasibility</b>	Poor	Good
<b>Noise sensitivity</b>	Poor	Good
<b>Temperature stability</b>	Moderate	Moderate
<b>Component availability</b>	Good	Moderate

The primary factor driving this decision was design feasibility. Developing a fully custom analog audio signal path can be a complex and time-intensive task. This is particularly true since these circuits require careful tuning and layout to ensure low noise. Given Senior Design's tight timeline (less than five months) the project needs to prioritize simplicity and reliability in implementation.

Implementing ICs also reduces the amount of time required for construction. It eliminates the tedious component matching that would be required to scale up discrete analog component designs. Additionally, ICs are small in size and work well for portable applications.

For these reasons, utilizing integrated circuits in the analog signal path offers the best practical solution. Realistically, the team does not expect to find an IC for every application necessary in the audio signal path of *NepTune*. As a result, the team will attempt to find suitable ICs, but if not, the team is prepared to design custom circuitry where necessary.

## 4.2 Hardware Parts and Technology

Now that the important overarching framework has been decided, it is time to examine each specific subsystem. The research for each subsystem will follow a specific process. First, relevant technologies are explored. These technologies are the ways in which the subsystem can be realized. Once a technology is chosen, specific parts will be compared with one another. The part that best suits the project constraints will be selected. For example, to realize a filter, several topologies may be explored such as the Sallen-key filter and notch filter. Once the topology is chosen, specific parts will be examined.

### 4.2.1 Digital Processing System

At the heart of *NepTune*'s hybrid architecture lies a critical design decision: the selection of what Digital Processing System will be responsible for handling core functionalities such as oscillator waveform synthesis, MIDI interpretation, control voltage (CV) generation, and preset management. When handling all these functions, it is important to evaluate parameters such as latency, computational headroom, and developmental complexity. In addition, scalability and power consumption must be analyzed, as they

pertain to the rest of the synthesizer design. Regarding *NepTune*, the following criteria are to be considered:

- Latency:  $\mu\text{s}$  resolution,  $<10\text{ms}$  acceptable
- DSP capabilities: required for oscillators and potential effects
- Daisy Library Compatibility: programmable with available open-source libraries
- Cost Effective: Necessary for project scope and prototyping capabilities
- GPIO Flexibility:  $>30$  pins for various control matrices
- CV Control Precision:  $1\text{mV}$  resolution preferred
- CODEC Compatibility: I<sup>2</sup>S/SAI compliant

### Digital Processing System Technology Research

When looking into the different Digital Processing architectures that could be implemented for *NepTune*, there were three that stood out that could provide all the criteria needed, with different caveats for each.

#### **Microcontrollers (MCUs)**

Microcontrollers are specialized ICs that combine essential components such as a processor core, memory, programmable I/O peripherals, and various other components onto a single chip. They are the foundation of embedded systems and provide control over various electronic systems and applications. They can be programmed in a variety of easy-to-use programming languages like C, C++, and Python, and are widely available and affordable. MCUs like the STM32H7 (e.g., Daisy Seed) excel in applications such as *NepTune* due to their accurate processing, low power consumption, and time-sensitive audio processing [28].

MCUs can deliver exceptional real-time performance with latency consistently attainable below 5ms, making them an ideal choice for music and rhythmic-based applications. Their various peripherals, such as high-resolution DACS, ADCs, and 5V-tolerant GPIO pins simplify analog circuit interfacing [29]. Development is accelerated through the use of mature programming ecosystems, such as the STM32Cube and Arduino Libraries. Power efficiency ( $<500\text{mW}$ ) and cost-effectiveness remain key advantages as well, with units such as the STM32H750IB costing as little as \$25.

On the other hand, MCUs carry the least sequential processing power of the chosen digital systems. MCUs such as the considerably fast MIMXRT1062DVL6A clock in at around 600MHz but may struggle with any synthesizer over 8-voices, especially if further DSP processing is implemented for effects or other forms of physical modeling. Memory constraints could also arise when working with many presets, or if wavetable synthesis was considered [30]. Such issues could be resolved through the use of external memory, however.

#### **Field-Programmable Gate Arrays (FPGAs)**

Field-Programmable Gate Arrays are reconfigurable integrated circuits that are composed of programmable logic blocks interconnected with customizable logic blocks and flexible interconnects [31]. They implement custom digital circuits defined through hardware description languages (HDL) such as VHDL and Verilog and enable massive parallel task execution with relatively few resources. With synthesizers, FPGAs dominate in high voice count and DSP-heavy applications such as wavetable, granular, and additive synthesis.

Parallel architecture allows for incredible voice density, with synthesizers such as the Waldorf Kyra reaching an astonishing 128 voice polyphony, each with 10 oscillators per voice [32]. Hardware reconfigurability allows for algorithmic updates without physical modifications, which grants FPGAs an enormous benefit when it comes to the capabilities of software updates. Some FPGAs are even capable of nanosecond-level precision and latency [33].

Despite its many advantages, FPGAs are also riddled with their own disadvantages. Development complexity increases significantly, as HDL expertise is needed, and long stretches of design verification would be required to ensure accuracy. Power consumption poses another issue, as typical FPGAs draw around 1-2W, which may impede the design philosophy of building something small and portable. Debugging will also prove exceptionally difficult compared to MCUs that have easier languages to program in, and the lack of available software libraries with the necessary DSP and physical modeling functions makes the package inaccessible [30]. Pricing is also an issue, with FPGAs being the most expensive of the choices.

### **Single-Board Computers (SBCs)**

Single-Board Computers (SBCs) are integrated computing platforms incorporating a microprocessor, RAM, storage, and I/O interfaces onto one PCB. Unlike microcontrollers that typically run bare-metal firmware, SBCs, typically running general-purpose operating systems like Linux on multi-core CPUs. Devices like the Raspberry Pi leverage these benefits for high-level computing tasks [30]. In synthesizer contexts, SBCs typically handle things such as non-time-critical tasks such as preset storage or display interfaces, but they are capable of many other use cases.

SBCs provide an order of magnitude greater computational power than MCUs, enabling features that could be overcomplicated for them such as Wi-Fi connectivity, advanced UI rendering, and greater processing power. They are typically easy to program for, with High-level languages such as Python and C++ being commonplace for development, and libraries for developing with them are easily available for all to use. SBCs are often used in synthesizers as auxiliary processors where they excel in handling non-real time tasks.

Unfortunately, non-deterministic OS scheduling can introduce a great amount of variable latency ( $>10\text{ms}$ ), making them unsuitable for use in real-time instrument use, and especially in synthesizers. Unlike MCUs with native 5V-tolerant GPIOs, some SBCs may only be 3.3V tolerant, necessitating level-shifters which may overcomplicate design and introduce noise. While SBCs simplify high-level features, they also come with higher power consumption than MCUs.

**Table 4.3: Pros and Cons for DSP System.**

	MCU	FPGA	SBC
<b>Latency</b>	1-5ms	<0.1ms	<10ms Unreliable
<b>Development</b>	Easy with library support	Very difficult	Moderate
<b>Cost</b>	\$10-30 Affordable	\$50-200 Expensive	\$15-100 Affordable
<b>Performance</b>	Moderate	Overkill	Good
<b>Power Consumption</b>	<500mW	1-5W+	3-10W
<b>Analog Integration</b>	Easy, direct control	Difficult PCB design	Moderate complications
<b>Fits project constraints</b>	Yes	With Difficulty	No

### Final Choice of Digital Processing System Technology

Considering all the options, choosing an MCU for the project is the clear choice. MCUs offer the most benefits, match all the criteria needed to design the synthesizer, with their only prevalent downside being that they lack the performance of the other two options; a downside that does not affect the design due to the scope of the project being well below the performance ceiling.

#### Digital Processing System Part Selection

Now that a Microcontroller has been decided on for the Digital Processing System of *NepTune*, an analysis of the various options can be made. When looking for what MCU to use for the project, the following options are considered:

- Software Support: Must have well-documented library support.
- Library Support: Compatible with libDaisy Ecosystem
- Large Package Size: >0.5mm pitch preferred, easily solderable
- GPIO Specifications: 5V tolerance, large pin count
- Analog Peripherals: Internal DAC and ADC Support
- Audio Peripherals: I<sup>2</sup>S/SAI support
- Flash Support: Internal, with external support
- Power Consumption: Under 500mW
- Low Cost: Under \$15.

## **MIMXRT1062DVL6B**

The MIMXRT1062DVL6B from NXB features a fast 600MHz Cortex-M7 core and 1MB of on-chip RAM; however, it contains no internal flash, requiring external QSPI memory for program storage. It targets high-performance applications but lacks a few analog interfacing components such as an integrated DAC that could position it lower than the other options.

The 600MHz clock speed of this MCU is the highest of the options and provides substantial headroom for a wide range of DSP algorithms beyond the scope of the basic synthesis *NepTune* is planned to utilize. The memory interface supports high-speed external memory up to 166MHz. Analog peripherals include internal Analog-to-Digital Converters (ADCs), 16-channels each, for a total of 20-channels active at a time.

There are, however, specifications that have made the MIMXRT1062DVL6B a suboptimal choice for *NepTune*. Absence of internal DACs necessitates external CV-generation ICs, which may hinder the analog integration goals. GPIOs are limited to 3.3V logic, which further complicates interfacing with the many analog components that function at  $\pm 5V$ . Finally, this MCU is only available in the 196-LFBGA package, which demands dedicated reflow equipment and is incredibly difficult to solder and troubleshoot by hand. No support for the planned libDaisy hardware library further complicates design and extends timelines, as it forces custom driver development.

## **STM32H743II**

STMicroelectronics' STM32H743II pairs a 480MHz Cortex-M7 core with 1MB of RAM and up to 2MB of internal flash memory. Its Quad-SPI memory interface supports running external memory up to 133Mhz and sports a 32-bit external memory controller for external RAM. It offers a large number of analog peripherals and is available in multiple different package sizes.

This MCU's generous 2MB of flash greatly simplifies preset storage without the need for including external memory, allowing *NepTune* the ability to potentially store all the complex patches it can directly on-chip. Dual 12-bit DACs enable control and modulation of analog components when needed without the need for external DACs. Each of the 168 GPIO pins are 5V-tolerant as well. This MCU is also available in the LQFP176 package, making it far easier to hand solder than BGA.

There are relatively few cons for this MCU. It is the most expensive of the options, with an average price of around \$18. The libDaisy Hardware library would also require significant peripheral remapping and minor custom firmware changes, as they differ from its reference implementation.

## **STM32H750IB**

Optimized for balanced performance, the STM32H750IB combines the same 480MHz Cortex-M7 and 1MB of RAM as the STM32H743II with 128KB of flash storage. Many of its peripherals and core features are identical sibling MCU; however, one primary

advantage is that the Daisy Seed development platform was designed with this MCU at its core.

Full libDaisy/DaisySP support provides pre-optimized drivers for I<sup>2</sup>S audio streaming, MIDI parsing, and ADC knob scanning, slashing development time by an incredible amount. It features the same analog peripherals as the STM32H743II, all at a lower cost of around \$10.

Minimal internal flash mandates an external flash chip for preset storage, which will add routing complexity to the overall design. The speed of the MCU at 480MHz limits the voice count to 6-8 voices of polyphony; however, this is well above *NepTune*'s stretch goal of 4 voices.

**Table 4.4:** Pros and Cons for Various MCUs.

	MIMXRT1062	STM32H743	STM32H750
Software Support	Moderate	Good	Good
Library Support	Difficult	Moderate	Good
Package Size	Difficult	Good	Good
GPIO Specifications	Moderate	Good	Good
Analog Peripherals	Difficult	Good	Good
Flash Support	Moderate	Good	Moderate
Performance	Good	Moderate	Moderate
Cost	Good	Moderate	Good

### Final Choice of MCU Part Selection

After weighing the options, the STM32H750 was selected as the primary MCU for *NepTune*. The choice between it and its upper-spec sibling was decided based on cost and library support, while the MIMXRT1062 was ultimately disregarded due to the difficulty of the BGA package to solder and its lack of analog peripherals. The inclusion of flash memory on-chip was not a big enough win for the STM32H743, as the price of simply buying external memory was still cheaper. As *NepTune* will utilize the libDaisy Hardware library, the STM32H750 was the perfect choice, as the Daisy Seed development board the library was designed around uses the same MCU, and integrating the software will be far easier than adapting it to any other MCU.

### Utilization of STM32H750 and Reselection of MCU Part (Senior Design 2 Revised)

After further consideration into the progress made integrating the libDaisy and DaisySP libraries into the bare metal STM32H7, time constraints and major difficulties at the given moment made it clear that a path forward with our current strategy was near to impossible.

While the code onboard DaisySeed's onboard STM32 worked with full functionality with our other devices with no issue, the opposite can be said about the bare chip itself. Started off, one thing we failed to research was the history of the H7 series of STM chips being one of the most difficult to handle on STM32cubeIDE. With over 40 pins to manage, general configuration matched with specific configuration of individual pins along with the code from libraries going from failing, to working, to failing again when we change the smallest configuration setting made for a never ending loop that we did not have the time for.

For these reasons, the group decided to keep some of the code on the DaisySeed dev board and have our primary MCU be the widely known and user friendly, ESP32-WROOM chip. ESP32 is easy because the tooling, libraries, and hardware all reduce friction. You plug it in, hit upload, and things work without much setup. We could use Arduino, ESP-IDF, or PlatformIO in plain C/C++. Other benefits include dual-core CPU with plenty of RAM and flash for its class, rich peripherals (UART, SPI, I2C, I2S for audio, PWM, ADCs, two DACs on classic ESP32, capacitive touch, RMT for precise timing, DMA), power features, production-grade features like OTA updates, secure boot, and flash encryption, many board variants at low cost, and a very active ecosystem.

#### 4.2.2 Flash Memory

Flash memory serves a minor role in the development of *NepTune*, as it will allow users to store “patches” they have made themselves in dedicated banks of memory for recalling later in time. When deciding on what flash memory technology to incorporate, the following parameters must be considered:

- Capacity: Patch file sizes generally small, only ~10MB needed
- Interface: SPI/QSPI preferred for speed
- Endurance: Over 100k cycles preferred, patches written/overwritten frequently
- R/W Speed: Preferably ~0.5ms per page, not incredibly important
- Data Retention: Ensures patches aren't lost/corrupted over time

#### NOR Flash

NOR Flash is a form of non-volatile memory that uses parallel cell connections for true random access. Each individual cell in NOR flash contains its own bit-line contact, further enhancing its random access. They are primarily used for firmware chips and memory in embedded systems [35][36].

Read performance on NOR flash is typically highly regarded; when compared to NAND flash, comparable multi-level cell devices are typically around 3 times faster than their NAND counterparts. NOR flash also has a great lifespan, with 100% known good bits for the life of the part and a high durability for program/erase cycles [34].

The primary drawback of NOR flash can be found in its write speeds and density. NOR flash, although having considerably low latency and high read speeds, has low write speeds. Comparatively, NOR write speeds can be nearly 7x slower than NAND, making it unsuitable for applications where constant writing may occur. NOR flash typically only comes in densities of around 64Mb to 2Gb, which is far smaller than what can be achieved with modern NAND flash alternatives today [34].

### NAND Flash

NAND flash, as opposed to NOR's parallel architecture, features serial cell chains that are optimized for high-density storage applications. This comes with a sacrifice in random access; however, the incredible density the tradeoff comes with is typically worth it for its use case. NAND flash can be found in devices such as USB drives, SD cards, and solid-state drives, with modern devices reaching density levels in the terabyte range [34][36].

NAND Flash offers higher storage capacity than any other technology, with the average cost per gigabyte being around the \$0.10/GB range and offering solutions with multiple terabytes of storage being widely available. NAND flash also offers far better write performance than NOR Flash, especially when handling sequential data read/write operations, making it the one of the best solutions for mass storage purposes [35].

NAND Flash's downsides appear when comparing the endurance and lifespan to NOR flash. Bad block handling is mandatory for NAND, as constant read/write processes can wear out cells at a far greater rate than NOR flash. NAND read speeds also lag behind NOR flash due to slower random-access times [34].

*Table 4.5: Pros and Cons for Memory Technology*

	NOR	NAND
Capacity	32KB - 256MB	64MB - 2TB
Interface	SPI/QSPI compatible	SPI/QSPI compatible
Endurance	Excellent	Moderate
Read Speed	Excellent	Moderate
Write Speed	Moderate	Good
Data Retention	~20 years	~10 years

Cost per bit	\$6.57E-9 USD/bit	\$2.533E-9 USD/bit
--------------	-------------------	--------------------

## Final Choice of Flash Memory Technology

The final choice for which flash memory technology to implement is simplified by the specific niche that the respective technologies fill. NOR flash will ultimately be the final choice for *NepTune*'s on board memory, as it boasts excellent read speeds for quick patch selection and longer endurance; however, the benefits of expandable storage via a NAND flash chip such as a microSD card have not been entirely ruled out. Therefore, the ability to expand the storage for extra patches or perhaps audio recording with a microSD card can be implemented with the inclusion of a microSD card slot.

### Flash Memory Parts Selection

Now that NOR Flash has been decided upon as the memory technology of choice for *NepTune*, parts selection can begin. Many of these parts will be very similar in specifications to each other; the most important criteria to look for when making a final decision would be the following:

- Capacity: At least 8MB for suitable patch storage. All chips mentioned below are 64Mb.
- Interface: SPI/QSPI preferred
- Max Clock: Affects speed at which patches are loaded
- Power Efficiency: Low active current preferred, also helps with temperature.

### IS25LP064A-JBLE

The IS25LP064 is a NOR flash chip developed by ISSI. It is the same memory chip that can be found on the Daisy Seed used to develop the project; therefore, it is a prime choice for *NepTune*'s memory chip. It boasts the best power efficiency of the chips compared, with an active read current of 13mA and a maximum operating temperature of 105°C. The chip also has a wider voltage tolerance of 2.3-3.6V, which provides stability in situations where voltage may fluctuate, or allows for lower power consumption overall. Its QSPI interface enables high-speed 133MHz data transfer for quick patch loading, with support for Double Transfer Rate (DTR) technology that can boost clock rates up to an equivalent of 553MHz. Its main drawback would be a higher cost than the alternatives.

### W25Q64JVSSIQ

The W25Q64JVSSIQ by Winbond provides a cheaper alternative to the chip mentioned above. It delivers the same 133Mhz clock speed, with similar support for DTR, at a cheaper price. Where it lacks can be observed when looking at its power consumption, where it has nearly double the active read current of the IS25LP at 25mA, and a smaller voltage range of 2.7-3.6V. Its maximum operating temperature is slightly lower at 85°C, though this most likely will not affect the synthesizer in any way.

**Table 4.6: Pros and Cons for Memory Options**

	IS25LP064A-JBLE	W25Q64JVSSIQ
Capacity	64Mb	64Mb
Speed	133Mhz (532Mhz w/ DTR)	133Mhz (532Mhz w/ DTR)
Interface	SPI/QSPI compatible	SPI/QSPI compatible
Active Read Current	13mA	25mA
Voltage Range	2.3V - 3.6V	2.7V - 3.6V
Cost	\$1.38/IC	\$0.97/IC

### Final Choice of Flash Memory Parts Selection

The IS25LP064A-JBLE, with its lower active read current, wider voltage range, and preconfigured libraries from the Daisy Hardware Library, is the obvious choice for the flash memory of *NepTune*. Although the W25Q64JVSSIQ is cheaper, its higher power consumption makes the price a not-so compelling tradeoff.

#### 4.2.3 Digital to Analog Conversion

##### Digital to Analog Conversion Technology Research

There are multiple instances of digital to analog conversion throughout the design of *NepTune*; however, where this process is most crucial is in the stage where the digital voices are generated by the MCU and converted to analog for processing them through the various signal processing systems. In order for the audio signal to produce a high-quality sound and for the design to be efficient, it is imperative that the Digital-to-Analog Converter (DAC) or CODEC chosen has the following characteristics:

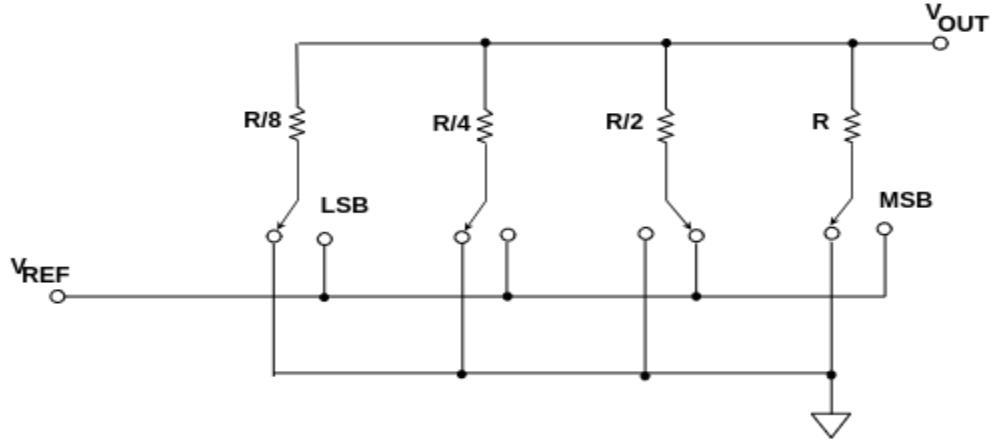
- Total Harmonic Distortion + Noise: Determines waveform purity
- Dynamic Range: Ensures subtle decays and loud transient responses are captured
- Signal-to-noise Ratio: Ensures noise is subtle/inaudible at maximum gain
- Sampling Rate: Affects the accuracy of the signal being generated, must support typical audio range of 44.1 kHz to 48 kHz.
- Bit Depth: Ensures clean sound by reducing quantization in signal conversion

#### Binary Weighted DAC

Binary Weighted DACs are one of the oldest forms of digital-to-analog conversion developed, with one of the first one being developed as early as 1921 [38]. They work by utilizing one switch (resistor or capacitor) per bit, where each bit switches a current proportional to its significance, into a summing amplifier. This, in turn, enables high

sampling rates that are limited only by switch settling times; however, the linearity depends critically on the matching of the components, both in value and in temperature [37].

The simplicity of this architecture allows for relatively high-speed operations up to several MHz, making it suitable for applications requiring rapid signal generation. They are simple to design and can offer decent sampling rates, with a trade off against resolution



**Adapted from:** B. D. Smith, "Coding by Feedback Methods," Proceedings of the I. R. E., Vol. 41, August 1953, pp. 1053-1058

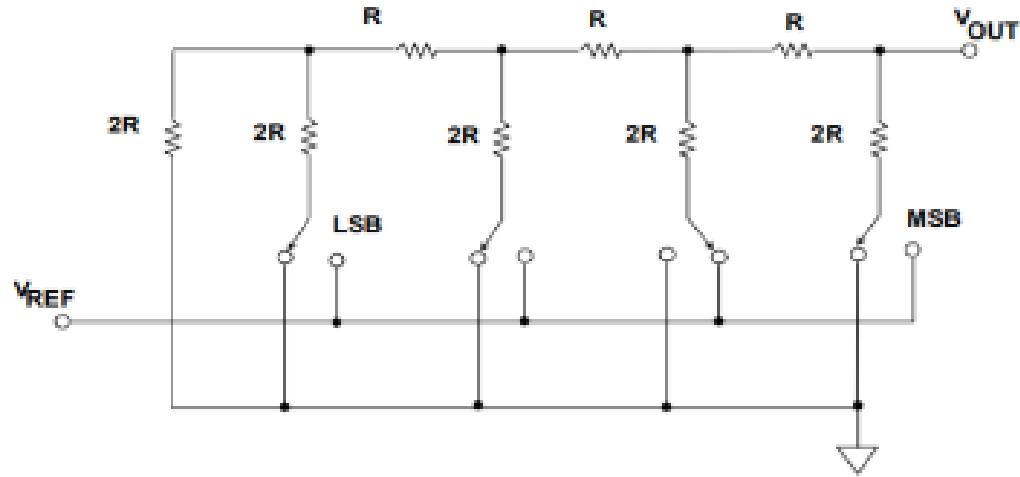
**Figure 4.4:** Binary Weighted DAC layout.

Binary Weighted DACs are not without their faults. As mentioned previously, exponential resistor/capacitor scaling imposes severe practical limits. For example, an 8-bit DAC would require an MSB:LSB resistor setup in a 128:1 ratio, all with the matched temperature coefficients to ensure accuracy. Capacitive versions face issues with  $kT/C$  noise, and leakage causes them to lose accuracy within a few milliseconds of being set [38].

### R-2R Ladder DAC

R-2R Resistor Ladder DACS are a very common form of discrete digital-to-analog conversion. This architecture utilizes a network of identical resistors (of value R and 2R) to achieve binary weighting through nodal division and requires  $2^N$  resistors per N-bit. This makes it inherently non-monotonic, though trimming and matching resistor values should still be considered [38]. The relaxed tolerance for the values allows for a higher resolution, with some DACs such as the Burr-Brown PCM63 reaching up to 20-Bit precision.

Non-monotonicity provides extra comfort when utilizing this technology, as it ensures code is not missed. THD+N typically reaches around -80 to -90 dB with this architecture, and SNR can be seen up to 116 dB minimum. They also can be used in both current and voltage modes, making them a flexible option for different applications. While not empirical data, audiophiles laude R-2R DACs for their high precision and resistance to drifting.



**Figure 4.5:** R-2R ladder DAC layout.

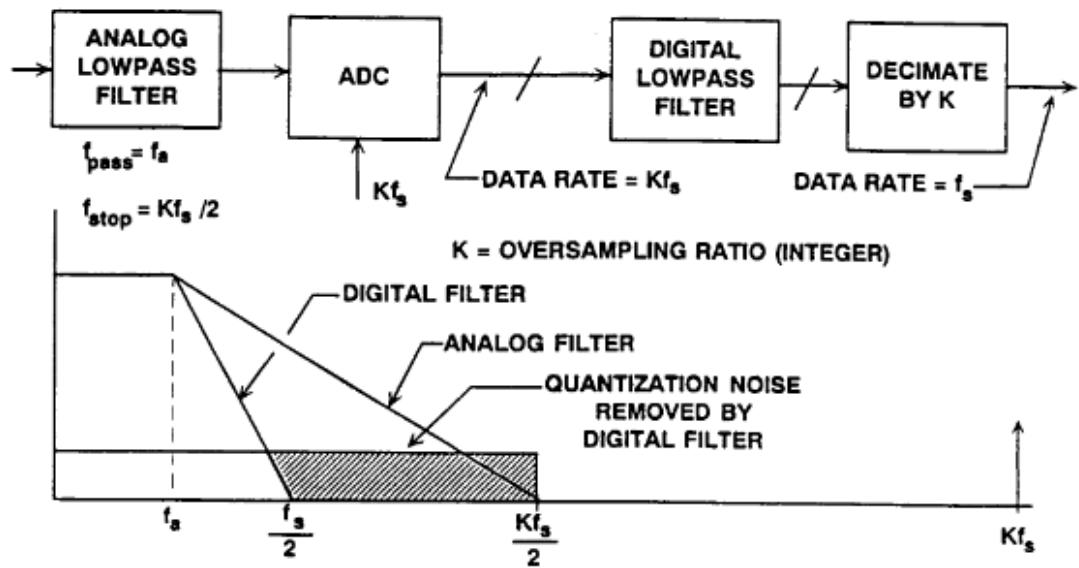
### Sigma-Delta ( $\Sigma\Delta$ ) DAC

Sigma-Delta DACs utilize oversampling and a low-resolution internal DAC to achieve ultra-high effective resolution. Digital interpolation filters upsample the input, while a delta-sigma modulator converts the signal into a high frequency bitstream. Quantization noise is shifted and filtered by an analog post-stage. This architecture has recently began to dominate the premium audio market, with ICs from Analog Devices and Texas Instruments making them mainstream. An application Note from Analog Devices notes that “Until recently, however, the process technology needed to make these devices commercially viable has not been available. Now that 1 micron and smaller CMOS geometries are manufacturable... sigma-delta converters will become even more prolific...”

Noise Shaping enables exceptional THD+N and dynamic range, with SNR exceeding levels of over 120dB thanks to concentrating quantization noise outside the audio band. Bit depths of up to 56-bits can be seen in devices such as the ADAU1401A, allowing for incredible resolution. Some modern implementations have reached sampling rates of 768 kHz, sufficient for ultrasonic content.

Sigma-Delta DACs are very digitally intensive, which can cause delay in real-time applications. The High-frequency noise aliasing requires good analog filters as well, which can cause noise and distortion if the circuit implemented is not properly thought out.

Sigma-Delta DACs are typically larger than their counterparts, which must be considered.



**Figure 4.6:** Sigma-Delta DAC diagram.

**Table 4.7: Pros and Cons for DAC Technology**

	Binary Weighted	R-2R Ladder	Sigma-Delta
THD+N	-55 dB to -60 dB (≤12-bit)	-80 dB to -90 dB (16-bit)	-100 dB to -120 dB (24-bit)
Dynamic Range	80-90 dB	90-100 dB	110-130 dB
SNR	80-90 dB	95-105 dB	100-130 dB
Sampling Rate	41-48 kHz compliant	41-48 kHz compliant	41-48 kHz compliant
Bit-Depth	6-12 bits	8-16 bits	16-32 bits

### Final Choice for Digital-to-Analog Conversion Technology

In order to ensure that *NepTune* has the highest-fidelity audio possible, a Sigma-Delta DAC delivers superior THD+N, dynamic range, and SNR, and is widely used and available in easy to implement ICs from popular manufacturers. While an R-2R DAC would be more than adequate for the application, the headroom that going with a Sigma-Delta DAC provides room for extra performance that may be achieved.

#### Digital-to-Analog Conversion Parts Selection

Now that Sigma-Delta has been chosen as the technology of choice for the Digital-to-Analog conversion, the devices used for the signal processing can now be chosen. The criteria needed are as follows:

- Channels: Needed per voice; stretch goal of 4 voices dictates 4 channels
- Software Viability: Must be compatible or adaptable to libDaisy library
- THD+N: Below -70dB
- Dynamic Range: Above 70dB
- Power Supply: No more than 5V, 3.3V digital preferred

### PCM3060 (Texas Instruments)

The PCM3060 is a 24-bit, stereo audio Codec supporting 96kHz ADC and 192kHz sampling rates. It is integrated into the same Daisy Seed Rev7 development board that is used for prototyping *NepTune*, therefore it will be very familiar throughout the development process. It offers two ADC channels and two DAC channels with optional differential outputs; therefore, two of them would be necessary for 4-voice capabilities.

The PCM3060 can be controlled via I2C, SPI, or left to its own as a hardware module. It receives signal via its SAI module and has drivers readily available for the Daisy Seed and STM32H750 that is utilized in *NepTune*. A single 3.3V power supply simplifies the power requirements for the Codec, and they are readily available from Texas Instruments. The 99 dB SNR and –93 dB THD+N are well over the requirements for subtractive synthesis.

The largest downside to the PCM3060 is the need for two of them, given the limitation of its stereo channels. The PCM3060 contains only one differential stereo output, with the ability to configure it for single-ended output. Sacrificing ease of programming for cost and PCB real estate is something to be considered.

### **PCM3168A (Texas Instruments)**

The PCM3168A is a 24-bit, 6-In/8-Out Audio Codec that supports up to 192 kHz sampling, a SNR of 112 dB, and a THD+N of –94dB, significantly outperforming the dual PCM3060's capabilities in a single chip. However, it requires both analog and digital supply voltages to operate

The PCM3168A's astounding 8 DAC channels eliminate any need for multi-chip solutions, reducing board space and interconnect complexity. As stated, the specifications also outclass the PCM3060; however, there is something to note about how much quality the better specifications could even provide to the sound *NepTune* emits.

The extra power needed by the PCM3168 complicates the power supply design onto the digital board, and the HTQFP-64 package, while more compact than the PCM3060, could be difficult to solder onto the PCB, especially with the thermal pad underneath. Full drivers would need to be configured since the footprint and pinout differ greatly from what is available in the libDaisy Hardware Library.

### **TAC5112 (Texas Instruments)**

The TAC5112 is one of the latest Codec offerings from Texas Instruments. It boasts a sample rate of up to 768 kHz, an SNR of 108 dB, and a THD+N of –97 dB. This Codec offers two stereo channels with differential outputs; however, unlike the PCM3060, setting it to single-ended grants the user four individual single-ended outputs. It takes 3.3V as its power input and offers low power modes, allowing it to run off 1.8V.

The TAC5112's sub-\$5 cost makes it an appealing choice for the budget of the project. It offers essentially every benefit of the PCM3060, but at greater specifications and lower potential power draw.

The main drawback of the TAC5112 is its awkward package, VQFN. While not impossible to solder by hand, it would provide a challenge greater than that of the PCM3060's exposed pins. As mentioned with the PCM3168, the greater specifications may be beneficial but also underutilized for the scope of the project. Furthermore, The TAC5112 is currently

facing supply issues, as it is unavailable in nearly every retail store available at the time of writing.

### **ADAU1401A (Analog Devices)**

The ADAU1401A combines a 28/56-bit DSP core with a 4-channel codec, enabling onboard effects processing without taxing *NepTune*'s MCU. Each of its DACS have a SNR of 104 dB and THD+N of -90dB. It can function as a complete standalone operation, is controllable via GPIO and supports a sampling rate of up to 192 kHz.

SigmaStudio's GUI allows drag-and-drop filter design and voice mixing, freeing MCU resources for voice allocation. Integrated DSP supports advanced algorithms like convolutional reverb, and the four DAC channels simplify routing. Everything is controllable via a 4-wire SPI port or an I2C bus.

Because of the complexity of the ADAU1401A, clock synchronization with the STM32H7 would cause potential nightmares, with firmware hacks and meticulous timing verification necessary to ensure everything runs smoothly. SigmaStudio's learning curve would increase development time, as it would be yet another program to learn. No community examples exist for integration with Daisy Seed and STM32H7. It is also the most expensive option, at around \$10 per chip.

**Table 4.8: Pros and Cons for Various DACs**

	PCM3060	PCM3168	TAC5112	ADAU1401A
Channels	2 ADC, 2 DAC	6 ADC, 8 DAC	2 ADC, 4 DAC	2 ADC, 4 DAC
Software Viability	Full Support	Moderate	Moderate	Difficult
THD+N	ADC: -93 dB DAC: -94 dB	ADC/DAC: -94 dB	ADC/DAC: -97 dB	ADC: -83 dB DAC: -90 dB
Dynamic Range	99 dB	104 dB	114 dB	100 dB
SNR	99 dB	104 dB	108 dB	104 dB

### **Final Choice for Digital-to-Analog Parts Selection**

The final choice for *NepTune*'s DAC came down to what would be the simplest to implement. The PCM3060, with its preconfigured drivers for the development board and it and the PCM3168 being the only devices in the list with a hardware mode that did not require and I2C or SPI communication, made it the primary choice for the synthesizer. The TAC5112 would have been an excellent choice if it was not facing supply issues, while the

PCM3168 would have simplified circuitry by needing only one of them, but its packaging would prove complex.

#### 4.2.4 MIDI Keyboard Interface

There are many options available when deciding on what MIDI interface will be the primary method for the user to play notes on *NepTune*. There is no correct way to decide on what interface to utilize; musicians have different preferences for different use cases. Nevertheless, *NepTune's design* philosophy is focused on four core tenets: portability, simplicity, affordability, and repairability. With these tenets in mind, the following criteria are the most important when deciding on what user interface to use:

- Lifespan: How many actuations each switch is rated for
- Height Profile: Fits backpack constraint (Spec: 18"x12"x4")
- Cost/Key: Affordable,  $\leq \$1.50$  preferred
- Ease of Integration: Can be integrated into PCB without issue
- Tactile Feedback: Piano-like weighted response
- Velocity Sensing: Dynamic range for expressive performances

#### User Interface Technology Research

There are three primary methods of interfacing *NepTune* that are commonly used in the synthesizer industry.

##### Piano-Style Keys

Piano-Style Keys, or more appropriately, Lever Action Soft Dome Switches, are the type of keys that most people are familiar with when looking at any form of MIDI keyboard or synthesizer. They are synonymous with the look of a traditional acoustic piano key bed and are functionally identical to them. They involve the use of a plastic lever arm (often pivot-mounted) to transfer finger pressure onto a silicone or rubber dome switch positioned above a PCB. When pressed, the lever focuses force onto the center of the dome, causing it to collapse and bridge conductive contacts within the PCB. Releasing the key allows the dome's elasticity to revert to the original shape, breaking the circuit and resetting the lever.

The greatest advantages to using this style of interface are ergonomics and expressiveness. Musicians are familiar with the look and feel of piano keys, making them the perfect choice for the performing artist on the go. Silicone domes are typically rated at 1-2 million actuations per key, lending them excellent durability. These switches can also be configured to support velocity sensitivity, enabling the use of MIDI Aftertouch. This feature allows expressive control over how notes are played by allowing velocity and pressure to be modulation sources for different components of the synthesizer, such as filter cutoff, volume, LFO rate, etc.

Despite the incredible advantages that going with piano-style keys would grant *NepTune*, there are glaring difficulties that make them a formidable option to go with. The first hurdle to pass is the mechanical design necessary to fabricate the piano keys, as they are not easily available to purchase. Purchasing keys would involve buying repair kits for existing

products on the market and adapting them to the design; these kits can be incredibly expensive, with most kits hovering around the \$4-\$8/key price range. Purchasing keys would involve buying repair kits for existing products on the market and adapting them to the design; these kits can be incredibly expensive, with some kits going for upwards of \$200, and have no standardized design for how the levers are implemented, nor standard for quality of the silicone domes that are readily accessible.

## Capacitive Touch Switches



Figure 4.7 Capacitive touch keyboard.

Capacitive touch switches are a type of electrical switch that detects physical contact by measuring the change in capacitance of a conductor within a membrane. This fluctuation of capacitance allows the switches to measure both when and where the user is pressing, allowing for a unique and dynamic experience. The switches are typically composed of three layers: a graphical overlay, the circuit, and a backer underneath. When the user presses down on the membrane, the electrical charge is disturbed, and the capacitance can be measured.

Capacitive switches have a few advantages that make them an appealing choice. First, they are the lowest profile option of the selections possible, with the membranes being incredibly thin, and the PCBs only being a couple millimeters at most. These switches also boast a near infinite lifespan, as they necessitate no moving parts and are only prone to damage if the contacts were exposed or corroded. The electrodes needed to build them are also very cheap, with each key only costing about \$0.30 each.

Capacitive switches are, however, the least desirable when it comes to comfort when playing. Because these switches offer little to no tactile feedback, they are often relegated to the smallest and cheapest synthesizers in the market, as musicians would vastly prefer anything with a tactile response more on par with traditional keys. There is also no way of implementing velocity sensing, preventing the compatibility of MIDI Aftertouch. There is also the possibility of the switches being susceptible to EMI/RF interference; however, this most likely is not a risk for the application.

## Mechanical Key Switches

Mechanical key switches can be found in computer keyboards all over the market and have recently seen a surge in popularity in enthusiast and hobbyist communities for their diversity, customizability, and availability. Each key operates through a physical actuation mechanism where pressing a key compresses an internal spring, forcing metal contacts to close an electrical circuit. Each key functions as an independent, self-contained unit comprising a stem that transfers finger pressure into vertical travel, a spring providing tactile resistance, and gold-plated metal leaf contacts that complete the circuit upon actuation. This design creates distinct physical feedback through either a tactile "bump" or linear resistance curve, giving musicians precise control over key movement—critical for expressive playing. Unlike membrane or capacitive alternatives, mechanical switches physically separate each key's operation, preventing unintended inputs during rapid playing.



*Figure 4.8: Mechanical switch diagram*

These switches offer significant advantages for synthesizers. Their tactile feedback can mirror traditional piano keys, providing musicians with intuitive physical response essential for dynamic performance. Mechanical architectures support advanced features like velocity sensitivity through Hall effect sensors that measure press speed, enabling MIDI expression. Durability far exceeds alternatives, with industrial-grade switches rated for 50–100 million actuations, surpassing silicone dome switches by 100x. Modular hot-swap sockets allow effortless component replacement without soldering, aligning with the project's maintainability goals.

However, mechanical switches present notable challenges. Their height profile, even in low-profile variants like Kailh Choc, requires 15–20mm of vertical space when accounting for keycaps, putting strain on the synthesizer's backpack-portability target. Component costs are substantial at \$1.10–\$1.80 per switch, potentially creeping onto the budget when scaling to 24 keys. They require per-key PCB mounting, and the addition of Hall effect sensors for velocity detection increases the cost and complexity significantly. Anti-ghosting diodes will also be required for polyphonic capabilities.

*Table 4.9: Pros and Cons for Various Keyboard Technologies*

	Piano-Style Keys	Capacitive Switches	Mechanical Key Switches
Durability	1-2M Actuations	Infinite	50-100M Actuations
Height Profile	Tall (30-40mm)	Small (5-10mm)	Moderate (15-20mm)
Cost/Key	\$4-\$8 per switch	\$0.30 per switch	\$1-\$2 per switch

<b>Ease of Integration</b>	Difficult	Attainable	Attainable
<b>Tactile Feedback</b>	Great	Poor	Moderate
<b>Velocity Sensing</b>	Great	None	Possible with Difficulty

### Final Choice of MIDI Keyboard Interface Technology

When weighing all choices together, the final decision ultimately was decided upon which interface provided the best tactile feedback relative to the cost. All three technologies were possible to implement; however, the group unanimously agreed that mechanical Key Switches were the most appropriate choice for *NepTune*. Capacitive Switches' many advantages were vastly outweighed by the very poor tactile feedback they provided, to the point that when demoing a synthesizer that utilized them, even the less experienced synthesizer users of the group agreed that they were not comfortable to play. Piano-Style Keys were the most ideal, as they provided excellent tactile feedback and excellent durability; however, the cost and the effort needed to implement them are beyond the budget and scope of the project. Mechanical Key Switches offered an appealing choice in their tactile feedback being both varied and customizable, with the choice of dozens of different key switches on the market, reliable availability, ease of repairability, and relative affordability. With this decision made, the type of mechanical key switch can now be determined.

#### MIDI Keyboard Part Selection

When deciding on what type of Mechanical Key Switch to implement for *NepTune*, there are an incredible number of options available due to the large variety of switches available. The decision for what switch profile to choose comes down to five factors:

- Height Profile: How tall the switch is from the PCB
- Cost: Average cost per switch
- Repairability: Whether they are compatible with hotswap sockets or not
- Availability: How many options are on the market.
- Ease of Integration: Whether additional programming is required or not

#### MX-Style

MX-Style switches cover a broad ecosystem of switches that are designed to adhere to the cross-shaped stem standardized by the German switch and keyboard company, Cherry. This standard ensures compatibility with a vast range of keycaps and hot-swap sockets. They typically extend around 12mm from the PCB, have an average travel distance of 3.5mm-4mm, and can be purchased for as little as \$0.10 per switch, but are more typically found in the \$0.30-\$0.70 range. They are compatible with the Kailh hot-swap platform, allowing them to be replaced if they wear out, or switched out for a different tactile

response based on the user's preference. These switches can be configured very simply using a switch matrix connected to the GPIO pins of the MCU.

### Kailh CHOC

There is little difference between Kailh CHOC switches and MX-style switches. Kailh CHOC switches can utilize the same Cherry standard stem as MX-style switches; however, they are offered in a far smaller profile than their competitors. Kailh CHOC switches extend only about 5mm from the PCB, and have an average travel distance of 3mm, with an actuation distance of half that at 1.5mm. They are a bit more expensive at around \$0.50-\$1.00 per switch and are just as simple to implement.

### Hall Effect

While technically a technology that can be applied to both profiles mentioned above, it is worth mentioning that there are readily available Hall Effect switches available in the Cherry MX profile. All the specifications of the Cherry MX style switches apply; however, rather than depending on physical contact to register a keypress, they contain a magnet in the stem. Utilizing Hall Effect switches would require implementing additional Hall Effect sensors underneath each switch, adding complexity and cost. Dealing with the complexity of Hall Effect sensors would allow us to measure the velocity and pressure of the keys by measuring the magnetic flux of the keys and sending the data to the MCU.

*Table 4.10: Pros and Cons for Various Keyboard Switches*

	MX-Style	Kailh CHOC	Hall Effect
<b>Height Profile</b>	Tall (~12mm)	Small (~5mm)	Variable (5-12mm)
<b>Cost/Switch</b>	\$0.10-\$0.70 per switch	\$0.50-\$1.00 per switch	\$1-\$2 per switch
<b>Durability</b>	Great	Great	Great
<b>Ease of Integration</b>	Attainable	Attainable	Difficult
<b>Velocity Sensing</b>	Not Possible	Not Possible	Possible

### Final Choice of MIDI Keyboard Part Selection

While all the options mentioned above are feasible, the commonly available MX-Style switch will be utilized for *NepTune*. This choice came down to the availability, cost, and variety that going with this style of switch provides. While Kailh CHOC switches would make the height profile of the synthesizer slightly smaller, there are a lot less available options to choose from for different tactile feedback or replacement parts; furthermore, the

smaller size can be a detriment to tactile feedback, as the taller MX-style switches provide a more similar experience to piano-style keys. Hall Effect, while a nice option to potentially implement, would be a costly and difficult task to integrate into the PCB design.

#### 4.2.5 User Interface

Designing the user interface for *NepTune* turned out to be one of the most unforeseen difficulties throughout the development of the project. In many synthesizers, potentiometers are used to adjust the control voltages of the various ICs. When these potentiometers are connected directly to the ICs there is typically no path back to the MCU for these voltage levels to be read and stored for later use, preventing the ability to recall sounds made and saved as patches. To circumvent this issue, the group came up with a design that involved *NepTune*'s user interface potentiometers being routed to the ADC of the MCU via multiplexing. This allowed us to save the control voltage values in memory if needed by saving the last value of an adjusted potentiometer. These digitized control voltages are then sent to digital potentiometers, where they are converted back to analog signals to be read by the ICs. Thus, the research and analysis of various multiplexers and digital potentiometers was necessary to progress.

The following criteria were considered when researching the parts for the multiplexers:

- High Channel Count: Higher channel count uses less PCB real estate and pins
- High Switching Speed: Ensures that values change in as close to real-time as possible
- Low On-Resistance: Ensures minimal voltage drop across channels

#### Multiplexer Parts Selection

##### **ADG706BRUZ**

The ADG706BRUZ is a 16-channel multiplexer from Analog Devices. It is the most premium of the multiplexer options analyzed. It features an exceptionally low  $4.5\Omega$  on-resistance, which minimizes the voltage drop and ensures that the data read from the potentiometers is as close to the original as possible without the need for compensation or buffer op-amps. Its 32ns on-time and 16ns off-time means that the switching time is incredibly fast, which is of great benefit in live performance situations where the user may be changing many knobs at once. Channels are isolated to about 60-80dB to prevent cross talk between modulation sources. It supports a +1.8V to +5.5V single-supply, or a  $\pm 2.5V$  dual-supply. Its cost is the greatest downside, with it being the most expensive of the alternatives at around \$10.

##### **CD74HC4067MG4**

The CD74HC4067SM96 is a 16-channel multiplexer that prioritizes cost-effectiveness over performance, with a higher maximum  $180\Omega$  on-resistance and slower 60ns on-time. It supports +2V to +6V single-supply power with no option for dual-supply. It is offered in an easy to solder SOIC-24 package and costs an average of only \$1.

## 74HCT4067PW

The last option analyzed was the 16-channel 74HCT4067PW from Nexperia. It provides a solution somewhere in the middle, with a  $130\Omega$  max On-resistance and a maximum on-time of 42ns. It provides the widest voltage range of all options with the ability to be supplied with +2V to +10V on a single-supply. It is offered in both TSSOP-24 and DHVQFN-24 packages and can typically be found for around \$2.

*Table 4.11: Pros and Cons for Various Multiplexers*

	ADG706BRUZ	CD74HC4067MG4	74HCT4067PW
<b>Channel Count</b>	16-Channel	16-Channel	16-Channel
<b>Switching Speed (Maximum)</b>	32ns	60ns	42ns
<b>On-resistance (Maximum)</b>	$4.5\Omega$	$180\Omega$	$130\Omega$
<b>Supply Voltage</b>	1.8V-5.5V, $\pm 2.5V$	2V-6V	2V-10V
<b>Cost</b>	\$10	\$2	\$1

## Final Choice of Multiplexer Parts Selection

For the highest fidelity possible, the ADG706 was chosen to be the multiplexers used in the user interface for *NepTune*. This choice was ultimately made to ensure that accuracy is held at the highest standard when the user is adjusting their values, as any delays or glitches in the experience of changing values will be very noticeable, leading to frustration and overall dissatisfaction.

## Final Choice of Digital Potentiometer Parts Selection

When deciding what digital potentiometers to use for *NepTune*, the two most important criteria to consider were resolution and the number of potentiometers in the IC. High resolution was imperative; anything under 256 positions could potentially be distinguishable audibly, making the adjustment of control voltages for various parameters like filter cut-off and resonance behave in odd steps instead of feeling smooth. Given that *NepTune* will have 32 different control voltages to handle, the more “channels” that could be used on each IC, the easier trace and power management would be for the ICs.

In the search, there was only a single IC that met the criteria noted above:

## AD5206BRUZ10

The AD5206BRUZ10 is a 6-channel, 256-position digital potentiometer from Analog Devices, designed to replace mechanical potentiometers in precision electronic systems. It features a configurable terminal resistance of 10 kΩ, 50 kΩ, and 100 kΩ with linear taper and operates via a 3-wire SPI interface, enabling independent programming of each channel. It supports both +2.7V to +5.5V single-supply operation or ±2.7 V dual-supply. It also supports a power-on midscale preset that can initialize the wipers to the center position upon startup. It has a typical wiper resistance of about 50 Ω and is offered in the TSSOP-24 package.

#### 4.2.6 Audio Signal Filtering

##### Audio Signal Filtering Technology Research

To determine what sound the team would like to be able to generate in *NepTune* a close examination of the different filters that are typically included in a synthesizer is necessary. There are a range of filters configurations to be examined, including, but not limited to, high-pass, low-pass, bandpass, notch, Sallen and key, resonant, all-pass, state variable and multimode. The criteria used will be as follows:

- Effects: How beneficial is the benefits produced
- Configuration complexity: Is it difficult or simple to implement
- Parts needed to generate: How many op-amps does the filter use

#### Notch Filter

One of the most basic filters in a synthesizer is a notch filter. As previously discussed, a notch filter is a type of band-stop filter. Specifically, a notch filter while also attenuating specific frequencies the bandwidth range is very different. Typically, a notch filter only allows for very small bands of frequencies rather than a bandwidth of frequencies, and it is used to attenuate sharp resonant peaks by creating almost a sharp V-shape in the frequency spectrum. Moreover, in order to do that a notch filter requires a high Q and precise frequency targeting. This filter is useful in creating different types of sounds such as a “laser” like sound or replicating a telephone sound.

#### Sallen-Key Filter

A Sallen-Key filter design is a second-order active filter which can be used to implement low-pass, high-pass, and band-pass filter circuits. This filter is designed to contain a high input impedance, and low output impedance. It is typically used in synthesizers sound design because it allows for amplification and gain larger than one. This filter is a good candidate for synthesizer sound design because it can be manipulated to form multiple filters using only one operational amplifier (op-amp). Furthermore, the Sallen- Key filter is ideal for a subtractive synthesizer because it gradually attenuates the harmonics curve allowing for a gentler sound change.

Moreover, the adjustable resonance allows for a range of sound ranging from smooth warm filtering to aggressive tones depending, and their low output impedance allows for less signal interference when connected to other synthesizer modules such as VCOs and VCAs. This filter can also be cascaded with other filters to create other designs such as Butterworth, Bessel, and Chebyshev configurations

The figure below displays all possible Sallen Key configurations in an SSI2140, including low-pass, high-pass, and band-pass. [10]

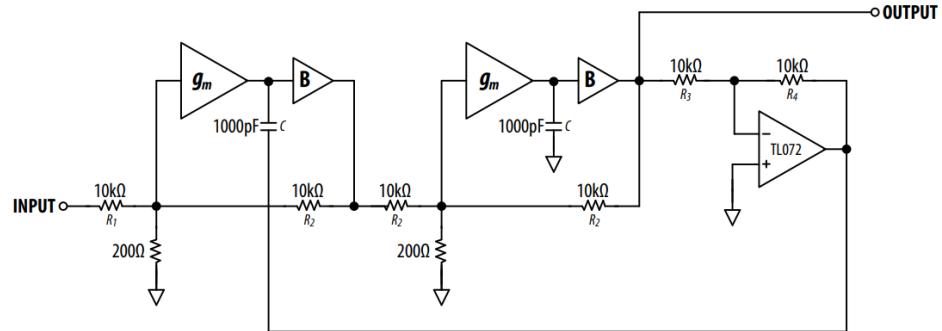


Figure 9: Sallen and Key Low-Pass Filter

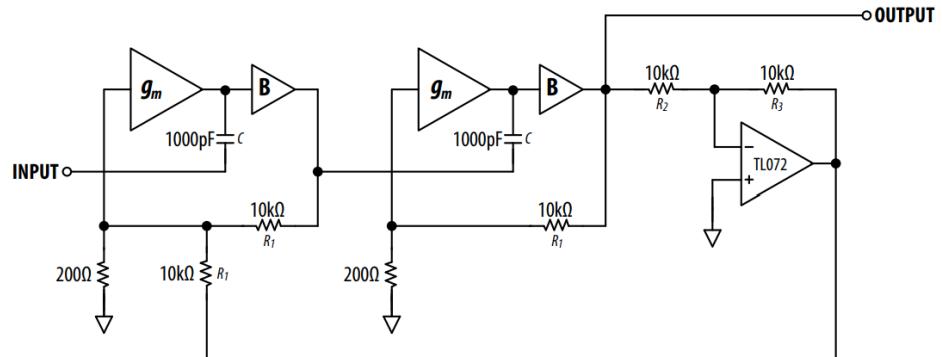


Figure 10: Sallen and Key High-Pass Filter

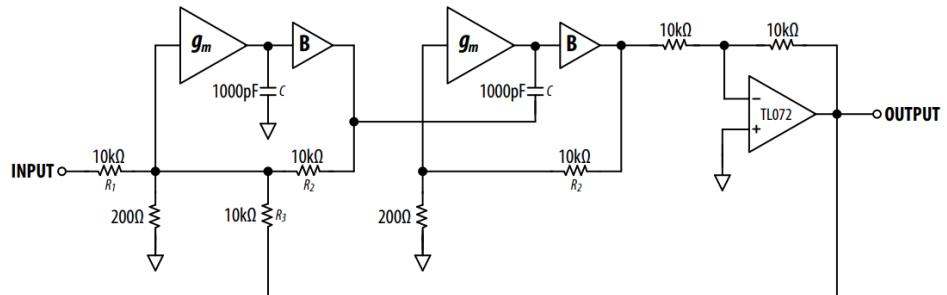
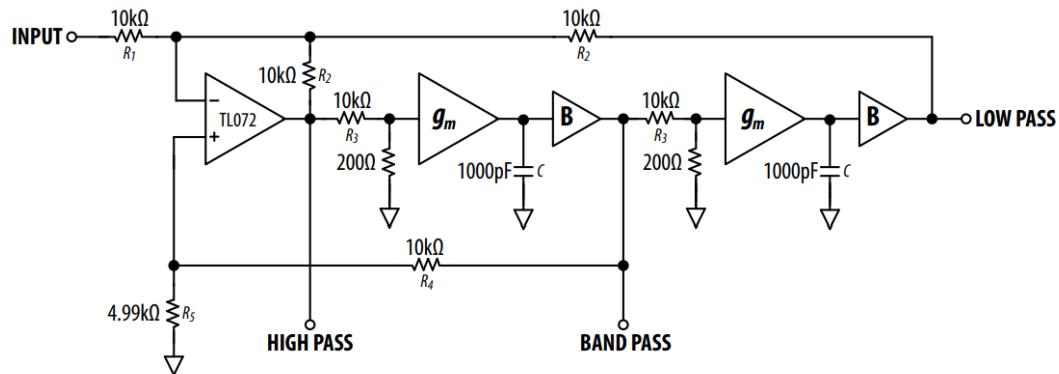


Figure 11: Sallen and Key Band-Pass Filter

**Figure 4.9:** Sallen and Key filter configurations

### State Variable Filter

Another common filter used in synthesizers is a State Variable Filter. This filter is made up of second-order RC active filters consisting of two identical op-amp integrators with each one acting as a first order, single-pole low pass filter, summing amplifier. The main advantage of each filter is that gain, cut-off frequency, and the filters Q can be adjusted independently without affecting the other filters' performance. That means that this filter can produce three filter responses of a low-pass, high-pass, and band pass simultaneously.

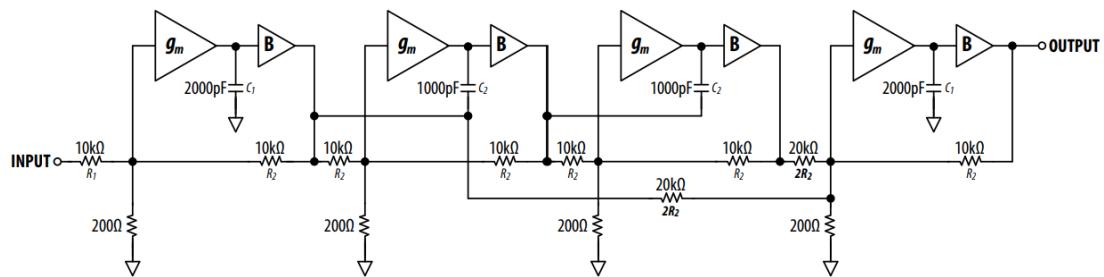


**Figure 12:** State Variable Filter

**Figure 4.10:** State variable filter configuration of SSI2140

### Cauer-Elliptical Filter

The elliptical filter is used to achieve a sharp cut-off but has a reduced stopband performance, and it is typically used when an object has to work over wide frequency range, and the filter needs to cut off sharply above the maximum operating frequency, and give good rejection of the harmonics of the minimum operating frequency, meaning this filter is great for reducing any abnormalities outside of the range when a sudden switch in frequencies occur. Another application for this filter is to reduce the second and third harmonics of a sound that has already undergone filtering, which is an application not necessarily needed but could be useful in a synthesizer.



**Figure 13:** Cauer (Elliptical) Filter

**Figure 4.11:** Cauer-Elliptical Filter Configuration of SSI2140.

## High-Pass and Low Pass filters

The high-pass and low-pass filters are a standard when it comes to filtering with additional variation for refinement. However, many synthesizers still use these basic filters to alter the waveforms as they offer a distinct sound with minimum effort. Moreover, they can be layered for additional sound effect option with simplicity, and are easy to build via any IC. A high-pass filter acts by filtering out any signal above a cutoff frequency, and low pass filters are used to filter any signal below a specific cutoff frequency.

*Table 4.12: Pros and Cons for Various Filter Technologies*

	Sallen-Key	State-Variable	Cauer-Elliptical	Notch	High-Pass or Low-Pass
Advantages	Easy to implement and cascade	Three independent filters in one design	Fastest transition in between frequencies of passband and stopband	Used for a single frequency or very narrow band	Easy to implement hard to adjust Q
Configurations	High-Pass, Low-pass, Bandpass	High-pass, low-pass, bandpass	Band-pass, Stop-band	Band-stop	High-pass, Low pass
Order	2 <sup>nd</sup>	3rd	4 <sup>th</sup>	2 <sup>nd</sup>	1 <sup>st</sup> to 4 <sup>th</sup>
Amount Possible to configure per a 4 pole IC	Single pole per filter	Three poles per filter	4 poles per filter	2 poles per filter	1-4 poles per filter

## Final Choice of Audio Signal Filtering Technology

The filter configuration that was chosen was the Low-pass and State-Variable. These were chosen because they offer the most possible configurations within one design. They also produce the most varieties to the sound signal while using the most efficient number of poles in the filter.

### Audio Signal Filtering Parts Selection

Next, an examination of the integral circuits that could potentially be used for the VCF. The following are the main parameters to examine:

- Is the VCF temperature tolerant

- Is the package easy to solder and place
- Is the VCF easy to manipulate
- Does the filter have at least 4 poles
- Cost

## V3320

The first part that was examined was the V3320 from the company Coolaudio Semiconductors. This IC is specifically designed for analog synthesizers and offers a four-filter with voltage controllable resonance, and a wide range of filters available for configuration including low-pass, high-pass, band-pass, and all pass, which can be achieved by connecting the different sections together. The V3320 also has a broad voltage control range of 12 Octaves with an accuracy of 60mV/decade which is perfect for the use of 1V/octave applications. One of its more unique features, however, is that it has a voltage-controlled resonance, which allows for more fine tuning of the sound through the modulation of Q using control voltages from the low-frequency-oscillators, or envelopes. This is also what provides for its unique vintage sound

This is done through a non-linear response and will require more development and mathematical control to produce certain sounds. This condition, however, contributes to the vintage sound that is often associated with older synthesizers. Nonetheless, despite producing a more unique sound, the amount of control needed makes this VCF a harder to manipulate device. Another noteworthy feature is that V3320 comes in Surface mount technology with 18pins which allows for a somewhat easy installation for the PCB board but makes it difficult to prototype. The V3320 also has a reputation for low noise at around 86dB and total harmonic distortion of approximately 0.1%. The first part that was examined was the V3320 from the company ALFA. This IC offers a four-pole filter with voltage controllable resonance, and a wide range of filters available such as low-pass, high-pass, band-pass, and all pass, which can be achieved by connecting the different sections together.

## SSI2140

SSI2140 by sound semiconductors is highly recommended as VCF in analog synthesizer usage. The SSI2140 is a 4-pole multimode filter as well with capabilities to form Low-pass, High-pass, Band-pass, and All-pass filter. This filter's most impressive feature is its extremely low signal distortion, with its total harmonic distortion of under 0.01% in the passband, it also has an extremely low noise floor of -90 dB and wide output swing of (+/-)10V. A wide output swing refers to the maximum peak to peak voltage range that a filter output can deliver without distortion, which in this case is 20V peak-to-peak with a power supply of 15V.

Another highlightable feature of this filter is built-in temperature compensation for the frequency and resonance control. This is significant in an analog synthesizer because temperature affects how accurate the circuit response is to the voltage. Especially with the presence of transistors as they are highly sensitive to a change in temperature and can therefore alter the consistency of the performance. Moreover, the filters resonance control is linear and current-driven allowing for easier control of the output response. This IC is only available in a SOIC-16 surface-mount package. Finally, the SSI2140's requirement of current base control inputs requires some complication when it comes to the PCB design as converters may be required to ensure that the VCF functions properly.

### **MF10-N**

MF10-N are universal monolithic dual switch capacitor filters. They are made up of two independent active filter blocks that when combined with an external clock signal and up to three or four resistors can produce various second order functions. Each building block contains three output pins. One of the blocks can be configured to perform either an all-pass, high-pass, or a notch filter, while the other two blocks can perform lowpass and bandpass functions. The configuration of the different filters is dependent for the most part on the clock frequency and the external resistor ratios. These are all present separately and can be cascaded to implement higher-order filter circuits.

The clock-to-cut off frequency ratio is tightly controlled, typically within 0.6% ensuring a very controlled and reproducible filter behavior. However, MF10-N can cause issues due to clock feedthrough and aliasing artifacts. Clock feedthrough is an unwanted leakage of the clock signal into the analog signal path, because the MOSFET controlling the capacitors switches can allow for the injection of small pulses into the signal due to capacitive coupling which can cause a degradation of the signal-to-noise ratio that controls the resonance. On the other hand, aliasing artifacts occur when frequencies above half the clock rate are processed by the switch capacitors. Because the switch capacitors operate by constantly sampling the input voltage at the clock frequency, if the input voltage changes too quickly for the sampling rate, the sample signal would be completely different and therefore false or an alias. Therefore, an anti-alias filter would be necessary. While the filter design is simple the tradeoffs of noise and aliasing make it inconvenient for this project needs as another circuit would have to be built to ensure no aliasing takes place. Finally, this comes in a 20-pin surface mount (SOIC) wide body-package.

### **AS3320**

The AS3320 by ALFA is the modernized version of the Prophet-5 analog filter. It also allows for low-pass, high-pass, band-pass, and all-pass configurations using its four-pole filter which can be interconnected. This filter can control voltages over a 12-octave ranges with a reasonable accuracy of 60mV/decade and contains a low noise floor of -86dB and low total harmonic distortion of 0.1%. One of its standout features, however, is that it supports self-oscillation with voltage-controlled resonance. This indicates that when the resonance is high the filter can generate a sine wave tone with no input signal. This can be

used to generate sine wave basses, kicks or whistles. However, this does not seem necessary for the design.

This filter also allows for linear resonance control which allows for predictability, and ease of use when generating a specific filter. However, this poses a problem because its exponential frequency tracking can have an error of up to 12% error in the upper octaves unless carefully calibrated. Frequency tracking is when a VCF changes its cut-off frequency due to a change to the input control voltage, so it is expected that each input increase of 1V should cause a doubling of cutoff frequency, but with a 12% error it makes achieving higher octaves more problematic. Finally, one of the advantages of this filter is that it is available in both PDIP through-hole configurations as well as QFN packages, and while this filter does not have a built-in temperature compensation the internal architecture allows trimming for temperature stability.

To choose the technology that is most appropriate for the design there are a multitude of things to consider:

*Table 4.13: Pros and Cons for Various Filter ICs.*

	V3320	SSI2140	MF10-N	AS3320
<b>Control Method</b>	Linear- Current control	Voltage Control	Clock-signal Control	Voltage Control
<b>Temp Stability</b>	$\pm 200\text{ppm/C}$	$\pm 50\text{ppm/C}$	N/A digital	$\pm 200\text{ppm/C}$
<b>Supply Range</b>	$\pm 4\text{V}$ to $\pm 16\text{V}$	$\pm 5\text{V}$ to $12\text{V}$	$\pm 5\text{V}$	$\pm 9\text{V}$
<b>Package</b>	SOP-18 (SMT)	SOIC-16 (SMT)	DIP-14, SOIC-14	PDIP-18 QFN-24
<b>Poles</b>	State-variable 4-poles	State-variable 4-poles	Switched capacitor 2	State-variable 2-poles
<b>Cost Range</b>	5\$	4\$	4\$	8\$

### Final Choice for Audio Signal Filtering Part Selection

After an extensive comparison, the part chosen for the VCF is the SSI2140 that is because out of the technology compared it offers the easiest control model, via voltage, it comes in an easily solderable surface mount package, as the soldering constraints requires an IC that is easily hand-solderable, moreover, it offers the most amount of poles in an IC whilst offering the most control.

#### 4.2.7 Low Frequency Oscillators

Low Frequency Oscillators are an integral part of synthesizers implemented on almost every synthesizer on the market. Their purpose is not to produce sound, but to modulate the parameters of other signal processing circuits. In *NepTune*, the output of the LFO will be applied to all the filters on the board. If time allows, *NepTune* may also implement an LFO for the amplifiers in the circuit.

Before diving into the various technologies that can be employed to realize synths, it's important to discuss the typical expectations of an LFO. An LFO usually runs at very low frequencies from below 1Hz to around 20Hz. It usually generates various waveforms, primarily triangle, sawtooth, and square. It also has some way to be controlled. In classic synths, this was usually realized via control voltages (CVs). For *NepTune*, the following criteria will aid the technology selection process:

- Frequency range: 0.1Hz to 20hz.
- IC circuitry: ICs will be preferred over designs involving discrete analog components.
- Available ICs on the market: Components must be in active production.
- Multiple waveforms: Sine, triangle, sawtooth, square, and ramp waveforms.
- Digitally controlled: Must be controlled digitally, via clock or PWM.
- Output analog signal.
- Fits the time, cost, and knowledge constraints.

When looking for ways to implement an LFO, these core requirements will be kept in mind.

##### LFO Technology Research:

There are various ways that an LFO can be realized, such as voltage-controlled oscillators (VCOs), digitally controlled oscillators (DCOs), and direct digital synthesis (DDS). There are other ways to create an oscillator at the frequencies an LFO needs to be at, such as bit-banging, but these are not realistic for *NepTune* [8].

##### **Voltage-Controlled Oscillators (VCOs):**

Voltage-controlled oscillators are fully analog devices that oscillate at a frequency chosen via a control voltage input. Meaning that for a given input voltage, the VCO will oscillate at a specific frequency—voltage in, frequency out.

There are many reasons to why a VCO might be a viable option for generating low frequencies in *NepTune*. While classic VCOs in synths were constructed from op-amp and transistor circuits, similar functionality can also be achieved using ICs such as the 555 timer or the XR2206 function generator. These are fully analog solutions, which align well with the team's knowledge base. They also produce an analog output which can feed directly into the VCFs with virtually no latency or digital conversion overhead.

Unfortunately, there are also many drawbacks to using VCOs for *NepTune*. Analog circuit development is time-consuming and will not align with the time and budget constraints.

Audio processing circuitry must be designed with high precision to minimize noise and distortion. This will require higher cost components and increase design complexity. In addition, the team plans to control the LFO via digital control signals from the MCU. A VCO, however, expects analog control signals and will require a DAC. Perhaps the worst part of utilizing a VCO is that purpose-built ICs (such as the XR2206) are obsolete and no longer in production. They have been replaced with components utilizing direct digital synthesis techniques (which will be covered shortly). This also means that the team might not be able to implement all of the desired waveforms.

### **Digitally Controlled Oscillators (DCOs):**

Digitally controlled oscillators are similar in function VCOs but differ in that oscillation frequency is determined by a digital control signal rather than an analog voltage. Typical control methods include digital clocks, PWM signals, or serial communication protocols such as I<sup>2</sup>C and SPI. The result is a system that takes a digital input and translates it into a specific analog output frequency—digital signal in, analog frequency out.

DCOs appear to offer several benefits that make them attractive for use as an LFO. Perhaps the most obvious is their compatibility with digital control systems. A DCO can interface directly with the onboard MCU while feeding stable analog signals into the VCFs. Additionally, DCOs have very little tuning drift due to the stability of the control signals from the MCU. They can also be constructed to fit the desired frequency range for an LFO (0.1Hz to 20Hz).

While DCOs are conceptually ideal for this application, there are practical challenges. Purpose-built ICs for this function are limited, meaning that custom circuitry will need to be constructed. It may be possible to design custom circuitry within the team's time and budget constraints, but not without tradeoffs. Some of the desired waveforms may need to be forfeited. Simple DCO circuits are only able to generate triangle and square waves. Adding more waveforms such as sine, random, or ramp will require extra shaping circuits and increase complexity.

To summarize, implementing a DCO as an LFO for *NepTune* is a viable option. However, it would require more attention as designing custom circuitry is difficult and may come at the cost of several waveforms.

### **Direct Digital Synthesis (DDS):**

Direct Digital Synthesis is a completely digital method to signal generation and can be used to produce analog waveforms. Unlike VCOs and DCOs, DDS is based on microprocessor technology. Waveforms are digitized and stored in memory at very high resolutions. A waveform is generated by stepping through this stored data at various increments. Smaller step sizes will yield a slower output frequency, while larger step sizes will yield faster ones [48].

DDS offers several benefits making it appealing for use in *NepTune*. Most notable is the abundance of purpose-built ICs and microcontroller implementations. This makes it much more accessible than classic analog function generators such as the XR2206 and ICL8038

which have been phased out. There are also no replacements for these classic function generators, further motivating the switch towards digital synthesis.

Beyond parts availability, DDS provides excellent frequency stability with virtually no tuning drift since the output is derived from stable clocks. It also offers exceptional waveform flexibility—not only supporting standard waveforms such as sine, square, random, and triangle functions, but also enables the use of custom or user-defined waveforms, limited only by memory capacity. DDS can also respond quickly to changes in control input [50] [51].

That said, DDS may be overkill for a basic LFO implementation. In a musical context, LFO waveforms do not require the perfect fidelity that DDS provides. In fact, these perfect output waveforms can sound “too clean” or mechanical. Additionally, for *NepTune*, the LFO only needs to output a few waveforms, not the infinite amount that DDS can produce.

### **Final Choice of LFO Technology:**

All three options for realizing a low frequency oscillator—VCO, DCO, or DDS—are viable options. Ultimately, DDS best fits the project criteria. The table below helps illustrate the reasoning leading to this decision.

**Table 4.14: LFO Technology**

	VCO	DCO	DDS
<b>Frequency Range</b>	Yes	Yes	Yes
<b>IC Circuitry</b>	Yes	With Difficulty	Yes
<b>IC Availability</b>	No	With Difficulty	Yes
<b>Multiple Waveforms</b>	Yes	With Difficulty	Yes
<b>Digitally Controlled</b>	No	Yes	Yes
<b>Output Analog Signal</b>	Yes	Yes	With Difficulty

<b>Fits project constraints</b>	No	With Difficulty	Yes
---------------------------------	----	-----------------	-----

The primary reason DCO was chosen to implement the LFO in *NepTune* is component availability. While VCOs were traditionally used for low-frequency modulation, sourcing parts has become very difficult since classic function generators have been phased out of production. Additionally, designing custom circuitry for the LFO is out of the scope of this project. This is also why DCO technology will not be used. There are little to no purpose-built ICs available for DCOs and designing custom circuitry to produce all the desired waveforms will be time intensive.

In contrast, DDS technology benefits from wide availability and is able to realize all the waveforms necessary for the LFO. While DDS is software-intensive and does not align with the team's electrical engineering background (4 EEs and 1 CpE), the availability of open-source libraries and development resources make it a feasible and practical solution. For these reasons, DDS was selected as the most appropriate approach to realizing the LFO in *NepTune*.

### LFO Part Selection

Now that a DDS-based approach has been selected to realize the LFO in *NepTune*, the next step is to identify a compatible part that meets the system's requirements. The following criteria will guide the selection process:

- Software Support: Must have well-documented library support.
- Large Package Size: THT or large pitch SMD components are desired.
- Low Frequency Output: 0.1Hz to 25Hz.
- Desired Waveforms: Sine, triangle, ramp, square, random.
- Analog Output: Outputs analog signal for VCF control.
- Low Cost: Below \$10.

This is the list of criteria that will be used to find and judge components against one another.

### **AD9833**

The AD9833 by Analog Devices is a promising candidate as an LFO in *NepTune*. It is a purpose-built IC utilizing DDS for generating periodic waveforms with high precision and stability.

Several features make the AD9833 attractive for this application. Notably, this microchip requires no programming. Instead, it receives commands via SPI allowing an external MCU to select waveforms and frequencies. Additionally, no external components are necessary. It contains its own DAC and can feed analog signals directly into the VCF. This chip is also low power, consuming only 12.65mW at 3V. While power isn't a major consideration for the current state of this project, it could be beneficial to limit power consumption now in preparation for a future battery-powered iteration.

Unfortunately, there are some drawbacks to the AD9833. Firstly, it only supports three waveforms: sine, triangle, and square, which excludes the other ramp and random waveforms listed in the part selection criteria. Secondly, it is only offered in a 10-lead MSOP package. The pitch between pins is 0.5mm, which makes it difficult to solder. Lastly, the AD9833 is costly. Each IC costs \$15 before tax and shipping, which exceeds budget.

## TAPLFO3D

The TAPLFO by Electric Druid is a DDS LFO implemented on a PIC microprocessor via software. The PIC processor is the PIC16F1824. While it is not a purpose-built IC for DDS, its processing capabilities are more than sufficient to handle the LFO requirements. Unlike the AD9833 which is made specifically for DDS, Electric Druid designs and loads software onto the PIC processor to create a DDS microchip. This means that no further programming is necessary. Additionally, Electric Druid has made their firmware open source and available to the public. This works in *NepTune*'s favor, allowing the team to make changes or expand functionality in the future if needed.

Several practical advantages make the TAPLFO3D appealing. It is available in a 14-pin DIP package, making it easy to handle and solder. It offers 10-bit output resolution, an output frequency range of 0.025Hz-50Hz, and supports 16 waveforms, including sine, triangle, ramp, random, and square. While these are all great, its main standout feature is its tap-tempo capability, which allows users to set a tempo by tapping a control button—a feature especially valuable in performance context. All of these benefits are packaged nicely in a device costing only \$6.

Unfortunately, there are some drawbacks. The output of the TAPLFO3D is PWM, not analog. This poses a problem, since the VCFs in *NepTune* expect an analog control signal. Because there is no built-in DAC, *NepTune* must extract the analog waveforms from the PWM signal via filtering. If the PWM signal is run through a high order low pass filter, an analog signal can be recovered.

Another minor consideration is that the processor will require a PICkit3 or PICkit4 programmer to be reprogrammed. It is highly unlikely reprogramming will be necessary, however, but it is something that needs to be considered as all options are weighed. These devices can be costly, ranging from \$30 for a used kit to \$200 for the newer PICkit4. Fortunately, one of the team members may have access to a compatible programmer, so this limitation may not be a significant issue.

## VCLFO10

Electric Druid offers another DDS-based LFO, the VCLFO10, which is also implemented on a PIC processor. It shares many similarities with the TAPLFO3D, including 10-bit output resolution, 16 waveforms, a PWM output, and the availability of open-source firmware. Like the TAPLFO3D, it is pre-programmed, so no firmware development is required. In fact, it is more productive to list the differences between the two processors.

The key difference, which may be beneficial, is that the VCLFO10 behaves more like a standard LFO. Rather than requiring the input a tap-tempo input to set the frequency, the

user can sweep through frequencies via a knob on the board. This may be beneficial to most users who expect this type of behavior.

In addition, the VCLFO10 also has a sample-and-hold (S&H) function. When enabled it will sample the selected waveform at regular intervals and hold the value, producing a stepped output. For example, if the user has a sine wave selected and enables the S&H function, it will create a stepped version of the sine wave by sampling it at discrete points in time.

This function expands the LFO's creative potential and offers a unique sound that is not available on the TAPLFO3D. This function may be more beneficial than the tap-tempo function.

There are also some minor differences worth mentioning. The VCLFO10 uses a PIC16F1825 processor (compared to the PIC16F1824) and offers a slightly higher output frequency range, from 0.05Hz to 102.4Hz. In this application, a slower frequency is desired but while its minimum frequency isn't as slow as the TAPLFO3D, it still falls within the desired range for LFO operation.

The VCLFO10 shares the same limitations as the TAPLFO3D. Since it is also based on the PIC processor, reprogramming would require a PICkit3 or PICkit4 programmer. Again, it is highly unlikely that the chip will need to be reprogrammed, but it is worth mentioning. Additionally, the output of the VCLFO10 is a digital PWM signal. This means that must be passed through a low-pass filter to extract a usable control voltage for the VCF.

### **Final Choice of LFO Part Selection:**

All three options—the AD9833, TAPLFO3D, and VCLFO10—are promising. Ultimately, the VCLFO10 best fits the project design criteria. The table below is provided to aid in understanding this decision.

*Table 4.15: Pros and Cons for Various LFO ICs.*

	<b>AD9833</b>	<b>TAPLFO3D</b>	<b>VCLFO10</b>
<b>Software Support</b>	N/A	Yes	Yes
<b>Large Package Size</b>	No	Yes	Yes
<b>Low Frequency Output</b>	Yes	Yes	Acceptable
<b>Desired Waveforms</b>	No	Yes	Yes

Analog Output Signal	Yes	No	No
Cost	\$15	\$6	\$6

First off, while the AD9833 is an exceptional DDS IC, its small pitch and limited waveform selection ultimately made it less suitable for use in *NepTune* compared to the TAPLFO3D and VCLFO10. Although it requires no external hardware and is able to function as a standalone DDS waveform generator, the TAPLFO3D and VCLFO10 already come with pre-loaded firmware and can be perceived as a standalone IC. These ICs do require a high-order filter at the output, but the team is capable of creating a sufficient design.

The final decision between the TAPLFO3D and the VCLFO10 came down to their interface and modulation features. The VCLFO10 behaves more like a conventional LFO and is more intuitive than the TAPLFO3D's tap-tempo interface. Furthermore, if tap-tempo functionality is desired, it can be realized externally using the MCU. In addition, the VCLFO10 has the S&H function, which allows more waveform modulation and expands the LFO's creative potential.

For these reasons, VCLFO10 was selected as the most suitable DDS-based LFO implementation for the *NepTune* synthesizer.

#### 4.2.8 Signal Amplification

For such a simple component, the VCA can be a very nuanced section of sound synthesis. There are 3 variations of VCA that were considered for the design of the synthesizer. These 3 types are:

1. Linear
2. Exponential
3. Dual/Quad

##### Signal Amplification Technology Research

###### Linear

VCAs hold the distinct quality of having a 1:1 ratio of CV to output, meaning that control voltage has direct and exact control over the amplification. This quality will improve the user's ability to control the device; however, it can exhibit some unnatural sound qualities due to the abrupt intentional rise and fall of sound from this effect. The upside to this precision means that the ADSR envelope generator will be able to accurately affect the sound according to its configuration.

###### Exponential

VCAs carry a logarithmic curve to the output, mimicking the natural decay of acoustics. This creates a more "natural" sound to the device but sacrifices the exact control described

in Linear VCAs. This form of amplifier is often regarded for its proficiency in audio envelope integration. The largest downside to this configuration is that it cannot handle fast CV frequencies which makes it less adept at handling control voltages; fortunately, this function is not required of the amplifier because NepTune are exclusively using it for modulation of the audio signal.

### Dual/Quad

VCAs are a combination of two or more amplifiers which helps significantly for more substantial designs. For the design, the synth is ranging from 2 ~ 8 voices, and each voice needs its own amplifier. Therefore, the most significant benefit to *NepTunes* performance would be the reduction in footprint from the VCA (lower IC count) and the configurability of Dual/Quad VCA inclusion. Because Dual/Quad can be either linear or exponential, and NepTune is expecting to include at the very least two voices, this VCA is what the team will pursue when researching parts.

*Table 4.16: Pros and Cons for Various VCA Technologies.*

	Linear	Exponential	Dual/Quad
Sound Quality	Low	High	Configurable
Precision	High	Medium	Dependent
Configurability	Medium	Low	High
Complexity	Simple	Simple	Complex

### Final Choice for Signal Amplification Technology

While comparing these different kinds of VCA, it is apparent that the emphasis of sound quality is extremely important and will outweigh the other aspects of each kind of VCA. This is because the overall quality of the music synthesis relies entirely on the sound quality that it produces so anything that degrades from that quality will be heavily penalized due to the heavy toll it would cost to include a part that might impact the sound negatively. This leaves the design team with choosing between simple Exponential and Dual/Quad that is configured to Exponential response meaning the largest decider, is how much the team needs the IC to be able to handle multiple voices and how easy it is to include within the PCB.

### Signal Amplification Parts Selection

Below will be a comparison of the parts the team considered for the design of a VCA.

When considering these different VCA ICs, important factors that will ultimately decide on the chosen component will be the produced

- Noise levels
- Distortion levels
- Ease of inclusion
- Price
- Required power

And other more nuanced aspects are also considered when relevant below:

### **THAT 2180**

While this IC comes in variations (A), (B), and (C), the only significant difference is the Output voltage offset, and the maximum harmonic distortion based on input voltage. This means that the only difference in version is scaling which then lets the team observe the actual performance the same for all versions as the performance remains equilateral between the variations.

THAT2180 uses a landmark Blackmer Cell which was developed in the 1970s which performed massive strides in the pioneering of professional audio VCA technology for the use of music synthesis. It also provides a maximum bandwidth of 10MHz which provides enough range for the audible notes. The datasheet also mentions a remarkably low noise that is ideal for the project.

While the control connection inside the IC is linear, the chip has its control port configured so that it changes exponentially from changes in the control voltage. This is known commonly as a logarithmic-responding VCA.

### **AS3360**

This is a dual VCA that came with matching schematics to the envelope generator that the team decided to use for the ADSR. This made it immediately tempting to run with the AS3360 in the design because of the easy installation and configuration that was already done by the datasheet of the ENVGEN8. The AS3360 runs a true exponential response which meets the audio signal expectations for quality. Another positive quality to this IC in particular is that it can run on power as low as +/- 5V. However, because of the inclusion of ENVGEN8, NepTune will have a minimum 12V bridge already on the PCB that the VCA will be housed negating the complexity avoided by having lower power but leaves the upside of overall less energy consumption from the device even by a small amount.

The price of the AS3360 is a large factor that can be compared by multiplying it twice, as the other options considered are all quad VCAs, only one chip is required for the middle ground goal of 4 voices, for every 2 voices, NepTune will need another IC rather than every 4 so the \$8 package leans slightly more expensive than the other options since in total it comes to \$16 ~ \$32 in all.

### **SSI2164**

Sound Semiconductor is a manufacturer of ICs that the design is relatively connected to as many of the chips are market leading examples of each component and with the VCA the team is integrating, their ICs once again show promising results. SSI2164 is a quad VCA with modular builds of synthesizers in mind which also aligns with the design envisioned by the group. The chip brings class A and AB operation for noise which beats out the other compared VCAs and is better documented on the corresponding datasheet allowing us for better integration in case of misalignment with the components used in the design.

Part of these prefab designs that are present in the datasheet for the SSI2164 is the use of external resistors which is not uncommon for VCAs but will require a level of carefulness and attention to detail with how the connection configuration results. The SSI2164 is also sensitive to power noise, which is another reason for the separate PCB setup, distancing the power portions away from the audio analog signal.

Now the team will compare the three parts directly against one another to draw a final conclusion on which IC will be best suited to be implemented into *NepTune* audio synthesizer.

**Table 4.17: Pros and Cons for Various VCA ICs.**

	<b>THAT2180</b>	<b>AS3360</b>	<b>SSI2164</b>
Noise ( $\mu$ V RMS)	1.3	2.0	0.8
Response	Linear	Exponential	Configurable
Bandwidth	10MHz	$\sim$ 500kHz	1MHz
Distortion	0.003%	0.03%	0.003%
Power Range (V)	12 – 15	5 – 15	12 - 15
Cost (per channel)	\$10	\$4	\$3.75

### Final Choice for Signal Amplification Parts Selection

After comparing the different VCAs considered, the clear choice is to integrate the Sound Semiconductor's SSI2164 especially due to the lower noise rating being critical to music synthesizer creation. The team already has to be concerned about how to operate the power supply and both analog and digital lines, creating many opportunities for noise generation to affect the sound. Avoiding noise at each step is incredibly important. The power issue with the card is not of actual concern as other ICs on that specific PCB will require 12V like the ADSR ENGEN8, which means no further complexity is created. It also is a major upgrade having a quad VCA as the choice because if NepTune ends up including a number of voices that is divisible by 4, the SSI2164 will allow for the smallest amount of IC's included for the project also scaling down the cost of creation.

## 4.2.9 Envelope Generation

ADSR technology is critical for the quality of the music synthesizer modulation capabilities, a cornerstone of quality of respected synthesizers. Ideal ADSR IC chips will have minor external components, and the package will consume little power as multiple ICs will be needed for the total amount of voices included. A wide ADR range is necessary too as it will extend the configurability of the sound to more extremes of the market for quality.

It is notable to also mention the variations of the ADSR formula because not every value has its concrete position when it comes to the use of envelope generators in music synthesis. Below, other recipes for audio signal envelopes will be discussed.

### *Envelope Generation Technology Research*

#### **AR (Attack – Release)**

This far simpler design only includes values for Attack and Release meaning that the other familiar values, Decay and Sustain are not included. This makes the discrete integration of an envelope generator far simpler and ultimately the cost go down as well. These more mundane generators offer less user customizability, so typically are reserved for highly specific devices. The exclusive use of Attack and Release allows for the generation of percussive sounds that imitate drums and plucks.

#### **AD (Attack – Decay)**

Similar to AR this envelope generator only hosts two of the four common values, so the user ability is once again reduced. The more one-off sounds that are capable with this configuration resemble sounds like snares and kicks. Again, these simpler orientations are reserved for more specific uses and not music synthesizers aimed at the large majority of customers.

#### **AHDSR (Attack – Hold – Decay – Sustain – Release)**

Contrary to the previous two versions, this envelope generator comes with an additional value to the common four. This value is Hold; a value that will create a fixed-time plateau of amplitude for the sound signal before decay occurs. This grants increased control from the users that may be interested in this feature, predominantly with recreating brass instruments by mimicking the swells of the sound found in real world playing. The inclusion of another element does increase the difficulty of design by adding additional timing elements, so such an advanced envelope generator is also found commonly only in higher end, more professionally rated synthesizers.

#### **ASR (Attack – Sustain – Release)**

Present in this orientation are three of the four values which means that the sustain is immediately following the attack stage. The sound often used with this technique is organ

and droning. Another specific envelope generator that has its purposes in the greater market for audio equipment but will only degrade the scope of the synthesizer.

### **DAHDSR (Delay – Attack – Hold – Decay – Sustain – Release)**

Delay is a value that will wait a defined amount of time before attacking, meaning the sound does not grow until after the delay has elapsed. Because the DAHDSR is a very complicated and expensive tool for any synth that integrates one, it is commonly only found in high-end professional builds. For this reason and the overall low demand for this envelope generator, it is unlikely to be integrated into *NepTune*.

Sounds like this do assist with complex music synthesis strategies which is commendable, sound layering and pre-echo on vocals and instrumentation are both possible, so a more advanced synthesizer would definitely contemplate adding it.

### **Multi-Envelope**

This envelope generator uses digital processing to create multiple envelopes for extreme precision control. Because all values can be directly programmed to certain amounts, it allows the most complex envelopes to be synthesized. While the insertion of this envelope generator would typically be quite easy from a hardware standpoint, the price and the complexity of coding such a device is incredibly taxing.

The extreme maneuverability of the envelope allows for incredibly desirable unique sounds like bouncing ball delays which are simply not possible to recreate without the multiple envelopes that this strategy provides.

### **Looping**

When using an envelope generator that is hybrid and can support a LFO, it unlocks the ability to loop particular values of a standard ADSR envelope generator. The LFO operates and, depending on the user's input, can loop at different frequencies. This is not mutually exclusive with other ADSR variants and in general can be thought of as an optional add on, one of the ICs the team mentions following this section will have LFO operation. Overall, looping is incredibly useful for music synthesis; certain genres will use it more than others, so it may not captivate the entire market for music synthesizers but will certainly add to the scale of features.

**Table 4.18: Pros and Cons for Various ADSR Technologies**

	ADSR	AR	AD	AHDSR	ASR	DAHDSR	Multi	Loop
Complexity	Medium	Low	Low	High	Low	Very High	Very High	Medium
Range	High	Small	Small	High	Medium	High	Very High	High

Sound	Standard	Percussion	Snare	Brass + Standard	Organ + Droning	Layering + Standard	Precise Unique	Looping
-------	----------	------------	-------	------------------	-----------------	---------------------	----------------	---------

## Final Choice of Envelope Generation Technology

Implementing the standard ADSR will lead to a good base for *NepTune* from which it can be modified if the design process provides enough time and resources. Having the different envelope generation methods laid out gives the team a solid foundation on what the user will be able to create using the final product. The team can also see the complexity of each addition meaning allocation of resources can be better managed rather than wasted chasing a difficult addition.

### Envelope Generation Parts Selection

Now to present a few parts that were researched during the task of finding which IC chip would be used for envelope generation.

#### AS3360

The first ADSR investigated is unconventional because it isn't intended to be an envelope generator primarily. During the planning the team was conscious of the market for analog synthesizers which was at one point, the ideal market the synthesizer was designed for; however as the team researched and properly scoped the design out, it also opened up to the possibility of simplifying the design to ease the manufacturing and reduce the complexity/cost of repair.

The AS3360 was originally researched as a part of the original goal of appealing to the niche with a more analog heavy design that has a dedicated consumer base. Because of that, the inclusion of this Dual VCA, configured to be used as an envelope generator, was to add to the analog complexity of the synthesizer. It offers a good noise level as the VCA is also graded for audio, with good exponential response which is ideal for the envelope as to imitate acoustic sound as close as possible.

In order to make this repurposed VCA into an envelope generator, you have to include discrete components in order to make multiple (1 for each ADSR element) RC circuits. These connections will create a timing that will act as the user configured ADR elements while the Sustain level can be entered via basic voltage divider. All of which are controlled on the user end by potentiometers affixed to the board. On the design side of this implementation though, the process of calculating and configuring the capacitor or resistor values is far more complex.

Part of the research on how to create this configuration and parallelly the research on figuring out a discrete solution to ADSR execution was a lesson to the group on the importance of simplifying the product. Not only for ease of design and reduction in

development cost, but to also supply the deadline expectations which would prove manual discrete calculation and design difficult to maintain.

## AS3310

AS3310 is the contemporary version of the CEM3310, which is a dedicated ADSR IC that has been discontinued. The AS3310 has a single channel and offers complete analog ADSR behavior is all housed within the IC package, meaning that the control is still entirely control voltage based and has its timing ranges dependent on the configuration of RC circuits externally. This would mean that similar to the previously mentioned AS3360, some level of discrete integration would still be required, bringing all of the resulting consequences to the design, albeit on a reduced scale. The IC requires 12V to 15V power, which is achievable without alterations to the current plan, and offers 0 – 10V output both with linear and exponential configurations; this is important when the team connects the ADSR to its corresponding VCA. One final important note about this ADSR chip is that it is the most expensive option the team has found, and with the complexity of socketing it along with external components, the overall likelihood of this IC ultimately being included is low.

## ENVGEN8

Electric Druid has been a store that the team has visited for the purposes of IC frequently which led to the discovery of this ADSR oriented envelope generator. The unique aspect to this IC package is that it runs a digital internal operation meaning that while it is still controlled via a CV connection, it still operates the output as either linear or exponential based on the input level at the unique pin. Digital gives us precise control which really assists with user ability to have control over their sound.

Timing ranges are fully programmable via the CV connections as well, which almost fully removes the necessity for external components. This is a huge upside as it greatly reduces the amount of effort and time required to integrate a functioning ADSR. A few other digitally added effects can also be included when certain pins are powered like the punch effect, giving the sound more of a jump. This jump is the same technique that is used in percussion instruments.

The strategy to allow the user to change whether the envelope generator is running linearly or exponentially can be solved with a simple switch between the 5V line and the ground. Additionally, the envelope generator package includes another pin that gives us the option to run it with looping, regular gate triggered operation or with looping behavior. This is a switch between the 5V line, voltage divided, 2.5V line, and ground.

Simple switches and knobs are ideal for what users will be required to manipulate to change their values, so easy access to these options fits nicely with the design outlook for *NepTune*.

Output runs between 0 – 5V and the power for this IC is notably lower at 5V maximum. With the synth running on 12V connection, the lower the power each component takes, especially for ICs that are needed per voice, the better for NepTune's performance. Now

that 3 devices have been discussed it is a good time to breakdown each aspect so a conclusion on which part to include in the final design can be made.

**Table 4.19: Pros and Cons for Various ADSR ICs.**

	AS3310	AS3360	ENVGEN8
Power	12V - 15V	12V - 15V	3.3V - 5V
Channels	1	1	1
Analog/Digital	Analog	Analog	Digital*
Timing Range	1ms - 10s	1ms – 5s	1ms - 10s
Linear/Exponential	Configurable	Exponential	Configurable
Integration Difficulty	Mediocre	Difficult	Easy
Price (each voice)	~\$6	~\$6.40	~\$5.25

### Final Choice for Envelope Generation Parts Selection

The research into finding an ADSR chip/Design that could manage multiple channels was quickly halted once the team discovered how complex that integration would be. Market available ADSRs with multiple channels cost hundreds of dollars and the design one is poignantly outside of the scope of this design. With that note in mind, Neptune is left with ENVGEN8 satisfying almost all of the requirements with the best performance including the cost of each voice added. Each voice will need one IC to work, so it will scale at 1:1 ratio unlike more effective IC sections. It is critical for these noise sensitive parts to be far from the power generation, so modular PCB design will influence the design to be separate from anything before the ADSR stage. Connected via power connections and output connections from the MCU digital PCB board.

ENVGEN8 is the best selection from the research and comes with handy unique features that the will be able to market as an extension from the basic goals of designing a music synthesizer.

### 4.2.10 Audio Mixer

In order for audio outputs from the VCA to be played out of the speaker/audio line out, the voices must be combined with a mixer. It will add all of the voices to a single signal which then can be played harmoniously through output. It is incredibly important that the part included for the final design is high enough quality to not disturb the sound in any way.

This means that any noise, phase difference, delay can all severely impact the effectiveness of *NepTune*'s operation.

Many mixers are either reconfigured VCAs, which may not be required for this section as each signal is already amplified according to the envelope generator and volume knob before mixing. It is still important to compare this reconfiguration against regular mixer ICs in order to paint a picture or performance. If the team decided to integrate the VCA portion into the mixing process, it would certainly be possible but might unnecessarily add to the complexity of the design beyond reasonable integration.

### **SSI2190**

The Sound Semiconductor Voltage Controlled Mixer is a chip that immediately appeared promising for *NepTune* as the manufacturers are also responsible for a few other selections along with showing outstanding performance and documentation. Documentation is an important upside that should not be discounted as building with these ICs for the first time can be daunting given the extremely tight deadlines presented.

The SSI2190 contains six channels meaning that as long as the voices produced by *NepTune* are not above six, only one IC package will be required for the final product. This helps reduce the cost of manufacturing as mixers are fairly expensive compared to other IC packages and with the large pin count that this mixer has, it also minimizes complexity of connections for the relevant audio PCB.

SSI2190 operates linearly with no gain control because no VCA is included within the package. Fortunately, this is not required because the VCAs follow the ADSR envelope generators and preceding the mixer. It rates very low noise at a typical  $-91\text{dBu}$  and distortion at 0.025% which matches nicely with high quality synths. The chip operates anywhere from 4V to 18V, meaning the 12V maximum delivery that the device is currently planned on containing will suffice. Along with all of those upsides, the datasheet also boasts that only a small number of external components are required for operation.

### **SSI2130**

This IC operates as a mixer without having the direct intention to serve that section explicitly. The SSI2130 is a VCO that comes with a five-channel mixer meaning *NepTune* could include up to five voices without the need for a second IC. What sets the SSI2130 apart is the inclusion of VCA on each line allowing for gain control of the sound. This can allow the user to once again change the volume of each voice before mixing. This is already covered by the VCAs the design has adopted so it may replace those VCAs and be integrated into both of those responsibilities or if the scope of the design calls for a more partitioned layout, could prove to be unhelpful for *NepTune*'s purposes.

It offers low noise and includes exponential VCAs if necessary, meaning that the SSI2130 will be properly tuned for music synthesis, which likely will be true for any Sound Semiconductor IC. This requires fewer precision calculations for the design team to implement, significantly reducing design complexity and risk of failure.

## CoolAudio V2164

The CoolAudio V2164 chip is another non-dedicated mixer IC as they are quite rare but can be configured to operate as one by summing up the outputs as there are 4 VCAs which can operate under exponential response. This means again that the quality of the sound will be more realistic and outperforms linear VCA alternatives. It has a longer legacy of inclusion with older models of audio synthesizers meaning there is significant documentation that could help the team with the installation of it on the PCB as there are many important details to audio signals which require dedicated knowledge of operation. Notable details are the importance of separating the power signal from the audio signal as noise is a serious obstacle to creating quality sound.

The inclusion of VCAs allows for us to possibly replace the VCA and mixer separation by including both tasks within the same chip, however, the channels can only be used for one operation meaning that if the team decides to opt into using this chip for VCA, there must be summing at the end of the process, which could be unintentionally worse than having the separate operation between different ICs. This is because the team has access to more dedicated ICs for each task so there could be a degree of specialty missed if one IC was chosen to operate both tasks of VCA and mixing.

*Table 4.20: Pros and Cons for Various Audio Mixer ICs*

	SSI2190	SSI2130	CoolAudio V2164
Noise	-100dBu	-106dBu	-94dBu
Classification	Linear Mixer	Stereo Mixer	Quad VCA
Channels	6	2	4
Gain Control	Fixed (None)	Exponential VC	Configurable VC
Distortion	0.003%	<1%	0.02%
Power	12V – 15V	12V – 15V	12V - 15V
Price Per	\$5.36	\$5.36	\$4.40

This has proved to be one of the more difficult comparisons as there are clear upsides and downsides to each part. Stereo mixing is ideal for the sound but only is easily implementable with a part that will impact the sound quality negatively compared to, for instance, the SSI2190. The price is acceptable for each part, but it should be pointed out that the SSI2130 would require double/triple the ICs to support the same number of voices as the other two options. Gain control is not required as it can be handled by the VCA section separately and so is not the biggest factor for decision making. The distortion of

the SSI2190 being the lowest is also massively influential to pick this part in particular as the quality of the audio signal and avoiding noise/distortion is of upmost importance. The consensus is that the SSI2190 satisfies most requirements for inclusion and also provides the easiest integration meaning the team won't need to over dedicate time to an IC that mostly is already configured.

#### 4.2.11 Audio Output Connections

In order for any synthesizer to function, an audio output is necessary. Because the goal of this design is to be portable but also have some use in a studio setting, it is important to both have a line-level out and headphone out. This dual-output configuration will add complexity (requiring both headphones and line-level drivers) but is standard amongst many synthesizers and will be important in *NepTune*.

##### Technology

There are two technologies to consider when determining what would be most appropriate for *NepTune*. Wireless and wire. Bluetooth connections would offer a wireless connection that would allow the synthesizer to pair with speakers without the cumbersome wires, which is an asset when it comes to portability. Moreover, Bluetooth is sometimes used in synthesizers to allow for wireless editing, and virtual midi control which would make it convenient for studio usage. However, adding Bluetooth to the design would add a layer of complication to the design that might not be fulfilled within the acceptable time constraint of the project, and the design would have to be modified to account for latency in audio which makes it slightly more inconsistent to play.

On the other hand, having a cable for an output while inconvenient allows for an uncompressed and full-quality signal with no latency, or signal degradation and can function over a larger range in comparison to Bluetooth, which makes it a better choice to be used in a range of situations including recording and live performance. Also, wired connections offer more stability while avoiding interferences. Therefore, the team decided to utilize cable as it is simpler to design its output, offering a more reliable connection, while avoiding additional unnecessary complexity.

##### Parts choice

Once it was determined to use cables, the next decision lay in deciding what type of jack to use. There are two main options to consider: 1/4" -inch jack and 3.5mm jack. The main difference between these besides the wire diameter is the usage. 1/4 inch jacks tend to be used for more professional audio equipment due to its larger size that provides more stability, and it has better capabilities when it comes to fighting noise. The 1/4 inch jack tends to use balance tip-ring-sleeve connections, which cancels electromagnetic interference. The 1/4 jack is also designed to better preserve signal strength due to its larger

surface area when compared to the 3.5mm. Moreover, 3.5mm is utilized for smaller devices, therefore  $\frac{1}{4}$  inch is typically preferred for larger devices, and it offers a more durable option.

#### 4.2.12 Audio Output Amplifier

Now that the medium for outputting audio has been selected (wired connections via  $\frac{1}{4}$ " jacks) it is time to define the details of its implementation. The output stage will be designed to support simultaneous audio delivery to both headphone and line-level outputs. To successfully achieve this functionality, extra care is necessary in selecting technology and parts.

##### Technology

As with most other audio output amplifiers, the amplifier in NepTune will utilize operational amplifiers (op-amps) to drive the output signal. However, drafting a sufficient schematic involves more than selecting a generic op-amp. Op-amps come in various types, each optimized for specific applications and performance criteria. Care must be taken when choosing an op-amp, as the characteristics of the op-amp can greatly affect system performance.

When picking an op-amp for a specific application (in this case, an audio output amplifier), a well-rounded understanding of op-amps is necessary. The better the engineer understands the characteristics, the better he or she will be at picking the right one for the task. Some common op-amp characteristics are listed below:

- Input offset voltage: The voltage that must be applied between the two input terminals of the op-amp to obtain zero volts at the output. This offset is amplified by gain, so that even a small input offset voltage can result in large changes at the output for systems with high gain.
- Common-mode rejection ratio: The ratio of the common-mode gain to the differential-mode gain [40]. In other words, it is a measure of how well the op-amp is able to reject signals that are common to both input terminals.
  - Common-mode gain: The gain applied to signals common to both input terminals.
  - Differential-mode gain: The gain applied to the difference between both input terminals.
- Input bias current: The “DC currents flowing into or out of the amplifier’s input pins to create a defined operation point during normal operation” [41]. This characteristic depends on the technology used to create the op-amp input terminals. MOSFET-based op-amps require lower input bias current as opposed to BJT-based op-amps [41].
- Noise: “Unwanted signals that corrupt, mask, or interfere with the desired signal which is being processed by an electronic circuit” [42]. Op-amp noise can be classified into input voltage noise and input current noise [43].

- Total harmonic distortion: The ratio of the sum of the powers of all harmonic components to the power of the fundamental frequency. In other words, it is a comparison between the undesired harmonics and the desired harmonics.
- Slew Rate: The speed at which the op-amp is able to react to changes in the input.
- Output voltage swing: The range of output voltages the op-amp is capable of producing.
- Short-circuit current: The max current the op-amp is able to safely source.
- Phase margin: The difference between phase at unity gain to  $-180^\circ$ . Essentially, it is a measure of the stability of an op-amp when placed in a feedback configuration. It indicates the level of phase shift the system can tolerate before oscillating due to instability.
- Quiescent Current: The current drawn when a circuit is in a “quiet” state with no load and inputs are not cycling.

Once an understanding of the basic characteristics of op-amps is established, the next step is to explore the various types of op-amps. It is necessary to examine how specific parameters are optimized for certain applications so that the best type can be chosen for application in *NepTune*. There are various types of op-amps, such as precision op-amps, low-noise op-amps, high-current output op-amps, and more. Because there are so many, only a few will be examined.

### **Precision Op-Amps:**

Precision amplifiers are primarily used in applications where slight disturbances must be detected and amplified with high precision. This is especially important for low-current and low-voltage sensing. The strongest traits of these op-amps are high common-mode rejection ratios, low input offset voltages, low input bias current, and low noise [39]. These parameters are important for NepTune since the output of the mixer is capable of only producing  $\sim 300\mu\text{A}$ . The op-amp responsible for amplifying this signal must be capable of accurately reproducing the signal with little distortion.

### **Low-noise Op-Amps:**

Low-noise op-amps focus on maintaining high signal stability while minimizing input voltage and current noise. These parameters are important for NepTune to achieve an SNR well above 40 dB. Additionally, many low-noise op-amps are optimized for audio applications, making them suitable candidates for the output amplifier.

### **High Current Op-Amps:**

High current op-amps typically have high short-circuit current capability ( $>100\text{mA}$ ), a wide power supply range, and high slew rates. These characteristics allow it to perform well under high-power applications. In addition, they also minimize parameters such as noise and total harmonic distortion. All these characteristics make this type of op-amp seem suitable for the headphone driver, since loads can reach as low as  $8\Omega$ s. In addition, the op-amp should either have a current-limiting system or be capable of dissipating high power should the  $1/4"$  jack be improperly inserted.

## Final Choice for Output Amplifier Technology:

The table below illustrates the important classifications and helps to distinguish them from one another.

*Table 4.21: Pros and Cons for Types of Operational Amplifiers*

	Precision Op-amps	Low-Noise Op-Amps	High Current Op-amps
<b>Parameters to maximize</b>	Common-mode rejection ratio	Stability	Short-circuit current Power supply range Slew rate
<b>Parameters to Minimize</b>	Input offset voltage Input bias current Noise	Input current noise Input voltage noise Total harmonic distortion	Noise Total harmonic distortion

Because this application is audio and does not require precision to the same degree applications such as sensing circuits, this leaves the best choice of technology to low-noise and high-current op-amps.

However, as it turns out, the output amplifier comprises of two stages and a combination of the two remaining technologies may be necessary. First, there is a transimpedance amplifier, converting the current output of the mixer to a voltage. This stage is commonly called the current-to-voltage (I-V) stage and low-noise performance is imperative. The next stage is the output driver, which drives the line-level or headphone loads. In this stage, high current output is important, in order to drive heavy loads such as  $8\Omega$  headphones.

For this reason, the first stage (I-V) will employ a low-noise amplifier, while the second stage will employ a high-current amplifier. A combination of these technologies should provide sufficient performance for this audio application.

### Part Selection

Now that the technology for each stage in the output amplifier has been established, it is time to select the specific op-amps that will be used to create this circuit.

#### Transimpedance (I-V) Stage:

To achieve a high-fidelity audio system, any op-amps considered for the I-V stage must have ultra-low noise and distortion for high SNR, THD, and dynamic range. High slew rates are desired for quick transient response, and high bandwidth capabilities for a flat frequency response. Additionally, tolerance for higher voltage power supplies can boost the signal-to-noise ratio through higher output voltage swings. The best candidates are shown below, with the selected component highlighted.

**Table 4.22:** Pros and Cons for Stage 1 (I-V) Operational Amplifiers

	AD 797	OP275	ADA4075-2
<b>Noise</b>	0.9 nV/ $\sqrt{\text{Hz}}$	6 nV/ $\sqrt{\text{Hz}}$	2.8 nV/ $\sqrt{\text{Hz}}$
<b>THD</b>	-98 dB @ 250kHz -120 dB @ 20kHz	-84dB @ 1kHz	-113dB @ 1kHz
<b>Slew rate</b>	20 V/us	22 V/us	12 V/us
<b>Input Power supply</b>	+/-5 V to +/15 V	+/-4 V to +/22 V	+/-4 V to +/18 V
<b>Number of op-amps</b>	1	2	2
<b>Cost</b>	\$18.12	\$5.65	\$3.74

The decision on which part to employ is made much simpler using this table. The characteristics that matter the most for application in *NepTune* are listed above for each respective part. The first option, AD797, will not be selected due to cost, although it has exceptional performance. The OP275 and ADA4075-2 have characteristics very closely related to one another. However, the ADA4075-2 is selected because of its superior noise and THD performance as well as being a bit less expensive. Hopefully, the lower slew rate will not negatively impact the sound performance.

### Output Driver Stage:

Similar to the I-V stage, this second stage (output driver) should have excellent low-noise and low-distortion performance for high SNR, THD, and dynamic range. High slew rate is required for quick transient response and high bandwidth for flat system frequency response. The main difference between this stage and the last, is the ability to drive heavy loads. Thus, the defining characteristic of the op-amp for this stage will be its ability to source high currents.

**Table 4.23:** Pros and Cons for Stage 2 (Output Driver) Operational Amplifiers

	ADA4841	ADA4807-2	AD8397
<b>Noise</b>	<p><b>@ 100kHz</b></p> <p>Input voltage noise 2.1 nV/<math>\sqrt{\text{Hz}}</math></p> <p>Input current noise 1.4 pA/<math>\sqrt{\text{Hz}}</math></p>	<p><b>@ 100kHz</b></p> <p>Input voltage noise 3.1 nV/<math>\sqrt{\text{Hz}}</math></p> <p>Input current noise 0.7 pA/<math>\sqrt{\text{Hz}}</math></p> <p><b>@ 10 Hz</b></p> <p>Input voltage noise 5.8 nV/<math>\sqrt{\text{Hz}}</math></p> <p>Input current noise 10 pA/<math>\sqrt{\text{Hz}}</math></p>	<p><b>@ 100 kHz</b></p> <p>Input voltage noise 4.5 nV/<math>\sqrt{\text{Hz}}</math></p> <p>Input current noise 1.5 pA/<math>\sqrt{\text{Hz}}</math></p>
<b>THD</b>	-97 dB @ 100kHz	-112 dB @ 100kHz	-84 dB @ 100kHz
<b>Slew rate</b>	12 V/us	225 V/us	53 V/us
<b>Input Power supply</b>	2.7 V to 12 V	2.7 V to 11 V	+/-1.5 V to +/-12.6 V
<b>Short-circuit current</b>	30mA	80mA	310 mA (AC current only)
<b>Number of op-amps</b>	2	2	2
<b>Cost</b>	\$8.49	\$6.89	\$8.49
<b>In Stock?</b>	Yes	No	Yes

All three choices of op-amps appear to be sufficient. Either one could make for a decent output driver. The second option, however, stands out the most, as it offers the lowest cost, with the best THD performance and fastest slew rate. It can also source up to 80mA. Unfortunately, it is out of stock during the design of this project. The remaining two choices offer almost identical performance for these specific characteristics, but since the AD8397 has a higher slew rate, it will be chosen for the output driver. The only drawback to this component is the datasheet lists no short-circuit limit for the op-amp unlike the

others which make it very clear. However, it has a high AC current capability, which provides a promising chance it has a high DC current capability.

#### 4.2.13 Power Supply

Power supply is essential to the functioning of *NepTune* as it will be used to power the various ICs. In order to determine what power supply to design for the device, multiple things must be taken into consideration. First, the power supply design must generate the least amount of noise possible so as not to interfere with the sound signal. Second, the power supply will most likely need to be a 12V input in order to support the various components. However, this power would need to pass through a linear regulator to ensure low noise values. Therefore, the decision was to increase the input power to 15V so as to not lose a huge amount of power when the regulation occurs.

Although the wall wart is responsible for altering the incoming AC power to DC power, regulation is still needed to ensure a smoother power signal, and lower noise interference levels. This will be done through an initial regulator that will change the incoming 15V DC power to a 12V DC. Next the power will pass through a series of regulators that will step down the power to the necessary values of 12V, 5V, and 3.3V.

This brought up another concern, however, as power loss will become a factor of consideration. Since power is being stepped down from the initial value of 15V to as low as 3.3V much of the power will be lost. This power loss will generate heat, and since the majority of our parts malfunction at high heats, there is a potential for a malfunction of the parts. Therefore, a heat sink may need to be created. This will be determined after we test the initial PCB to determine if the function of the ICs will be compromised.

The next order of things was to determine what type of power supply would be the most appropriate. There are three potential options when choosing a power supply in terms of configuration: battery, linear power supply, and switching power supply.

#### Technology

One of the main goals of *NepTune* design is portability, therefore, the power supply form becomes a main aspect to discuss. The main choices pertain to whether to use battery or cable connection power such as a wall wart. There are multiple things to take into account when discussing this choice: life span of power, portability, and projects constraints.

The first thing that was considered was utilizing a battery. Based on components used, a battery of 15V would be needed to fully supply the circuit. This is a problem for multiple reasons. A 15V battery is not available for purchase, which would yield two options, either, utilize multiple batteries to generate a strong enough current that would power all of the components, or the other option would be to get a 24V battery and then step it down to the necessary voltages. This would cause a bigger issue in regards to power loss, as heat dissipation would be way larger.

Moreover, while offering portability to the device, would be quite large in size, after examining the various batteries sizes the space either of these choices would require a

redesign of the PCB and the other components forcing a reconstruction of the final design or alternation of the size, which would deviate from the goals of the project. Life span would also be a problem as well. For instance, the average 24V battery delivers approximately 9Ah, if the approximate the load current to be 2A, that is an average battery life of 4.5 hours which is okay, but not ideal for professional usage.

On the other hand, if a wire connection is chosen there would be no need to worry about life span of the device, especially if the team chooses a wired wall wart. This, however, would make portability slightly more inconvenient as the user would need to carry the wall wart around. But, this would allow for more liberty in the design and for the initial expectation of the size of 12 x 18 to be achieved more easily.

### Part Choice

Linear power supply is a power supply that operates by taking the incoming AC voltage and stepping it down to whatever voltage is utilized by the circuit, in this case 12V, then this current is rectified by diodes in a rectifier circuit, and smoothed by a capacitor to produce a DC voltage, which is further smoothed via a filter circuit [11]. In a switching power supply, however, the incoming AC voltage is first rectified and smoothed to produce a high DC voltage, which is then passed through a switching transistor that converts it into a pulsed DC signal. This is done by the transistor constantly switching on and off generating high frequency waveforms, which is stepped down and rectified and filtered into desired DC output voltage [11].

The most imperative difference between these power supplies is the output noise level, and the size of the power brick. Since switching power supplies operate by essentially turning on and off, they tend to generate electromagnetic noise. This could generate up to 10x as much noise in comparison to the linear power supply [11]. This can be rectified, nonetheless, through the addition of a filtering circuit, however, it would add a layer of complexity to the design. The other key difference is the size of each power supply. A linear power supply tends to be bigger and bulkier because it contains a larger transformer. This large difference in the size of the transformer is attributed to the operating frequency of each power supply, linear operating at the input line frequency of 50/60Hz versus the switching power supply which converts the signal into higher frequencies and utilize smaller transformers. [11]

These are both issues that can affect the design greatly. The bulkiness of the power supply would limit the portability of the synthesizer, however, using a switching power would expose the circuit to an additional level of noise that would need to be mitigated especially since *NepTune* is an instrument meaning any additional noise would greatly distort its performance.

For *NepTune* the decision was to utilize a switching power supply, as it is readily available and cheap to purchase, and not as cumbersome to replace or carry around. However, that would mean that a rectifying circuit is needed to eliminate the noise produced by it. Moreover, the microcontroller needs a 12V switching supply, whilst the rest of the circuit

requires a linear supply. This would indicate a need for designing a circuit that would transfer the switching supply into a linear one to be used by the remaining

Since the power supply has been selected. The next step dictates determining how to situate the ground. The method of grounding is very impactful to the design as the sound signal may be impacted if the parts are not grounded appropriately. A very common issue with audio signals is a condition called ground loops. Ground loops occur when two separate devices are connected to ground via different paths, and then also connected to each other via another path creating a loop; This creates another current flow that causes interference to nearby devices [12]. This can be easily mitigated by separating the analog and digital circuitry, and having separate grounds for each, and eventually reconnecting them at one point.

One of the main issues to be addressed is the power needed for each individual component. This introduces the need for multiple voltage regulators. As of right now the different power supplies needed are:

*Table 4.24: List of Parts with Power Requirements.*

Part	Description	Digital (D) or Analog (A) Power	Input Voltage (per IC)	Input Current (per IC)
ADG706	MUX	D	+3.3V	10mA
AD5206	DigiPOT	D	+3.3V	TBD
PCM3060	CODEC	A & D	+5V(A) & +3.3V(D)	10mA
ENVGGEN 8D	ADSR Env. Gen.	A	+5V	5mA
VCLFO 10	LFO	A	+5V	5mA
TL074 (Not acquired)	Op-amp	A	+/-12V	10mA (for TL084)
STM32H750IBT6	MCU	D	+3.3V	TBD
IS25LP064A- JBLE-TR	Flash Mem.	D	+3.3V	TBD
SSI2190	Mixer	A	+/-12V	11.4mA
SSI2164	VCA	A	+/-12V	6mA
SSI2140	VCF	A	+/-12V	11 mA

*Note: This table is not a representation of what each part requires but is instead a representation of what it will be given. For instance, the VCLFO10 does not require analog power supply (non-switching source) but it will be given this type of power, due to various design constraints.*

Since all these components utilize different input voltages it is important to account for all the different circuits needed for the synthesizer to function, including the voltage regulation circuits that would be needed. Another objective to consider is the heat generated by the circuits. Thermal loss is a major factor to consider in a synthesizer because if the temperature inside the case reaches beyond the operating temperature of the IC's this will cause the parts to malfunction, and ruin the output signal produced, and therefore the sound signal.

## Voltage Regulators

An important part of power circuits is voltage regulators. Voltage regulators are used for a multitude of reasons in *NepTune*. One of the main reasons is to smoothen the incoming voltage signal, and minimize noise interference. Since the voltage is an AC source that is converted to DC, there is noise associated with this transformation. Another reason is that *NepTune* utilizes multiple voltages for the various ICs, 5V, 12V, and 3.3V, there is a

need for transformation, and maintenance of multiple steady signal, and therefore multiple regulators.

First, the VCF, mixer, VCA, and all the operational amplifiers use the 12V input. Since these require an analog signal for operation which contains both a negative and a positive cycle. The inputs to these parts must be made up of a +12V, and -12V to ensure both the positive part of the signal and the negative part of the signal are used by the parts. This is also true for the parts used for 5V. Since these require both portions of the signal to be regulated there is a need for both a negative signal voltage regulator, and a positive signal voltage regulator. This is also needed for the 5V voltage supply.

The other power supply needed is 3.3V. In this case the signal needed to power on this device is a digital signal. Since digital signals are made up of zeros and ones, and therefore only one voltage regulator to manage the positive portion of the signal. The MCU, DigiPots, codec and the MUX utilize this power source. After identifying the amount and general types of regulators that are needed, the next step is to identify which type of regulators are needed.

### Technology

There are two main types of regulators that are being considered for usage. Linear regulators and switching regulators. These are considered for their simplicity in design, low cost, and general effectiveness within the design.

Linear regulators operate by controlling the output voltage via the load connected that forces a fixed voltage to appear at the regulator output terminal, and dissipating the excess input voltage as heat. They tend to use elements such as transistors to adjust resistance and ensure the output remains constant. There are multiple types of linear regulators, and their differences lie in the dropout voltage, which is the minimum voltage drop required across the regulator to maintain the output voltage regulation. Most linear regulators because their efficiency is dependent on the voltage drop being the smallest as to dissipate the least amount of internal power. Therefore, the larger the voltage drop the less efficient linear regulators tend to be.

Linear regulators are the best choice for low noise applications because they generate limited noise in comparison to switching regulators which makes them perfect for *NepTune* especially since there are a multitude of analog parts that could be impacted by a noise, which would affect the output audio signal. Linear regulators also have a fast transient response, which is important when generating an audio signal.

Linear regulators are inefficient for high-power applications or ones with large voltage differences, as excess energy is lost as heat. However, in *NepTune's* applications the

voltage drop is not that large as the max drop would be between 12V to 3.3V which would not be enough to generate a concern, however, testing is needed to ensure that.

Switching regulators, on the other hand, utilizes a switching power transiter to convert one DC voltage to another, by rapidly switching on and off, then filtering the output to maintain a stable voltage. Switching regulators are more efficient because they store energy inside inductors or capacitors and release the energy in them in controlled pulses, which makes them very efficient.

Switching regulators tend to be ideal for situations where there is a large voltage difference, as it is very efficient and releases minimal heat. However, they generate a lot of noise due to rapid voltage transitions, which would mean, if implemented they would need a noise filtering circuit. Moreover, they require more components and therfore a larger footprint within the PCB.

Thus, based on the research conducted. The choice in *NepTune* was to utilize linear regulators as they are cost effective, provide minimal noise, which is excessively imperative in an analog audio circuit that has a signal that cannot allow for interference or the performance of the machine is compromised. Finally, a linear regulator is also better due to the simplicity of the design requirements.

### Parts

There are multiple parts for consideration when it comes to which linear regulators to use in *NepTunes* design. The parts were chosen to be examined LM2576, LT3045, TPS7A47, LM317, and L78XC series. These were selected for a mix of noise sensitivity, cost effectiveness, and footprint of part. The objectives that are to be examined are:

- Cost
- Noise generation level
- Output Voltage
- Current Generators
- Package Type

### **LT3045**

The LT3045 is a low dropout linear regulator, with low noise, and a power source rejection radio architecture specifically designed for powering noise sensitive applications. This makes it excellent for this application, moreover, this IC has a wide voltage range between 1.8V to 20V, with an adjustable output between 0-15V, which meets our requirements. Moreover, this IC provides a high current of approximately 500mA output current and only requires a minimum of 260mV to operate with that much current, which means that even

if the  $V_{in}$ - $V_{out}$  are very close in range, the regulator will have no issue operating as needed. LT3045 is also equipped with a thermal shutdown feature which stops the operation of the regulator when the  $T = 165C$ . This offers protection as to not affect the other parts of the analog circuit.

### TPS7A47

The TPS7A47 is a low-noise, high-PSRR, low dropout voltage regulator by Texas Instruments. It has an input voltage range of 3V to 36V, and an any-out user programmable output voltage that ranges from 1.4V to 20.5V, with an output current of 1A at a dropout voltage of 307mV. This indicates that it drives a very significantly high current at a very low dropout voltage. This linear regulator also comes with a thermal shutdown, which as stated earlier, could be very useful in our design that multitude of components that are temperature sensitive. The TPS7A47 has a higher thermal shutdown protection though of 170C. The downside of this regulator though is that it only comes in QFN package which is not solderable in lab and would make it hard to adapt this into *NepTune* PCB design.

### LM317

The next linear regulator to be examined is the LM317 which is an adjustable regulator with an output range of 1.25V to 37V, with an output current of 1.5A. While this has a thermal regulator the maximum operating temperature is 150C max which is on the lower end, but not necessarily a concern unless that much heat is generated through our voltage dropout which is not likely for our usage. One of the main advantages of this regulator is the packaging, which are various configurations of surface mount, which is easy to solder within the equipment available. The other disadvantage is that the configuration of this would be dependent on outside components which would an additional circuit design and components, although it is a simple task.

### L78XC series

The L78XC is a series of fixed output voltage regulators with an output current of up to 1.5A and varying output voltages depending on the one needed. For *NepTune*, L7805, and L712 are being examined for usage. These regulators also contain thermal overload protection, and operation temperature of 125C max. These are also available in a multitude of packages including through hole and surface mount, which yields multiple options to how to attach them to the PCB board. The dropout voltage required is larger than most other regulators discussed with a need of a 2V drop in order to draw 1A at the output.

### LM2576

The final regulator to be discussed is the LM2576xx, a series of fixed power converters where the input voltage is up to 40V and has a fixed output voltage based on choice. This was chosen due to one of the advisors' recommendations for the 3.3V switching to power on all digital components of *NepTune*. This regulator also contains a thermal shutdown for protection, and cycle-to-cycle current limiting, which monitors the current flowing through, and if it exceeds anywhere between 4.2A-6.9A it forces the regulator to terminate the current switching cycle. These regulators have through-hole footprints which are very useful for soldering purposes. Since noise interference is the main issue concerning switching regulators, this will be dealt with through separating the digital components from the analog components.

	LT3045	TPS7A47	LM317	L78XC Series	LM2576
Cost	\$10.45	\$6.28	\$0.41	\$0.48	\$8.51
Noise generation level	Ultralow	Ultralow	Low	Low	High
Output Voltage	Adjustable 0-15V	Adjustable 1.4V to 20.5V	Adjustable 1.25V to 37V	Fixed (12V), (5V)	Fixed (3.3V)
Current Generated at Voltage drop	500mA at 260mV	1A at 307mV	1.5A	1A at 2V	0.5A to 3A MAX

<b>Package Type</b>	-DD PACKAGE -MSE PACKAGE	-VQFN (20)	-4-pin, SOT-223 –3-pin, TO- 263 – 3-pin, TO- 220	-TO-220 -TO-220FP -DPAK	-TO220B- 5L -TO220-5L -TO263-5L
---------------------	-----------------------------	------------	-----------------------------------------------------------------	-------------------------------	------------------------------------------

After comparing all the choices, the decision was to utilize L7812 and L7805 (12V and 5V in the L7XC series), and the switching regulator LM2576. This was done because cost is one of the major constraints to consider. Ordering multiple of each of the more expensive options would be a considerable burden on our budget. One of the other main reasons is that the fixed voltages offer more ease of use and configuration, and they also are available in the preferred package types which is through-hole in this case. The main drawback of the LM7XC series, however, is that the voltage drop needed to draw out the current is very large. This should not be a huge issue though as our smallest voltage drop is 3V. On the other hand, the switching regulator LM2576 was chosen due to its availability in fixed voltage form, and through-hole footprint, although it is on the more expensive side only one or two are needed for this circuit.

### 4.3 Serial Communication Protocols (SCP)

Serial communication protocols have evolved over the years to accommodate different technologies and solve various hardware needs. Universal Asynchronous Receiver/Transmitter (UART) was the earliest introduced, designed for teletypes and terminals in the 1960s. Serial Peripheral Interface (SPI) was developed in the 1980s for fast, full-duplex communication between chips in PCBs. I2C was introduced in 1982 for simple, low-speed communication with peripherals over just two wires. Each protocol, evolving as hardware becomes more complex and compact, emerged from a specific industrial need: simplicity, speed, distance, or bandwidth.

A serial communication protocol (SCP) benefits synth development by letting your microcontroller control and interact with many digital and analog-supporting components without needing a large number of physical pins. Protocols such as SPI or I2C enable the synth to talk to parts like audio codecs, external memory, or displays, UART can be used to receive MIDI messages, and SAI (running I2S) can be used to stream audio between the MCU and the codec.

In this project, several Serial Communication Protocols are needed to communicate from the MCU (STM32H750IBT6) on the Daisy Seed to the following peripherals:

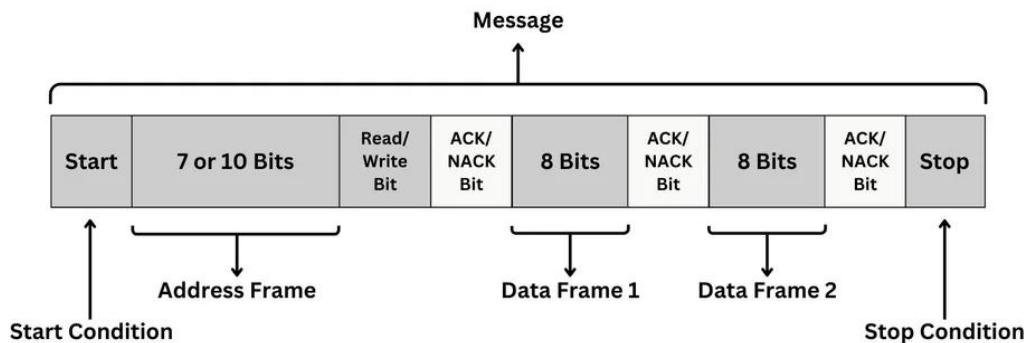
- Communication to DIN-5 MIDI ports (UART)
- Communication to external 2x PCM3060 CODEC (I2C or SPI)
- Communication to external AD5206 Digital Potentiometer connected to the VCA, VCF, LFO, and ADSR (SPI)

In the above peripherals, each one comes with their own data sheet containing the appropriate SCP to use as well as its respective configuration required to smoothly transmit data. Therefore, this section will be a comprehensive description of the SPI, I2C and UART protocols focusing on how they communicate with their respective peripheral and their data transmission rates.

#### 4.3.1 I2C

I2C (Inter-Integrated Circuit) is one of the most common serial protocols around where it is primarily used for short distance communication. It is a synchronous master-slave type protocol which operates in half-duplex mode, meaning data flows in one direction at a time, synchronous to the clock. I2C as a master-slave type protocol refers to a microcontroller (master) and one or more peripheral devices (slaves) using just two wires, SCL (Serial Clock Line) and SDA (Serial Data), in an open-drain configuration, requiring pull-up resistors on both lines.

With the level of complexity required in the synthesizer project, the simplicity of using just two wires to connect to the PCM3060 is desirable. The master connects to SDA and SCL and exchange data over this shared bus in messages which are broken up into frames of data.



**Figure 4.12:** Frame diagram of an I2C message [12].

The above diagram of I2C's configuration of data packages is the reason for significant overhead. Start and stop conditions, 7 or 10-bit addressing, a read or write bit, and acknowledge (ACK)/not-acknowledge (NACK) bits are the cause of delays in data rate with 7 or 10-bit addressing causing most of the overhead. When the master is connected to multiple slave/peripherals, I2C uses a slave's unique binary address to "select" and send data to it or read data from it. With more peripherals, the overhead of addressing increases.

Fortunately, the overhead linked with addressing and constant switch between a read or write bit over the SDA can be overlooked in the case of communicating with the PCM3060. Out of the devices NepTune needs to communicate with, the PCM3060 is only one that supports I2C, meaning there is only one address that the MCU needs to write to. On that note, the MCU is only sending data to the PCM3060, not reading from it. So, there is no need to switch the R/W bit to Read.

Looking further into I2C the team saw there are several clock speed modes that the STM32H750IBT6 MCU supports. There are three modes: Standard mode (100kHz), Fast mode (400kHz), and Fast mode + (1MHz). All of which will be compared to the SPI's clock speeds later against the project's needs.

### 4.3.2 SPI

SPI (Serial Peripheral Interface) is a high-speed, full-duplex communication protocol used to connect a microcontroller with one or more peripheral/slave devices. It operates using a 4-wire interface: MOSI (Master Out, Slave In), MISO (Master In, Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select or Chip Select). The master sends a clock signal through the SCLK wire to the slave. Using the CS, the master can communicate with multiple slaves. The master device controls the clock and initiates data transfers, while each slave responds when selected. The MOSI line sends data from the master to a slave, while the MISO line sends data from a slave back to the master. Since these are independent lines, data can be sent and received at the same time, reducing delays when communicating with ADCs and DACs and enabling tighter timing in audio related systems where latency matters.

In an Independent Slave configuration, each slave shares the same MOSI, MISO, and SCLK, but has their own independent CS wire connected to the master. If a master was connecting to 2 peripherals, 5 wires would be needed. In this project, NepTune must communicate with the AD5206 DigiPot and the option of using SPI on the PCM3060. Unlike some peripherals that require both MOSI and MISO, these devices only receive data from the master. Hence, they only require 3 wires: CS, MOSI, and SCLK. So with both devices using a 3-wire SPI, the team only needs 4 wires to connect to both devices from the Daisy Chain: MOSI, SCLK, and 2 Chip Selects.

When it comes to data speed, SPI is not bound by set modes. Rather than speed modes, SPI is subject to 4 modes (0, 1, 2, and 3) by CPOL (Clock Polarity) and CPHA (Clock Phase). This method greatly improves the speed SPI can transmit data, making the only limitation of data speed the max SCLK frequency (~10 MHz) and general signal integrity, according to the PCM3060 datasheet. However, at the cost of such low overhead comes with the use of more GPIO pins, making it less efficient in terms of pin count, and no built-in error handling such as the ACK/NACK bits of I2C that acknowledges when a slave is receiving signal from the master.

### 4.3.3 I2C vs. SPI – PCM3060

Between using I2C or SPI for communication between the MCU to PCM3060, there are several factors that must be discussed and considered. Among them are the number of pins required for either SCP connection to the PCM3060 plus the pins used for SPI connection to the AD5206 DigiPots. There is also clock speed, error handling, and the amount of overhead that must be considered.

The comparison made for the number of pins used is either a 2-pin I2C connection for the PCM3060 and a 3-pin SPI connection for the AD5206 or a 3-pin SPI connection for both

external devices. Utilizing the latter method would imply both devices would share the SCLK and MOSI lines while having their own CS line, meaning 4 pins used. On the other hand, the former method gives up two pins for the SDA and SCL lines for I2C while the AD5206 uses three separate pins (SCLK, MOSI, and its own CS) for SPI for a total of 5 pins used. Therefore, the difference between I2C and SPI in terms of pins used is one pin. A trivial and negligible comparison and one that hardly constitutes SPI on PCM3060 being a better choice over I2C. In contrast, the decision to give each device its own communication protocol removes the overhead of two peripherals sharing data lines and the complexity of sending data to two places at once.

I2C has several clock speed modes that go up to 1 MHz, while SPI's clock is limited to an astounding 10 MHz. This seems like an easy choice to make at first and in truth, it is. But the question the team needs to ask is if it's too fast. Looking at the role that the communication protocol is responsible for, 10 MHz of clock speed can be viewed as unnecessary. The chosen communication protocol is responsible for resetting the PCM3060, configuring the audio format, and setting up the DAC. In short, it's configuring and resetting multiple registers for future uses. Such commands do not require such high speeds, especially since they will realistically only be called a few times or not that frequently. In this case, I2C's clock speed is more than enough to handle the workload.

While leaning toward the I2C communication protocol, the decision cannot be made without discussing the drawbacks and the inevitable overhead it brings. With I2C, the hardware becomes more complex as the number of peripherals increases. This is due to the addressing mechanics of sending data to multiple devices, which in turn creates more overhead. But as discussed before, this issue can be overlooked as there is only one device connected to the MCU via I2C. Next are the start/stop condition, R/W bits, and ACK/NACK bits. Admittedly, the start/stop bits cause some overhead but are necessary given a protocol using two wires. R/W bits can also be overlooked as the MCU only sends write bits and while ACK/NACK bits seem to be nothing more than move the I2C bus along, they provide crucial error handling. If the slave doesn't respond to the address or data byte, the master receives a NACK.

While writing, the team still felt the appeal of using SPI. That was until the team found out that the undisputed reason to use I2C is because of libDaisy documentation. Although it has finalized the decision to use the libDaisy library (discussed later), code written for the PCM3060 codec configured for I2C communication is already available for use in the libDaisy GitHub repository. The file is responsible for the following:

- Sets up I2C communication with the codec
- Issues two software resets via I2C to initialize the codec
- Configures audio format to 24-bit MSB-aligned I2S
- Disables ADC/DAC power-saving for full operation

Because of this, the documentation for I2C with the PCM3060 creates a huge advantage and it's something the team will certainly consider in the decision for the communication protocol with the most advantages. However, originally overlooked, in the data sheet of

the PCM3060, it is revealed that I2C nor SPI is required for the configuration of the PCM3060 codec, as there is a unique HW (hardware) mode dedicated to it.

#### 4.3.4 PCM3060 HW mode

Taking the short review on why NepTune needs to use a SCP and what the project needs from it, the team sees that there are two things: code to set up the chosen SCP and code to set up the audio data format (SAI/I2S). As the team is essentially building a synth, the latter of the two is the most important part.

While searching through the data sheet of the PC3060, the design team came across the hardware control mode which removes the middleman by rendering the use of I2C, and SPI obsolete in terms of the needs of this project. By setting a pull-up resistor to  $V_{DD}$  or a pull-down resistor to DGND NepTune activates parallel hardware control mode. In this mode, the PCM3060 configures itself based on logic levels of dedicated pins at power-up. These hardware pins determine audio data format (I<sup>2</sup>S, Left-Justified, etc.), master/slave mode, power-down state of ADC/DAC, and de-emphasis and filtering options. Some of the drawbacks include a less flexible interface that won't allow you to change things like sample format, attenuation, or filters without rewiring. Fortunately, none of those things require reconfiguring. Furthermore, no pins are required for hardware mode and unlike serial protocols, they don't have a defined "speed" in the traditional sense (like clock frequency or baud rate). Instead, it works through static voltage levels on specific pins that are latched at power-up. This basically means no runtime communication at all and instead initialization is immediate after power-up.

In conclusion, the benefits of using parallel hardware control for the PCM3060 are summarized as:

- No need for I2C communication/setup code
- Simpler initialization (especially in early prototyping)
- Transference of audio over I2S via the STM32's SAI peripheral
- No pins

For these reasons, the team has ultimately decided to use the parallel hardware control mode for the PCM3060.

*Table 4.25: Pros and Cons for Various Communication Medium*

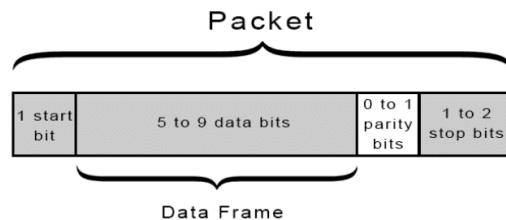
	SPI	I2C	HW Mode
Pins Used (PCM3060 + AD5206)	4	5	No pins used
Clock speed	~10MHz	~1MHz	N/A Instantaneous
Overhead	Low	Medium	None

Library Documentation	No	Yes	N/A
-----------------------	----	-----	-----

### 4.3.5 UART

UART (Universal Asynchronous Receiver/Transmitter) is a widely used serial communication protocol for transmitting data between devices. As it says in the name, UART is asynchronous, which means that there is no shared clock signal and instead the devices communicating rely on pre-agreed configurable baud rates between 9600 bps to 115.2 Kbps. Unlike the I2C or SPI, UART is not a master-slave SCP. Both devices communicating has a transmitter (TX) and a receiver (RX) where the TX of one device is connected to the RX of the other. Hence, there are two lines needed to facilitate communication. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device.

UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional parity bit, and 1 or 2 stop bits:



UART is great for simple, point-to-point links: it needs only TX, RX, no shared clock, supports an optional parity bit for basic error checking, and you can define your own packet format as long as both ends agree. It is ubiquitous and well documented. The tradeoffs are fixed, small data words (up to 9 bits at the hardware level), no native multi-drop or multi-master support, and both sides must run closely matched baud rates (within about 10%).

#### UART over MIDI Communication

In MIDI communication, the standard method of sending MIDI messages is to use a 5-pin DIN cable connector. The 5-pin DIN MIDI connection does not include a clock line, which makes it an asynchronous protocol; exactly what UART is built for.

The original MIDI 1.0 specification, introduced in 1983, is fundamentally an asynchronous serial protocol that matches the capabilities of UART hardware. In this setup, MIDI data is transmitted one byte at a time, using a fixed baud rate of 31,250 bits per second, with 8 data bits, no parity, and one stop bit. Because MIDI is unidirectional, a standard connection consists of separate “MIDI In” and “MIDI Out” ports, each requiring its own UART if bidirectional communication is needed.

Using UART for MIDI is a simple, efficient way to send and receive MIDI messages, especially on microcontrollers like the STM32, where hardware UART blocks are readily

available. Developers can directly write or read MIDI bytes using the UART's transmit and receive registers without needing to interpret timing or framing manually. As long as the correct baud rate is used and the electrical circuit meets MIDI's spec, UART serves as the ideal method for supporting MIDI over the classic DIN connection.

#### *UART between ESP32 and DaisySeed/STM32 (Senior Design 2 Revised)*

This project uses a simple UART link to carry control data from the ESP32 to the Daisy Seed. Both ends run 76 800 baud, 8N1. The line idles high, each byte has a low start bit, 8 data bits LSB first, and a high stop bit. Both MCUs are 3.3 V TTL, so a direct connection works once grounds are tied. ESP32 TX goes to Daisy RX on D11. Daisy TX on D12 can be wired to ESP32 RX if you later add Daisy-to-ESP messages.

The protocol is poll reply. When the Daisy wants fresh controls it sends one request byte 0x69. The ESP32 continuously scans two 16-channel multiplexers, converts the readings to 12-bit values expanded to 16 bits, and waits. On seeing 0x69 it emits a fixed 18-byte frame: 0xAA, then 16 payload bytes for eight 16-bit knobs, then 0x55. Each knob is big endian at the message level, MSB then LSB, while the UART hardware still shifts bits LSB first inside each byte.

At 76 800 baud with 8N1, an 18-byte frame consumes about 180 bit times, roughly 2.3 ms on the wire. The Daisy sends 0x69 with a short blocking transmit, performs a blocking receive for exactly 18 bytes with a reasonable timeout outside the audio callback, checks the start and end markers, reconstructs the eight 16-bit values, and maps them to synth parameters. Fixed size plus unique markers keeps parsing deterministic and quick.

This pull model avoids FIFO overruns and gives predictable control bandwidth aligned with the audio engine. If you need extra robustness, add a CRC byte after the payload and verify before applying parameters. If you want a higher update rate, call the Daisy's polling function on a periodic schedule in the main loop in addition to any MIDI-driven calls, without changing the frame format.

## 4.4 Software Parts and Technology

In the world of electronic music and audio engineering, synthesizers have evolved from bulky analog machines into compact, powerful digital devices. At the heart of many modern DIY or custom digital synthesizers is a development board. A compact, pre-assembled circuit board that integrates a microcontroller or processor along with essential components like power regulation, memory, and I/O interfaces. Development boards provide a flexible and accessible platform for prototyping, experimenting, and deploying embedded systems without needing to design custom hardware from scratch. They act as a bridge between software and hardware, allowing developers to write code that interacts directly with sensors, knobs, displays, audio signals, and other electronics.

While the hardware capabilities of a development board are essential, the software ecosystem surrounding it is just as critical, especially when it comes to building a reliable and expressive synthesizer. The programming language, development environment, audio libraries, and digital signal processing (DSP) tools available on a given platform will shape how efficiently and creatively a developer can realize their ideas. Some boards rely on low-level languages like C or C++ for precision and performance, while others support higher-level scripting or visual patching tools, making them accessible to artists and musicians with less programming experience. In this section, the team will explore the software technologies behind embedded synthesizer development; focusing on the languages, IDEs, libraries, visual patch support, and DSP complexity associated with this kind of work. After covering these key aspects, NepTune will return to the three most promising boards, the Electro-Smith Daisy Seed, Teensy 4.x, and Bela, to evaluate them through this software lens and determine which is best suited for the synth project.

#### 4.4.1 Embedded Language Comparison

When selecting a programming language for embedded audio development, it is crucial to evaluate each option not only in terms of syntax or popularity, but across several core dimensions that directly influence both the developer experience and the system's capabilities. Among the most relevant dimensions are programming paradigm and abstraction, learning curve and accessibility, performance and resource usage, and toolchain and development workflow. Each of these aspects plays a pivotal role in determining how effectively a language or environment supports the development of responsive, efficient, and maintainable software for audio synthesis. In the following sections, the team compares standard C++, Arduino C++, and Pure Data across these four dimensions, offering a detailed analysis of how each environment aligns with the demands and priorities of real-time sound generation and embedded systems design.

##### Programming Paradigm & Abstraction

Programming paradigm and abstraction refer to the style and level of operations used to describe computations. C++ is a general-purpose language supporting multiple paradigms: it offers control flow, object-oriented classes, and low-level memory manipulation. In C++, the programmer must manage details like memory allocation but gains fine control and efficiency. Arduino "sketches" use standard C++ under the hood, but the Arduino environment provides domain-specific libraries and a simple setup/loop structure. By contrast, Pure Data is a visual dataflow language: programs are built by connecting functional objects (boxes) with "patch" cables to route signals and messages. In Pure Data, the programmer does not write text code. Instead, one instantiates objects (oscillators, filters, logical operators, etc.) and links their inputs and outputs in a graphical editor.

In comparison, C++ offers textual, multi-paradigm coding with explicit abstraction mechanisms (classes, templates, etc.), while Arduino C++ is effectively the same paradigm but tailored for embedded hardware with simpler project structure. This is very different from Pure Data (Pd), which uses a graphical dataflow paradigm instead of text. The implication is that C++/Arduino code is written and reasoned about in terms of statements and procedures, whereas Pd is organized visually. In embedded audio contexts this means

C++ can implement any algorithm given enough coding effort, while Pd lets a user assemble signal chains quickly but with less programmatic flexibility.

### *Learning Curve & Accessibility*

Learning curve and accessibility refer to how easy it is for a newcomer to start using the language and tools, including documentation, community support, and conceptual simplicity. When it comes to C++, it has a steep learning curve. It involves detailed understanding of programming concepts (memory management, pointers/references, types, compilation errors, etc.) and a complex syntax. So, while C++ is well documented and widely taught, mastering C++ and debugging embedded issues can take significant time. This high barrier means pure C++ is generally less accessible to those without prior programming experience.

The opposite can be said for Arduino C++, providing a much more accessible environment for beginners. The simplified IDE and API hide overused syntax (e.g. no need to write `main()`) and provide hands-on examples. In Arduino forums, many agree that the design goal was explicitly to make embedded development “accessible to everyone”. Many tutorials and example sketches abound. Thus, someone with little or no coding background can often blink an LED or play a tone with Arduino within hours. However, novice Arduino users may not realize they are still writing C++ code, and more advanced features (e.g. classes or dynamic memory) are typically learned later. In summary, Arduino’s user experience trades off some power for ease of entry: it provides a gentle learning curve.

Pure Data also aims for accessibility, especially for musicians and artists. Because Pd is visual and requires no textual coding, many beginners find it intuitive. The Bela documentation notes Pd “has a gentle learning curve and a block-based approach to coding” [28]. The visual patcher model lets one immediately hear the effect of changes. Of course, Pd has its own concepts (e.g. understanding signal vs. control rate, or the patch-cable metaphor), so there is still learning involved. But for audio prototyping, the barrier is low.

In summary, pure C++ is the hardest to learn, requiring formal programming study. Arduino C++ occupies a middle ground: it is built on C++ but packaged in a beginner-friendly IDE and API, so novices can do basic tasks very quickly while hiding complexity. Pure Data is arguably the most accessible for non-programmers in the audio domain, since it relies on dragging and connecting modules rather than writing code. Each environment thus targets different audiences: expert developers versus hobbyists versus artists, with corresponding learning curves.

### *Performance & Resource Usage*

Performance and resource usage refers to the runtime speed, latency, and the memory/CPU resources required by programs in each environment. For C++, code is compiled to machine instructions, so it generally yields high performance and low overhead. Well-written C++ (especially with optimization) can achieve near-optimal use of the hardware. This is important for real-time audio and DSP, where tight loops and manual memory control can minimize latency.

Trying to compare normal C++ to Arduino C++ is a bit difficult as Arduino programs are also C++ under the hood, so they share the efficiency advantages of compiled code. However, hardware constraints are critical. Typical Arduino-class microcontrollers, such as 8-bit AVR microcontrollers (widely used in embedded systems and are especially known for being the core of many Arduino boards [29]) run at only a few tens of MHz and have very limited memory. This minimalist environment means programs must use resources very carefully

Pure Data, under this review criteria, runs as an interpreter or under a runtime environment (e.g. libpd), so it inherently has more overhead. Every object connection in a Pd patch incurs function-call costs at audio rate. As one practitioner explains, “PureData has some intrinsic performance limitations: all processing blocks are connected at runtime, which adds function-calling overhead and does not allow the compiler to optimize” [30]. In bench terms, the same signal chain in C++ will often use a fraction of the CPU compared to Pd, simply because C++ lets the compiler inline and optimize code across what are separate Pd objects. In addition, Pd’s graphical interface and data structures consume memory: running a patcher requires more RAM and CPU cycles than raw C code. On powerful hosts this is usually acceptable, but on embedded targets one must use trimmed-down builds or compile patches (e.g. with Heavy/HV) for efficiency.

Thus, C++ offers the greatest raw performance and the leanest resource usage (limited only by hardware). Arduino C++ can run reasonably fast but is bounded by the microcontroller’s modest CPU speed and tiny memory. Pure Data trades efficiency for flexibility: it uses more CPU and memory to support dynamic patching. In audio applications this means a pure C++ DSP engine can handle many more simultaneous voices or higher sample rates than a comparable Pd patch on the same hardware.

### Toolchain & Development Workflow

Toolchain and development workflow refers to the tools and processes used to write, build, and deploy projects in each environment (IDEs, compilers/interpreters, debugging, etc.). In the C++ environment, general-purpose toolchains are typically used. Programmers write *.cpp* files in an editor or IDE (Visual Studio, VS Code, etc.) and use compilers like GCC or Clang (on Windows, macOS, Linux) or vendor-specific ARM toolchains. Build systems (Make, CMake, etc.) manage compilation and linking. The workflow is edit, compile/link, then run, often with separate debugging tools. For embedded targets, cross-compilers are used, and code is flashed to the device. This process is powerful but can be complex: setting up includes paths, libraries, and board support requires configuration.

In the Arduino environment, the standard Arduino IDE (or alternatives like PlatformIO) provides a very simple workflow, write your sketch in the editor and click Upload. The IDE automatically invokes the correct compiler and sends the binary to the board. There is no need to manage a separate project or make file. However, this simplicity comes at the cost of advanced features where the Arduino IDE has limited debugging, and the build settings are less configurable (though they can be modified with effort). Despite this, the “plug-and-play” nature of the Arduino toolchain means a user can go from code to running hardware in seconds, which greatly speeds up experimentation.

Finally, Pure Data, as discussed before, uses an interactive graphical environment rather than traditional compilations. Development happens inside the Pd “patcher” GUI where the programmer drags objects (from libraries) into a window and connects them with patch cords. There is effectively no compile step and changes can be made in real time and immediately heard or tested. Deploying Pd patches on embedded hardware (via libpd or exported formats) is usually an additional step where the Pd file is often saved and uploaded to the target device’s runtime. Debugging in Pd is also visual; one inspects signal flow or uses probe objects rather than stepping through code.

In practice, C++ and Arduino involve a code-edit-compile-deploy cycle, while Pd is more like a live performance session. Arduino abstracts much of the boilerplate away, so it feels simpler than a raw C++ toolchain. The implication is that C++ gives full control over the build and deployment (with a steeper setup), Arduino offers a quick “sketch and upload” experience for embedded code, and Pure Data enables on-the-fly patching.

### Summary and Selection

Based on the in-depth analysis across the dimensions of programming paradigm and abstraction, learning curve and accessibility, performance and resource usage, and toolchain and development workflow, standard C++ emerges as the most suitable language for audio development in a synthesizer; particularly in the context of embedded, real-time systems.

While Arduino C++ offers a simplified syntax and lower barrier to entry, it lacks the depth, modularity, and performance control required for advanced audio synthesis. It abstracts away many low-level features, which is helpful for beginners but limiting for precise DSP tasks or managing CPU and memory in tight embedded environments.

Pure Data, on the other hand, is excellent for rapid prototyping and artist-friendly experimentation. Its visual patching model is intuitive and expressive but sacrifices control, performance, and portability. Its higher abstraction level makes it less suitable for production-level firmware or standalone synthesizer hardware where timing and DSP efficiency are critical.

Standard C++, although more demanding in terms of complexity and learning curve, provides the best balance of control, performance, modularity, and ecosystem support. It allows precise implementation of DSP routines, efficient memory and CPU management, and scalable architecture design. Additionally, its integration with professional-grade audio libraries, debugging tools, and real-time scheduling support makes it ideal for building responsive and reliable synth engines.

*Table 4.26: Pros and Cons of Languages Used by Teensy, Daisy Seed, and Bela*

	C++	Arduino C++	Pure Data (Pd)
--	-----	-------------	----------------

Programming Paradigm and Abstraction	Can implement any algorithm given enough coding effort	Can implement any algorithm given enough coding effort with simpler project structure	Graphical dataflow paradigm instead of text; less programmatic flexibility
Learning Curve and Accessibility	Hardest to learn, requiring formal programming study	Built on C++ but packaged in a beginner-friendly IDE and API	Most accessible for non-programmers in the audio domain
Performance and Resource Usage	Generally, yields high performance and low overhead	Reasonably fast but bounded by MCU hardware constraints	Uses more CPU and memory to support dynamic patching
Toolchain and Development Workflow	Involve a code-edit-compile-deploy cycle; powerful but can be complex	Simpler than a raw C++ toolchain at the cost of advanced features	Workflow is fundamentally different; less suited to structured software engineering

#### 4.4.2 Software Audio Library Comparison

In selecting an appropriate development platform for hybrid synthesizer design, the capabilities and structure of the supporting audio libraries play a central role. These libraries not only define the available DSP modules and synthesis techniques, but also shape real-time performance, ease of development, and hardware integration workflow. This section compares the libraries associated with three leading candidates: Daisy Seed (libDaisy and DaisySP), Teensy 4.x (Teensy Audio Library and Audio System Design Tool), and Bela (Bela C++ API, Heavy, and libpd). Each platform is evaluated across three key dimensions: Audio and DSP Capabilities, Runtime Flexibility and Workflow, and Integration and Portability.

## *LibDaisy & DaisySP*

### **Audio and DSP Capabilities**

DaisySP (by ElectroSmith) provides a comprehensive collection of DSP modules covering a broad range of synthesis and processing methods. Although all that is required for NepTune is classic subtractive synthesis oscillators, its library includes noise generators, envelope generators and an extensive set of filters and effects [7]. This breadth of modules means DaisySP has deep coverage of core DSP tools. The Daisy audio pipeline (using libDaisy on Daisy hardware) typically runs at professional sample rates (e.g. 48 kHz by default, optionally 96 kHz) with 24-bit I/O converters, while all signal processing is at 32-bit float for precision [6]. In summary, as a standalone DSP library DaisySP offers a rich set of synthesis and signal processing capabilities, well-suited for building complex audio applications.

In contrast, libDaisy is the hardware abstraction layer for the Daisy platform. It manages audio I/O, ADC/DAC interfaces, MIDI, GPIO, etc., but does not itself add DSP algorithms. Instead, developers use libDaisy in tandem with DaisySP: libDaisy supplies the real-time audio callback and hardware interfacing, while DaisySP contributes the sound synthesis/processing modules. Overall, DaisySP+libDaisy provides very broad audio/DSP capabilities comparable to small synthesis toolkits, with support for modern music DSP techniques out of the box.

### **Runtime Flexibility and Workflow**

Development with DaisySP and libDaisy follows a traditional compiled firmware workflow. Developers typically write C++ code (using the DaisySP library for DSP functions) and compile it to a binary that runs on the Daisy microcontroller. There is no inherent runtime patching or dynamic reconfiguration on the device as the audio graph and DSP code are fixed at compile-time and flash-loaded onto the hardware. Thus, the edit–compile–flash cycle is the main iteration path. This cycle is aided by modern toolchains; for example, Daisy provides support for the Arduino IDE and a GCC-based build system, making it relatively straightforward to build and upload firmware. The test/debug cycle on actual hardware can be in the order of a few seconds for compiling and flashing small projects. However, it is not as instantaneous as a live coding environment since Daisy (an ARM Cortex-M7 MCU) runs without an OS and cannot load new code on the fly.

### **Integration and Portability**

DaisySP and libDaisy were designed with portability and modular integration in mind. DaisySP is an open-source, standalone C++ DSP library. It can be embedded in many contexts beyond the Daisy hardware: for example, it has been used in VST/AU plugins (via JUCE), mobile app projects, and even in software modular synthesizers like VCV Rack [7]. This cross-platform support is possible because DaisySP is self-contained and does not assume a specific hardware layer

In contrast, libDaisy is specific to the Daisy Seed and its derivatives as it encapsulates drivers for the STM32 MCU, audio codec, and peripherals. Theoretically, it can be adapted to other STM32 boards with the same chipset, but it is essentially tied to Daisy's hardware definition. Although this is the case, the Daisy toolchain also supports standard ARM Cortex-M development tools (make/gcc or Arduino IDE with Teensyduino-like integration), meaning it fits into typical embedded workflows without proprietary dependencies.

### *Teensy Audio Library*

#### **Audio and DSP Capabilities**

The Teensy Audio Library (from PJRC) is a C++ framework providing a collection of audio building blocks optimized for the PJRC Teensy microcontroller boards. It implements a streaming audio engine at fixed “CD quality” resolution (16-bit samples at 44.1 kHz) [31]. Despite the fixed-point audio format, the library features a fairly rich set of modules for synthesis, playback, and processing. Key capabilities include polyphonic sample playback, audio recording to memory or card, and basic waveform generation. However, compared to DaisySP or PC-based frameworks, the scope of provided DSP modules is somewhat limited. The library focuses on fundamentals (oscillators, filters, delays) and omits some advanced or niche algorithms. In summary, the Teensy Audio Library offers a solid selection of basic DSP and synthesis modules sufficient for synthesizers or effects in an embedded context.

#### **Runtime Flexibility and Workflow**

The Teensy Audio Library uses a compile-time, graph-based workflow within the Arduino/Teensyduino environment. Users define audio signal chains by instantiating audio objects in C++ and connecting them with `AudioConnection` objects. To simplify this, the PJRC Audio System Design Tool provides a visual drag-and-drop interface that generates the C++ code automatically.

Although the tool resembles environments like Max/MSP or Pure Data, the resulting code must still be compiled and uploaded to the Teensy board. The audio graph is static—repatching or structural changes require recompilation and re-uploading via USB. However, real-time parameter changes (e.g., filter cutoff, oscillator frequency) can be handled efficiently in code while the firmware runs.

Teensy’s audio engine operates on synchronized audio blocks (typically 128 samples) and uses high-priority interrupts for real-time scheduling. While it supports integration with tools like Faust for advanced DSP development, this too follows an offline compilation model. Overall, Teensy offers a fast compile-and-test cycle and a beginner-friendly workflow, but it does not support live editing or dynamic patching on the device.

#### **Integration and Portability**

The Teensy Audio Library is closely integrated with the Teensy hardware and PJRC’s software stack, which influences its portability. It is essentially an Arduino-style library

tailored to the Teensy 3.x and 4.x series microcontrollers (ARM Cortex-M4/M7 MCUs) and the PJRC Audio Shield or equivalent audio ADC/DAC hardware. The library's code makes use of the Teensy's specific I<sup>2</sup>S audio interface and DMA mechanisms, and it assumes the presence of certain hardware features (e.g., 16-bit audio codec, specific MCU timers). As a result, using the library on non-Teensy platforms is not possible; it is not a generic cross-platform DSP library.

Within the Arduino environment, the library is modular and compatible with other Arduino libraries (e.g., for MIDI or sensors). It supports various audio I/O options on Teensy, including I<sup>2</sup>S codecs, on-chip ADC/DAC, S/PDIF, TDM, and USB audio streaming. The ability to act as a USB audio interface allows Teensy-based synths or effects to integrate directly with a DAW without extra hardware.

Written entirely in C++, the library must be used within Arduino IDE or compatible toolchains. It does not support other language or environments. While its structure helps developers build complex DSP graphs and learn audio design principles, the software itself is not portable beyond PJRC hardware. In short, the Teensy Audio Library is well-optimized for its native platform but lacks flexibility for use elsewhere.

### Bela C++ API with libpd

#### **Audio and DSP Capabilities**

The Bela platform (based on BeagleBone ARM computers) approaches audio differently, using a full Linux-based environment with ultra-low latency scheduling. In the context of Bela's C++ API combined with libpd (Pure Data), the audio/DSP capabilities are essentially those provided by Pure Data's DSP engine, augmented by any C++ code the developer writes. When running a Pd patch via libpd on Bela, one has access to virtually all of Pd's *vanilla* objects. In other words, the breadth of DSP methods is limited mainly by what can be expressed in Pure Data.

Bela enhances Pd's performance by reducing the standard DSP block size from 64 to as low as 8 samples, enabling ultra-low latency (as little as 0.18 ms at 44.1 kHz). The system uses 32-bit floating-point audio internally and supports 16- or 24-bit audio I/O depending on the hardware configuration.

In addition to Pd, developers can write custom DSP in C++ using Bela's API, allowing integration of advanced algorithms or third-party libraries. This hybrid approach gives Bela a high level of DSP flexibility, combining the modularity of visual patching with the power and precision of compiled C++ code.

#### **Runtime Flexibility and Workflow**

Bela's standout strength is its exceptional runtime flexibility. Using Pure Data (Pd) via libpd, Bela allows developers to build, edit, and reload audio patches in near real-time without recompiling C++ code. Pd patches are designed visually on a PC, then uploaded to Bela through its browser-based IDE even while audio is running. This enables rapid

iteration: developers can make DSP changes, save the patch, and immediately hear the results on hardware.

Though Bela runs Pd headlessly, it preserves Pd's interactive workflow. The IDE supports drag-and-drop patch updates and automatic relaunching of the patch, making the test/debug cycle fast and fluid. This live coding-like experience is not possible on Daisy or Teensy, where recompiling firmware is required for most changes.

For advanced workflows, Bela allows hybrid development: Pd handles high-level logic or sequencing, while C++ routines (e.g., the render() function) can perform low-level DSP. Bela supports real-time communication between Pd and C++ for both audio and control messages, offering tight integration. Developers can also programmatically load new Pd patches at runtime.

While Pd via libpd is less CPU-efficient than compiled C++, Bela's fast processor mitigates this. For performance-critical patches, Bela supports Heavy, a compiler that converts Pd patches into optimized C code—trading live flexibility for speed.

In summary, Bela enables an interactive and expressive development process, ideal for prototyping and live experimentation. It offers a workflow far more dynamic than traditional microcontroller systems like Daisy or Teensy.

## Integration and Portability

Bela offers strong integration due to its Linux-based architecture and flexible software stack. Its C++ API and real-time kernel (via Xenomai) allow developers to combine real-time audio with additional capabilities such as network communication, file I/O, high-level scripting (e.g., Python), and sensor interfacing—all in the same environment. The Pd engine, accessed via libpd, runs inside the audio thread and acts as a modular DSP unit within this broader system.

This setup enables hybrid workflows; developers can write C++ routines for low-level I/O or control, while using Pure Data for DSP and musical logic. Real-time communication between Pd and C++ is fully supported, allowing audio and control messages to flow both ways during execution. Bela provides examples where, for instance, sensor data is collected in C++ and passed to a running Pd patch for sound generation.

In terms of portability, Pd patches are largely platform independent. So long as only standard Pd objects are used, a patch created for Bela can usually run unchanged on desktops, Raspberry Pi, or mobile platforms. Libpd itself is also portable, enabling Pd patches to be embedded in other systems, such as games or mobile apps.

However, the Bela C++ environment is less portable. Its reliance on the BeagleBone's real-time extensions, PRU co-processors, and specific I/O hardware makes direct transfer to other platforms non-trivial. While some components (like DSP algorithms or Pd integration) can be adapted to other Linux systems, full Bela functionality depends on its hardware and kernel configuration. Experimental efforts to run Bela software on platforms like Raspberry Pi exist but are unofficial.

Unlike more monolithic environments (e.g., Teensy), Bela encourages modularity by cleanly separating audio logic (Pd) from system-level code (C++ or shell processes). It can also interconnect with other systems (e.g., Max/MSP, SuperCollider) over a network. In summary, Bela excels at integrating diverse software elements around a central, real-time audio engine, and while its full stack is hardware-bound, the Pd-based DSP logic is widely portable and reusable beyond the Bela ecosystem.

### Summary and Selection

*Table 4.27: Pros and Cons of Various Libraries*

	LibDaisy & DaisySP	Teensy Audio Library	Bela C++ API with libpd
Audio and DSP Capabilities	A rich and curated set of modules; floating-point precision and optimized for STM32	Moderate but accessible DSP toolkit; fixed-point arithmetic; static audio graphs and ease of use	Derives its audio capabilities from Pure Data; DSP highly flexible and comprehensive in coverage
Runtime Flexibility and Workflow	Conventional embedded development cycle; deep control over the firmware and timing	Semi-flexible workflow; convenient for visual learners and suitable for fast prototyping	Excels in runtime flexibility; live coding-like experience; makes the test/debug cycle fast and fluid
Integration and Portability	Written in portable C++; reused in desktop or plugin projects; can be written in Arduino (DaisyDuino)	Tightly coupled to PJRC hardware and Arduino-style C++; limiting reuse outside that ecosystem	Pd patches and libpd engine are highly portable across platforms; stands out in system integration
STM32 Compatibility	High STM32 compatibility; can theoretically be adapted to other STM32 boards	Targeting ARM Cortex-M processors; does not support STM32 natively	Running on a BeagleBone Black with a Linux OS; not compatible with STM32 at all

While all three library sets serve specific use cases, DaisySP + libDaisy is clearly the best-suited for developing a hybrid synthesizer on an STM32 platform. It combines a rich set of music-focused DSP modules, reliable real-time performance, seamless STM32 integration, and the flexibility to reuse code in future projects. Bela + libpd offers unmatched runtime interactivity and system-level flexibility but is tied to Linux and not usable on STM32 microcontrollers. Teensy Audio Library is a solid tool for fixed-point

prototyping and education, but lacks the precision, modularity, and platform compatibility needed for advanced STM32-based synth development. Therefore, DaisySP + libDaisy provides the most capable and efficient foundation for embedded synthesis on STM32.

#### 4.4.3 Development Board Final Selection

Given the previous selections of the programming language and the audio library stack, standard C++ and DaisySP + libDaisy, the most appropriate development board for NepTune is the Electro-Smith Daisy Seed. This is because the Daisy Seed is designed to run standard C++, not a simplified Arduino dialect or interpreted language. It is also tightly coupled with the DaisySP and libDaisy libraries, which are written in C++ and optimized for real-time audio on STM32H750, the MCU previously selected.

*Table 4.28: Pros and Cons of Various Platforms*

	Daisy Seed	Teensy 4.x	Bela
Primary Language	C++	Arduino C++	C++, Pure Data
Audio Library	LibDaisy + DaisySP	Teensy Audio Library	Bela C++ API, libpd, Heavy
Main IDE	VS Code / Arduino	Arduino IDE / PlatformIO	Wed IDE / SSH (Linux-based)
Visual Patch Support	No (code only)	Yes (Audio System Design Tool)	Yes (Pure Data)
DSP Complexity	High, modular, efficient	Moderate, beginner-friendly	Very high, ultra-low latency
Best for	Embedded DSP synths	Quick prototyping, MIDI synths	Responsive instruments, audio experiments
STM32 Compatibility	Yes	No	No

DaisySP offers deep DSP coverage without needing external extensions or visual patching. Its codebase is readable, modular, and encourages reuse, meaning one can implement oscillators, filters, and modulation systems using reliable and optimized components. Additionally, libDaisy offers structured access to I/O (buttons, knobs, displays, gates, CV) directly through C++ classes, making hardware integration straightforward and performant.

Another reason why the Daisy Seed was chosen is how tailored it is for embedded audio. Unlike Teensy or Bela, the Daisy Seed was purpose-built for audio synthesis. Its entire ecosystem, from the firmware architecture to its DSP library design, is oriented toward

modular synthesizer development with special attention to low latency, signal quality, and audio-rate modulation. The MCU's floating-point unit and dual ADCs/DACs make it ideal for real-time sound engines, control voltage (CV) interfacing, and responsive synthesis control.

Furthermore, the Daisy Seed's firmware toolchain is based on STM32CubeMX and ARM GCC, providing professional-level control over memory layout, peripheral initialization, and scheduling. Its firmware can be extended using CMSIS, STM32 HAL/LL, or even RTOS-level constructs, which is advantageous for building complex hybrid synths that may require multitasking, MIDI parsing, storage, etc.

#### 4.1.4 Chassis Design

Numerous images of what the synthesizer would end up looking like had come up during planning of the device, however, serious consideration into the housing of our components is important to the viability of NepTune as an industry product. Not only will it determine what is possible inside the device, but it is heavily reliant on what user capabilities the team manages to implement. Each knob and button on the device that the user interacts with needs the matching hardware underneath the front cover, and with modularity in mind, the chassis will have to also be modular in design. Additionally, this is one of the only designs that will require the unique consideration of aesthetics, along with the print left on PCBs.

Originally, the members that had considered the physical image of the device were hoping to have a hybrid wood and metal design that could be separated. For NepTune to include wood, attention would have to be placed upon its manufacturing as wood needs to be cured and often polished/sealed for it to be included in audio equipment. Because NepTune is portable, exposure to water is not a possibility that the team can ignore. Sealing can be achieved via numerous strategies which will be covered now:

##### 4.1.4.1 Water-Resistant Design

###### Water Resistant Materials - Metal

The simplest solution is to make sure that the materials used for housing the hardware of NepTune is as resistant as possible. This property has been integrated into numerous materials and designs so there is quite a bit to base NepTune's design on. Firstly, the one member that suggested wood and metal pointed out that a high durability solution was aluminum created via Computer-Numerical-Control (CNC) manufacturing. This would also require protective coating which would allow a single large metal plate to be bent so that the entire front, back, and keyboard would be protected with a stronger material. One major downside to this design however is the cost and complexity of creating a metal shape that would fit NepTune's requirements. Added to this downside is the lack of adaptability, because the process of acquiring a custom metal piece either through manual creation from the team's access to school labs which do require permission, or through ordering it from a CNC-based metal manufacturer, would be a lengthy and difficult to change process.

###### Water Resistant Materials - Wood

A more adaptable design would be completely relying on wood for the chassis; this includes a more complicated sealing process on the contrary. For electronics, it is best to use a number of strategies because of the heat generated on the inside from IC's and power elements. When deliberating the fully wood design, even small details such as the wood type are necessary for safe construction. Softer woods like pine and cedar woods require extreme sealing procedures because of their water absorbing properties. Harder woods like walnut or oak are better at resisting moisture. Unexpectedly, the additional fear of wood warping is presented, if there is any moisture build up on the inside or outside of the device without proper sealing and coating, then physical deformities will affect NepTune, very likely causing complete device failure. While the cost of wood is cheaper than that of metal, the additional coating techniques could prove to result in higher net expenses unexpectedly. This is a very meticulous decision the design team has to come to agreement on.

### Water Resistant Materials - Plastic

The other third realistic option for the chassis' make-up would be a 3-D printable plastic. The type of plastic that could be used is a large consideration to breakdown but there are numerous types that embody water-resistant properties so for now, only the general tradeoffs compared to the other two materials will be listed. Plastic should prove to be the lightest material that can be used which helps the weight goal of NepTune with the requirement of less than 8lbs. This material could however also prove to be the weakest of what can be used so major attention needs to be given to the structural integrity of the device. The cost of plastic is also much less than the other two materials so as long as the functionality and complexity of the design is acceptable, cost will likely be the deciding factor. Because plastic is printable as well, the complexity of design will be decreased for the shape, the team will be able to add minute details and the difficulty of creating certain structural shapes should be significantly reduced.

*Table 4.29: Comparison of structural materials*

	Aluminum/Other Metal	Wood	Hybrid (Wood + Metal)	Plastic/Resin
Cost	High	Low-High	High	Low-Medium
Availability	Low	High	Low	Medium
Water-Resistability	High	Low	High	High
Complexity of Design	Low	High	Medium	Medium
Adaptability	Low	Medium	Medium	High
Structural Strength	High	Medium	High	Medium
Weight	Heavy	Average	Average — Heavy	Light - Medium

### Final Decision

The final decision the design team made was to proceed with a plastic chassis, this does require some sealing strategies but is the cheapest, most accessible and the most familiar material to the team. This also benefits the likelihood that the team does not need to

gressively change designs to accommodate the weight of the chassis in order to complete the engineering specifications. Further research will have to be done to determine the type of plastic that is used, another comparison will be made, and the individual sealing techniques will be explored as well.

## **Plastic Materials**

Different plastics will be studied to observe their individual properties including price, availability, water-resistance, weight and strength. Starting with PLA (Polylactic acid) as the base, this material is not resistant to water so explanation of integration and reasons to avoid will be the main goal to then compare the other materials too.

### **Polylactic Acid (PLA)**

This is a thermoplastic meaning that it is malleable while hot and solidifies upon cooling. Derived mainly from biodegradable resources like plant matter, it is remarkable for its ease with printing but results in a brittle and hygroscopic structure. Both properties subtract from the feasibility of including it as a brittle design would result in the movement of the device to be a risk in of itself and with the material being hygroscopic, it will slowly absorb moisture and would require additional sealing techniques.

### **Acrylonitrile Butadiene Styrene (ABS)**

Another thermoplastic however this time derived from petroleum, is far more prevalent in industrial uses ideally widening its usefulness for NepTune's design. It has increased strength which is necessary for on-the-go mishaps. While it is not hygroscopic, the water-resistance only reaches as far as there are no seams so additionally sealing is still required. In order to print using this material as well, it is required to have a more advanced printer, one that can be enclosed and performance-wise can raise the printing bed to 100 degrees Celsius. Additionally, for the sake of user interaction and aesthetics, sanding and acetone smoothing might be required.

### **Polyethylene Terephthalate Glycol-Modified (PETG)**

This material has good water-resistance, wouldn't require any more special sealing than ABS. It also maintains strength along with more flexible properties meaning that it would be more resistant to cracking in case a user drops NepTune and the vibrations shakes the chassis. PETG is less prone to warping, requiring less heat and therefore, not as advanced of a printer is required for creation. The major downside to PETG is that it is not as tough as ABS, leading to increased surface-level scratches. Having the best consumer grade waterproofing without sealing, PETG is certainly a candidate for NepTune's final design.

### **Polycarbonate (PC)**

This thermoplastic is unique in its incredible resilience, often used for impact resistance for engineering applications. Polycarbonate plastic is extremely strong and would offer the best protection for NepTune's structure. Additional to physical endurance, PC offers the best waterproofing and thermal protection available. All of these characteristics would make NepTune incredibly safe as a product. However, the drawback is difficult to overcome when it comes to this project and product. Access to an advanced,>290°C capable printer is required to print with Polycarbonate making access to this technology

near impossible for the resources available to the design team. On top of the limited access, PC is 2 – 3x as expensive, which even in the limited amount that NepTune will be using, still adds up to a gruesomely large amount, contributing to the possibility of overextending past the budget. Because NepTune has no moving parts and in every realistic case will not be subject to extreme physical shocks, this premium part may not be justified enough to include over a less-performing, affordable option.

Now the individual plastic material types will be compared to one another for the final conclusion of which type of thermoplastic will best suit NepTune's design;

*Table 4.30: Comparison of thermoplastic materials*

	PLA	ABS	PETG	PC
Cost (per kg)	\$15-\$25	\$20-\$30	\$20-\$35	\$40-\$70
Density (g/cm <sup>3</sup> )	1.24	1.05	1.27	1.21
Strength	Brittle	Durable	Durable + Flexible	Impact Resistant (Best)
Printer Requirements	Basic	>100°C	Basic	>290°C
Water Resistance	Low	Medium	Good	Best

### Final Choice: PETG thermoplastic.

PETG offers accommodating printer and sealing requirements that should streamline the process of working with the material for NepTune's design. Additionally, the cost of the product is significantly less than the stronger material PC, which we have no personal equipment that could produce PC designs either way and would have to outsource. The weight of PETG is not the best in class but should not significantly impact the weight of the final product any more than PC or PLA which are a few mg/cm<sup>3</sup> lighter than PETG. ABS's lower weight would certainly give the team more weight to work with for other design aspects but the drawbacks of increased design requirements for sealing and possible error with the higher temperature synthesis encouraged the team to stick with PETG.

### Sealing Techniques

To properly secure the NepTune synthesizer with a water-resistant chassis, several sealing steps are required as PETG is not sufficient against the environments that NepTune is encouraged to be used in (mobile). This section will cover the options that are available to the team to consider and at the end a decision will be made contemplating the benefits and drawbacks of implementing each option. Ideally, the NepTune synthesizer will be practically resistant to any spills, rain or moisture that it comes across. Due to the nature of electronics and the resources available to the team, NepTune will not be entirely waterproof nor in any real-world scenario would it be expected to be. This precautionary

design step is very important for expensive electronics because of the catastrophic damage that water coming into contact with various elements of the synthesizer has to itself and the user. Real world accidents should be accounted for in any design worth its cost.

### Silicone Gaskets

One simple way to significantly reduce the likelihood of water from a spill or splash from entering through connection seams is to add a gasket within the chassis at each of these points. Along with the seams that connect the chassis segments to each other, gaskets should be implemented along the ridges of each knob so water cannot enter through the top of the board. Adding this element requires that the team acquires a few meters of silicon cord a few millimeters in diameter and engrave the chassis design so that crevices are available for insertion. This requires very little manual work, is relatively cheap, and is easy to acquire. The only careful procedure has to be the design of the grooves, a size too large will not hold the watertight characteristic that the design is aiming for and a size too small will create gaps in the creases of joints, further exposing NepTune to outside liquids.

### Epoxy Resin Coating

Coating NepTune in a resin coating is a more involved process that could create a wonderful seal and look to the device if properly applied. Initially, the plastic must be sanded, ~220 grit so that the surface is smooth to the touch, this reduces the chances of irregularities during curing and increases the appearance of the board. Then using epoxy kits, lightly covering the outside of the device so that seams are protected with the layer of resin. Theoretically this should keep water from entering previous access points thanks to the additional layer of protectant, diverting the water away from the inside. This is a relatively manual process that can be incorrectly performed, so additional precautions like printing multiple chassis' might be required for a safe design process. Unfortunately, the resin can add weight to its surface which could contribute to exceeding the limit that was set, and with the process being difficult to "undo", committing to using epoxy could set back the development of NepTune intensively. The cost of adding epoxy can also quickly add up due to the tools and amount required to consistently coat the device.

### Conformal Coating

Protection of the PCBs located within NepTune's chassis can help prolong the life expectancy of the device by preventing moisture and other contaminants from entering the chassis and effecting the efficacy of any electronics inside. While sealing the device conventionally can sufficiently keep humidity away, anytime with the electronics exposed, possibly for repairs or to clean the inside of the device, introduces the possibility of moisture entering and polluting the PCBs. Coating using a brush or spray on requires careful application to avoid ruining certain parts that might be especially susceptible to any kind of addition. Moving/exposed contact parts like switches and connectors are recommended to be covered during the application process. Curing and application are faster than epoxy and should serve as a deterrent from risks of humidity within the case. One unique challenge this leaves the design team with is that the circuitry is usually un-

alterable after application, testing and proof will have to be conducted before the final decision whether to apply is made.

### **Paint-On Sealant (Polyurethane)**

Another, lighter strategy would be to use paint on sealants rather than an epoxy resin coating as the application time is significantly reduced in complexity and time. Heavier sanding will be required around ~400 grit which can alter the look to be sleeker but could also expose the design to structural concerns. Uniquely these paints can come in custom colors allowing for NepTune to be themed easily should that be the route in which the design team decides to improve the aesthetics. The major downside to using this process is that, structurally, it is weaker and prone to leakage after extensive use, possibly shortening the average lifespan of NepTune.

Now will be the comparison of these strategies, unlike other technologies that have been compared, this time multiple options will likely be selected as each one creates another barrier between component failure and a safe operating device.

*Table 4.31: Comparison of Sealing Techniques*

	Gaskets	Epoxy	Conformal Coating	Paint Sealants
Effectiveness	Good	Best	Good	Good
Cost	\$5-\$10	\$15-\$20 (Multiple needed x2-3)	\$5-\$10 (Multiple needed x2)	\$5-\$10 (Multiple needed x2-3)
Complexity	Medium	Difficult	Medium	Medium
Installation Time	Fast (~ 1 Hour)	Long (~ 2 Hours Manual + ~5 Hours Curing)	Fast (~ 30 Minutes + 1 Hour curing)	Very Long (~2 Hours Manual + ~24 Hours curing)
Additional Benefits	Additional physical layer	Glass-like finish, more waterproof/resistant	Extended circuitry life	Matte finish, color choice

### **Final Choices:**

The gaskets were chosen to be included due to the sheer ease of installing them and the low cost, additional attention to detail will be required when creating the 3-D model of NepTune in order to install them but should prove a useful tool for avoiding floods inside the device.

Conformal coating was chosen as an expectation that should be included no matter how difficult the process is as any trapped water, heat or environment that might encourage humidity to build up within NepTune is extremely threatening to the prolonged effectiveness of the ICs and other electronics. Great care will have to be put in when applying this sealing strategy so as not to accidentally apply liquid where it can create more problems.

Paint Sealant is the current choice that the design team has chosen to pursue, mostly because of the high cost and difficult application process that comes with using resin, while long, the application of paint sealant is simple in practice, good planning can avoid this being a problem.

## Chapter 5 – Standards and Design Constraints

This chapter deals with the various standards and constraints this project must adhere to. These standards provide a framework for the team to follow, and are necessary for demonstrating professionalism. As aspiring engineers, understanding, applying, and incorporating these standards is very important. It offers insight into what is typically expected in industry and helps ensure that the project will be taken seriously by peers, faculty, and future professionals who may review this work.

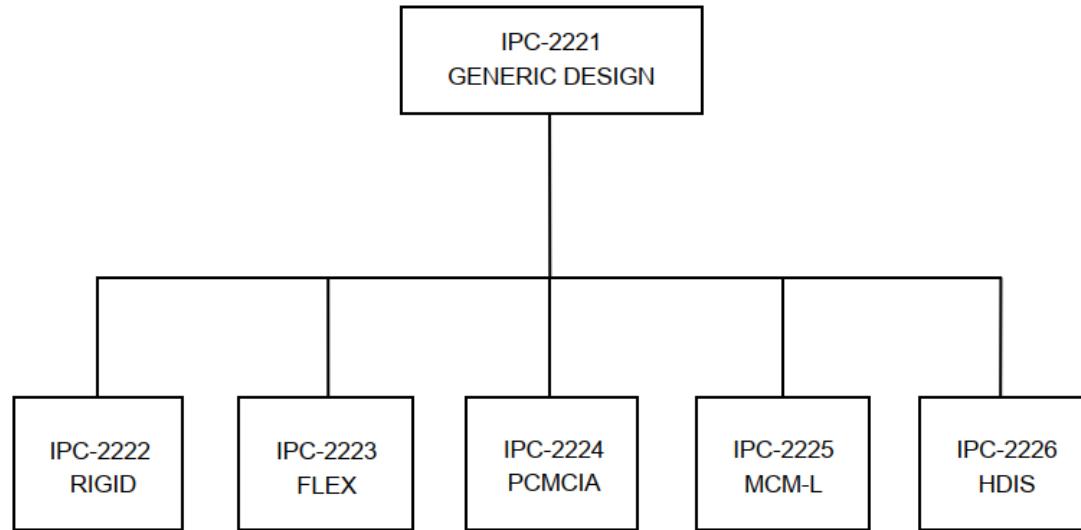
In addition, identifying the constraints placed upon the project is essential before entering into the design phase. Recognizing these limitations early on will minimize the amount of unforeseen issues that lead to wasted time and effort. The chapter is split into two sections: the first section covers important standards related to the project, while the second section deals with constraints.

### 5.1 Standards

Standards are types of guidelines or rules that designers and manufacturers agree to follow to maintain seamless collaboration and product compatibility. An example would be the standard thickness of a 1oz. copper pour on a printed circuit board. Most designers would expect a 1oz. copper pour to be about 35um thick. For a 1oz. copper pour spread across a square foot of board, the designer's expectation of 35um would be correct. However, if manufacturers decided to use a different size area, the thickness of the pour could vary greatly. Without agreeing on this standard, designers could not assume all manufacturers would provide the same thickness. While it would not be impossible to find out what standard each manufacturer uses, it would make the process of choosing a manufacturer a bit more difficult, resulting in a slower, more cumbersome workflow. As a result, even though standards may not be required by law, it is in everyone's best interest to agree on certain things. The following section covers some related standards that will be incorporated into NepTune. These standards include elements such as design guidelines and communication protocols for data transfer.

### 5.1.1 IPC Standard: IPC-2221A

The IPC standards are a set of standards produced by the Global Electronic Association (GEA), formerly known as the *Institute of Printed Circuits* (IPC). These standards are recognized worldwide and cover a variety of topics within electronics such as PCB layout and integrated circuit manufacturing. The focus of this section will be on IPC-2221A and its accompanying standard, IPC-2223, relating to the rigid PCB design and layout of NepTune.



**Figure 5.1:** Topics covered under IPC-2221A

The IPC-2221A is a generic standard establishing a broad set of guidelines concerning printed circuit board design with organic materials (such as the FR4 dielectric used in circuit boards). This standard establishes generic guidelines that affect all types of circuit board design. Within it are more specialized standards that focus on a specific type of technology. As depicted in Figure 5.1, it covers a range of printed circuit board (PCB) technologies such as rigid PCBs, Flex PCBs, and multi-chip module-laminated PCBs. Since NepTune will be utilizing rigid PCBs, the focus of this discussion will be on how IPC-2221A and IPC-2223 may be utilized during the design of this project's various circuit boards.

The IPC-2221A raises some considerations to make before laying out a PCB. The considerations that apply to NepTune are listed below:

**Board Density.** How many components will be on the board? What is the pitch spacing between each of the pins? How many traces will be required? No matter how small the footprint of a component is, the number of traces remains the same. As a result, there is a limitation to the amount of traces that can fit into a certain area on a board. Thus, there is potential for the limiting factor in board size to be the number of traces.

**Critical Layout Areas.** Which are critical layout areas? Which parts of the PCB are sensitive to noise and will require shielding or differential signaling? To streamline the process, it is important to note down these specific areas first. How many parts will be on the board? A full parts list is required in order to estimate the amount of space necessary to realize the board's construction.

**Testing Points.** Testing points are vital during the prototyping and debugging process. Consequently, the standard discusses the types of testing points that can be used to test the functionality of the circuit. Since NepTune is not going to be mass-produced, it will be sufficient to place small testing points with vias or pads to streamline debugging and testing during the prototype phase.

**Material Type.** The standard also covers the types of materials that can be chosen for the PCB. Important considerations include fire resistance, thermal stability, structural strength, among other electrical properties. For NepTune, the strength of the PCB is important to ensure product longevity.

**Board Layers.** To minimize the amount of bow and twist resulting from mechanical stress, proper care must be taken to balance the circuitry. This includes even copper pours throughout the board and across cross-sectional layers. The core and dielectric thickness must be of symmetrical size to minimize bow and twist. If necessary, fasteners may also be used to reduce the amount of warping the board may experience over time.

**Mechanical Hardware.** The printed circuit board must be designed in such a way that any mechanical or mounting hardware can be easily mounted onto the board. Additionally, proper clearance must be given to alleviate the risk of circuit damage when installing such hardware. Additionally, any parts weighing 5.0 grams or more must be supported by spacers or other similar hardware. This will reduce the chances of solder cracking.

**Power Distribution.** Copper pours can be used to create power and ground planes to aid in the distribution of power. In addition, any power input should use decoupling capacitors to reduce noise introduced into logic. If using power conductors, power traces should be run as close as possible to ground traces. In addition, both power and ground traces must be made as wide as possible.

**Electrical Clearance.** Spacing between traces should be maximized wherever possible. If there are power traces carrying separate voltages, they must be identifiable on both the silkscreen and on the master drawing. In addition, partitioning the board into areas based on function is important for maintaining an organized board. Grouping related components together, such as power elements, can streamline the prototyping process. Additionally, it may also be necessary to create separations between digital and analog circuitry for better electrical performance. This is especially true for audio circuits and proper care must be taken to isolate analog components from noise caused by digital switching.

**Thermal Management.** Thermal management is especially important and must be planned out before routing. It is important to know the operating temperature of each integrated circuit so that an appropriate cooling method can be selected. For designs meant

to be in a rugged environment, such as a portable synthesizer, indirect cooling may be considered. Such a design would utilize heatsinks and fans if necessary. The preferred method for *NepTune* would be to operate off heatsinks alone. However, voltage regulators can generate quite amount of heat so the possibility of a fan is not completely out of the question.

### 5.1.2 UL Standard: UL-60065

UL, Underwriters Laboratories, are a global safety organization that develops standards for system safety. They provide something akin to instruction manuals which provide safety guidelines for designers, and manufacturers. The standard to be discussed, UL-60065, is the UL standard for audio and video safety requirements. This standard is primarily discussing objects intended for household and similar use but also covers professional objects that are used in public.

There are a multitude of conditions that this standard covers. The general requirements of this standards are as follows:

**Hazardous Currents Passing through the human body:** This test is inclusive of any type of objects that utilize power supply, and the range of the rated voltage is tested from maximum to minimum. Moreover, any portion of the object that utilizes a change inn voltage is also tested. Therefore, *NepTune* containing multiple potentiometer the effects the voltage passing through would undergo testing for all knobs as well as its plug in supply. This test is conducted under normal operating conditions, and preferred to be tested on the apparatus the user would receive but can also be used on individual components utilized by the user.

This test must be carried out on the material, subassembly, and the synthesizer (in this case) as a whole when no power is present. Then the tests are conducted with power on under normal operations, abnormal operating conditions, and if the object is able to operate acceptably with common destruction that is likely to occur to the object during usage. The test is conducted through a range of voltages between 0.9-1.1x rated supply voltage.

A fault test is also applied to object which includes an examination of all circuit diagrams which shows potential areas subject to hazard creation. These are then applied in sequence and tested multiple times to ensure similar results are always obtained.

Another requirement regarding electric shock is through constructional requirements, which state that the apparatus shall be designed and constructed so that the operations by hand such as changing voltage, replacing indicator lights do not involve a risk of electric shock, and the apparatus shall also be constructed that no electric shock may occur from parts rendered accessible from removing a cover by hand.

**Hazardous radiation** is designed to examine any potential source of ionizing radiation, and that protection is available under normal operating conditions, and under fault conditions. This would not apply to *NepTune* as it does not contain any source of ionization.

**Mechanical instability** is examined via multiple tests that test the durability of the device. These tests are made up of the bump test where the device is allowed to fall from a certain height multiple times, vibration test, impact test, and drop test, all of these are testing that no parts became loose due to the conditions of each test.

**Resistance to Fire** this test applies to all parts of the device, and is aimed so that a start and spread of fire is prevented and shall not spread beyond the origin of the object. This is done through utilizing basic design practices, and low flammability material. These are considered fulfilled if material that is chosen falls into the categories provided by the standards.

**Excessive Temperatures** this part of the standard seeks to check that no part of the instrument raises to excessive temperatures under normal operating conditions after a steady state is attained, the standard defines steady state to be attained after 4 hours of operation. Temperature rising is measured through the all accessible parts of an apparatus, and any protective device or component of a protective circuit in regards to temperature. All these values are then compared to a predetermined expectation table relating to temperature as follows:

Parts of the apparatus	Normal operating conditions K	Fault conditions K
<b>a) ACCESSIBLE parts</b>		
Knobs, handles, etc. if		
– metallic	30	65
– non-metallic <sup>c</sup>	50	65
Enclosures if		
– metallic <sup>b</sup>	40	65
– non-metallic <sup>b + c</sup>	60	65
<b>b) Parts providing electrical insulation <sup>d</sup></b>		
Supply cords and wiring insulation with		
– polyvinyl chloride or synthetic rubber	60	100
– not under mechanical stress	45	100
<b>c) Parts acting as a support or a mechanical barrier including the inside of enclosures <sup>d</sup></b>		
– natural rubber	45	100
Other insulations of:		
– thermoplastic materials <sup>e</sup>	f)	f)
– non-impregnated paper	55	70
– non-impregnated cardboard	60	80
– impregnated cotton, silk, paper and textile	70	90
– laminates based on cellulose or textile, bonded with		
– phenol-formaldehyde, melamine-formaldehyde, phenol-furfural or polyester	85	110
– epoxy	120	150
– mouldings of		
– phenol-formaldehyde or phenol-furfural, melamine and melamine phenolic compounds with		
– cellulose fillers	100	130
– mineral fillers	110	150
– thermosetting polyester with mineral fillers	95	150
– alkyd with mineral fillers	95	150
– composite materials of		
– polyester with glass-fibre reinforcement	95	150
– epoxy with glass-fibre reinforcement	100	150
– silicone rubber	145	190
<b>d) Coil winding wires and insulation systems <sup>d + g</sup></b>		
– insulated with		
– non-impregnated silk-, cotton, etc: <u>Class 105 (A) insulation system</u>	55 75	75 110
– impregnated silk-, cotton, etc: <u>Class 120 (E) insulation system</u>	70 90	100 125
– oleoresinous materials <u>Class 130 (B) insulation system</u>	70 95	135 130
– polyvinyl-formaldehyde or polyurethane resins <u>Class 155 (F) insulation system</u>	85 115	150
– polyester-resins <u>Class 180 (H) insulation system</u>	120 135	155 170
– polyesterimide-resins	145	180
<b>e) Other parts</b>		
These temperature rises apply to parts not covered by items a), b), c) and d):		
Parts of wood and WOOD-BASED MATERIAL	60	140
Lithium batteries	40	50
Resistors and parts of metal, glass, ceramic, etc.	No limit	No limit
All other parts	200	300
Conditions applicable to table 3		

**Figure 5.2: UL standards for individual temperature of parts.**

The standards, however, allow for deviation based on climates, and parts not touched during use. The above table was used as an example, however, there exists multiple predetermined values for many of the discussed standards thus far.

The UL standard goes into more details concerning things such as wiring, winding, and each individual section of the audio device which can be viewed in more detail if the standards are purchase. UL, however, is not a mandatory requirement for production however many companies will choose to have their products UL certified as it is a method of insurance when it comes to expectations, that is if you are certified your products receive some sort of guarding when it comes to malfunctions.

### 5.1.3 MIDI Standards and Implementation

To ensure robust interoperability and modern usability, NepTune is built around a carefully selected subset of foundational MIDI standards. These standards define how musical data is formatted, transmitted, and interpreted between electronic instruments. MIDI is not a single protocol, but rather a layered ecosystem of specifications that govern everything from message structure to physical connectors and tuning behavior. While the MIDI landscape has expanded significantly since its inception, NepTune adopts three core components: the original MIDI 1.0 protocol, the MIDI transport specifications for both 5-pin DIN and USB MIDI 1.0, and the MIDI Tuning Standard for alternate and microtonal scale support. Each of these plays a vital role in shaping how NepTune communicates and responds in a modern performance environment. The following sections explore these standards in detail, highlighting their structure, purpose, and implications for implementation.

#### MIDI 1.0 Standard

MIDI 1.0 is a digital protocol (first published in 1983) for exchanging musical performance data between electronic instruments and controllers. At the heart of MIDI 1.0 is a simple byte-oriented message format. Each MIDI message starts with a status byte (whose most significant bit is 1) and is followed by one or two data bytes (MSB=0) [24]. This MSB convention distinguishes commands from values. The status byte encodes the message type (e.g. Note On, Control Change) and the MIDI channel (0–15). For example, a Note Off event is encoded as status 0x8n followed by two data bytes (note number and release velocity), and a Note On is 0x9n with note number and attack velocity [25]. Other Channel Voice Messages include Polyphonic Aftertouch (0xAn, key pressure), Control Change (0xBn, controllers like modulation wheel), Program Change (0xCn, one-byte patch number), Channel Pressure (0xDn, overall pressure), and Pitch Bend (0xEn, two-byte 14-bit value). These allow rich expressive control: for instance, velocity sensitivity, modulation, and continuous pitch bend are all built in. Because each channel has 128 possible note or controller values (7 bits), MIDI can transmit fine-grained performance data.

## MIDI MESSAGE

eeee nnnn Okkkkkkk 0vvvvvvv

eeee - event: 1001 note on    nnnn - channel 0-16

1000 note off

[eeee nnnn - status byte]    kkkkkkk - key data byte: between 0-128

vvvvvv - velocity data byte: between 0-128

**Figure 5.3:** Binary format of MIDI message.

Beyond channel messages, MIDI 1.0 also defines system messages and synchronization. A System Exclusive (SysEx) message (status 0xF0) lets manufacturers send custom data; it begins with a manufacturer ID and ends with 0xF7 [25]. This is how bulk data (like sound presets or tuning tables) is shared. MIDI also includes System Real-Time messages (single-byte codes 0xF8–0xFF) for timing and control: for example, Timing Clock (0xF8) is sent at 24 pulses per quarter-note, Start (0xFA) and Stop (0xFC) control sequencer playback [25]. Finally, Channel Mode messages (Control Change 120–127) let a MIDI master reset or reconfigure a device: All Notes Off, Reset Controllers, Omni On/Off, Mono On (Poly Off) and Poly On (Mono Off) are defined.

While NepTune supports polyphony and is capable of producing multiple simultaneous voices, it does not implement polyphonic aftertouch. In the MIDI 1.0 specification, polyphonic aftertouch (status byte 0xAn) allows a performer to apply individual pressure values to each key after it has been struck, enabling highly expressive control where each note can modulate parameters independently such as filter cutoff or vibrato depth. However, NepTune only interprets channel aftertouch (0xDn), which applies a single pressure value globally to all notes on a given channel [25]. As a result, even if a controller sends polyphonic aftertouch messages, NepTune will not respond to per-key pressure variation; instead, it may either ignore those messages or translate them into a general channel pressure response. This design choice simplifies the instrument’s internal modulation architecture while preserving standard expressiveness through velocity, pitch bend, modulation wheel, and other control changes.

NepTune remains fully compatible with standard MIDI polyphony and responds appropriately to note-on and note-off messages across its voice structure. However, users should be aware that the nuance of individual key pressure control—while supported by some high-end MIDI controllers—is not implemented in this synthesizer and will not affect its output.

### MIDI Transports (5-pin DIN and USB)

The original transport layer defined by MIDI 1.0 is the 5-pin DIN interface, a simple electrical standard designed for reliable point-to-point communication. Operating at a fixed baud rate of 31,250 bits per second, this serial connection uses an opto-isolated current loop to transmit data, where a logical “0” corresponds to approximately 5 mA of current through an optocoupler and a logical “1” corresponds to no current. The standard connector is a 180° 5-pin DIN jack, although only pins 4 and 5 are used for data transmission and pin 2 serves as a shield ground; pins 1 and 3 remain unused. Cables are typically constructed as a shielded twisted pair and limited to a maximum length of 15 meters (about 50 feet) to preserve signal integrity. Devices adhering to this standard implement separate MIDI OUT and MIDI IN ports, since the communication is unidirectional, and often include a MIDI THRU port to pass incoming data along to additional devices. However, excessive chaining through THRU ports can introduce latency and signal degradation. The use of opto-isolation at the input stage provides critical protection against ground loops and ensures compatibility across different electrical environments. Notably, an updated specification in 2014 allowed for the use of 3.3 V logic levels in place of the original 5 V standard, broadening compatibility with modern low-voltage hardware.

In addition, NepTune supports USB MIDI 1.0 on a USB port. USB provides a very different transport layer: a USB-MIDI device class (part of the USB Audio class) defines how standard MIDI messages are carried over USB packets. The big advantages of USB are bandwidth and flexibility. USB full-speed (12 Mb/s) is hundreds of times faster than the 31 kb/s DIN link, so latency is negligible. Crucially, USB can carry multiple virtual MIDI cables on one wire. Each virtual cable behaves like a separate 16-channel MIDI port. The USB-MIDI 1.0 spec uses 32-bit packets, where the first 8 bits identify a “Cable Number” (0–15) and message Code Index, and the remaining 24 bits hold the actual MIDI bytes [26]. This means a single USB connection can multiplex many independent MIDI streams. For example, a USB cable could carry 8 or 16 virtual MIDI ports (128 or 256 total channels) if the device supports it [26]. In practice, NepTune will enumerate as a USB-MIDI device, appearing to the host computer as an Audio/MIDI class peripheral. The computer (acting as USB Host) then sends and receives MIDI messages to NepTune on this port. Because USB is bidirectional, one cable handles both MIDI In and Out (unlike DIN which required separate cables).

### MIDI Tuning Standard

The MIDI Tuning Standard (MTS) is an extension to MIDI 1.0 that lets instruments use alternative tuning systems (microtonal scales) instead of (or in addition to) the default 12-tone equal temperament. As defined by the MIDI Manufacturers Association, MTS “allows the sharing of ‘microtunings’ (user-defined scales other than 12-tone equal temperament) among instruments, and the switching of these tunings during real-time performance” [27]. In practice, this means NepTune can be instructed to tune its notes to any desired frequencies on the fly.

MTS uses MIDI System Exclusive (SysEx) messages to transmit tuning data. The core messages include the Bulk Tuning Dump and Single-Note Tuning Change. A Bulk Tuning Dump transmits a complete tuning table that maps all 128 MIDI note numbers to specific frequencies or cent offsets. When received, the instrument updates its internal tuning reference accordingly, enabling support for non-standard tuning systems such as just intonation, historical temperaments, or custom microtonal scales. The Single-Note Tuning Change message allows for real-time adjustment of individual notes without replacing the entire tuning table, making it useful for dynamic or performance-based retuning. Later additions to the specification introduced Scale/Octave Tuning, a more compact format in which tuning offsets are provided for just one 12-note octave. The synthesizer then extrapolates this tuning across all octaves by repeating the pattern, which is particularly efficient for repeating scales such as 19-TET or other structured microtonal systems.

In NepTune’s design, which features a fixed 29-key layout (17 white keys and 12 black keys) and octave-shifting controls via up and down buttons, Scale/Octave Tuning becomes especially practical. Since NepTune does not present the full 88-key piano range at once, octave selection is determined by the current state of the octave control buttons. The selected octave range determines how incoming MIDI note numbers are mapped to the 29 physical keys. When using Scale/Octave Tuning, NepTune can apply the specified 12-note pattern consistently as the user shifts between octaves, ensuring that each note retains its intended microtonal offset across all playable ranges. This approach minimizes the amount

of tuning data required, supports expressive alternate tunings, and integrates naturally with NepTune's limited-key physical interface.

### 5.1.3 Digital Audio Standard: AES3

AES3, also known as AES/EBU, is the professional digital audio interface standard developed jointly by the Audio Engineering Society (AES) and the European Broadcasting Union (EBU). AES3 defines a protocol for transmitting uncompressed, high fidelity digital audio between devices such as synthesizers, mixing consoles, and audio interfaces. This standard promises low latency ensured communication; this extends its usefulness when multiple devices are following the standard as compatibility allows for connections between different professional studio equipment and broadcast devices. While the physical aspects to this standard are not completely relevant for the sake of how NepTune operates, it is important to highlight the pertinent information that does exist as well as the rest for education, to create a better understanding of AES3, and how it impacts the industry.

Neptune should appeal to the widest range of users possible and one way to assist this is by taking advantage of AES3's both balanced and unbalanced connections. For professional environments, this implies use of shielded twisted-pair cables, either with XLR or TRS connection, operating at nominal impedance of  $110\Omega$  and voltage range between 2-7 V peak-to-peak. With the recommended configuration, noise interference is minimized, allowing for incredibly long cables and fast sampling rates at 48kHz. Alternatively, to accommodate consumers, another connection type should be used, the S/PDIF (Sony/Philips Digital Interface) and that involves  $75\Omega$  coaxial cables as well as optionally, optical connections for lower peak-to-peak voltages.

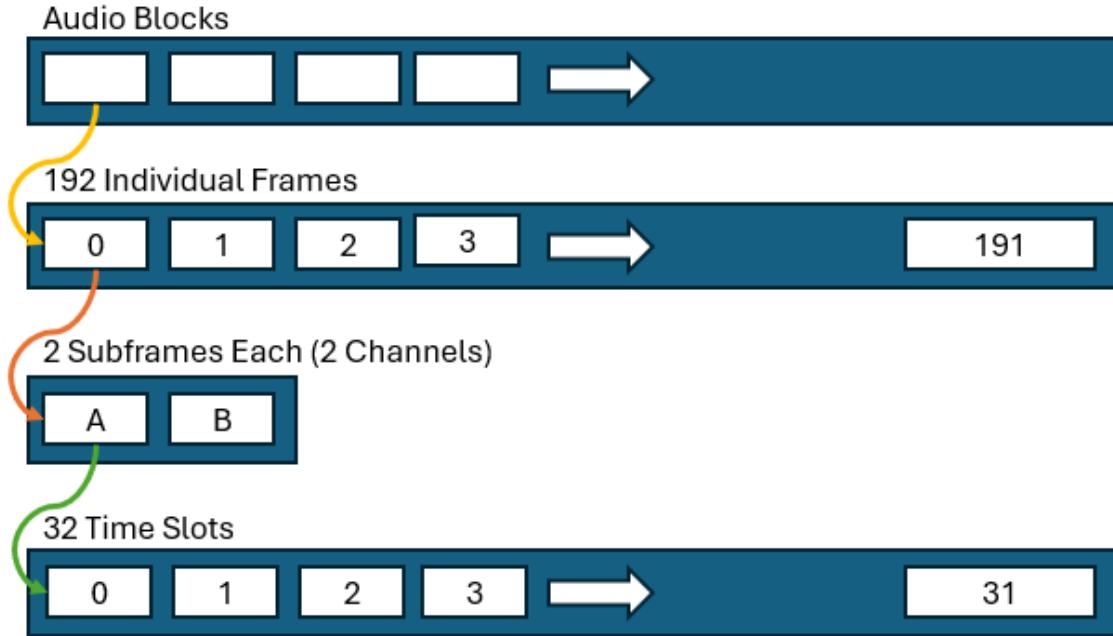
The protocol also organizes audio data into frames with each containing two subframes (one for each stereo channel). Every subframe includes a 4-bit preamble for synchronization, followed by a 20- or 24-bit audio sample, and auxiliary bits for metadata and error checking. Critical among these are the channel status bits, which provide required information like sample rate, timecode, and copyright flags. The data is encoded using Biphase Mark Coding (BMC), a method that ensures DC balance and embeds clock signals within the data stream, eliminating the need for a separate clock line. Framing like the AES3 standard brings the guarantee of effective transmission and audio quality, which is an option for the design team to consider in the case that timing remains an issue for the MCU as implementation progresses.

A quick visual layout of each subframe is shown below:

**Table 5.1: Bit Comparison of AES3 Subframe**

Bit slot(s)	Name	Function
0-3	Preamble	Synchronization for frames
4-7	Auxiliary Sample (optional)	Low-quality auxiliary channel to be used based on bit 30, the

		channel status, usually for inter-studio communication.
4-27 / 8-27	Audio Sample	Includes bits 4-8 if unused by auxiliary, and stores one sample of the audio. Stores MSB (most significant bit) last.
28	Validity	The bit is low if audio data is detected as correct and suitable for D/A conversion. If defective samples are detected, the receiving device is signaled to mute output.
29	User Data	Serial data stream for each channel, formatting specified by bit 30 as well.
30	Channel Status	Each frame contributes 1 bit into a 192-bit channel status word and the structure depends on whether AES3 or S/PDIF is operating.
31	Parity	Even weighted parity for error detection. Bits 4-31 have even number of ones and preamble is ignored.



**Figure 5.4:** Diagram of Audio Block makeup using either AES3 or S/PDIF

AES3 uses different signaling when configured to balanced mode which rejects electromagnetic interference. This enhances the usefulness of standardized devices in studio environments; increasing the proficiency of NepTune in professional contexts. AES3 also ensures deterministic latency which is the strict guarantee that audio devices greatly benefit from for precise control in real-time production and processing.

What sets AES3 apart from the similar standard, S/PDIF is only really in implementation. AES3 operates with higher voltages and balanced connections which creates professional reliability, whereas S/PDIF is optimized for simpler wiring and easier implementation. Also, a meticulous detail differing the two is that AES3 carries metadata in the channel status bits such as track identifiers, while S/PDIF carries simpler data such as emphasis flags (activating pre-emphasis or not).

Because NepTune is a hybrid synth, AES3 is not fully implementable as the main purpose, avoiding analog conversion for more secure transmission of audio data, is cast redundant by the analog hardware designed to convert the digital values read by the potentiometers and user control. However, the application could still be exploited by simply integrating AES3 via adapters that bridge the digital and analog aspects of the signal so as to maintain the integrity of the data via this standard. AES3 is not without drawbacks however, due to the stereo format, multichannel audio requires multiple cables meaning that for the adapter integration, more than one cable would be required to carry the multiple channels that NepTune provides. Additionally, timing issues are not completely out of the picture as

connections between devices need to account for clock jitter, this requires terminations and impedance matching which prevent the signal reflections that can occur.

AES3 is a great tool if the direction of NepTune becomes fully digital or if the timing problems point towards the solution of integrating adapters, it was also helpful in learning of alternatives like S/PDIF which is an easier to incorporate alternative. Having the certainty of timing, quality and compatibility are all great benefits to the final product so adding any of these certainties would be sure to improve NepTune's performance.

## 5.2 Constraints

### 5.2.1 Major Constraints

#### Cost

A major constraint under consideration at all times for *NepTune* is cost. Cost is a target objective for engineering, and a main marketing target as well. This mean that most decisions taken have been based on cost. The targe goal is for the synthesizer to cost a net total of 500\$ for the consumer, that means all major decisions when it comes to part selection depend on meeting the final price. An example of how this affected the project is through choosing an IC for VCF, and the configuration for the filters. Initially a high-pass and low-pass Sallen-key filter was chosen and the idea was to utilize one IC for two filters, a high-pass filter and low-pass filter, which will be utilized per voice. However, a closer examination of the IC SSI2140 that was chosen revealed that the resonance control is connected to only one of the poles, meaning that in order to be able to control the resonance only one filter would be doable per IC. However, this is not feasible because of the goal of 4 Voices that would be 8 ICs which is 45 \$ of the budget. Therefore, a change was made to the filter design and a simple 4<sup>th</sup> order low pass was chosen instead.

Another aspect of cost being a deterrent was during the selection process of other IC's. For example, when examining the choice for the VCA cost was a major significance because if additional voices were added it would cost more to choose the initial choice as the first choice examined cost 10\$ per IC, SSI2164 was the more cheaper choice as it was only 5\$ per IC, therefore SSI2164 was chosen for cost purposes as one of the main reasons.

The decision making for the audio amplifier also followed similar logic, when comparing the different audio operational amplifiers the one chosen, the ADA4075-2 was not the first choice when it comes to performance, as the AD797 was the superior choice, however, AD797 cost 18\$ per unit so it was not the ideal choice for budgeting, hence the ADA40755-2 was chosen as it cost 3.75\$. This part could have been sourced from one of the advisors with no spending cost to the team, however, since the goal was to create a product that would cost less than 500\$ for the consumer, it was rejected as it would contribute significantly to the total cost.

Another one of the main objects effected by the cost constraint was the keys. The keys chosen for this project were mechanical keys. However, part of the original design conception was the inclusion of piano keys for aesthetic purposes and ease of usage when

it came to user interface. However, since Neptune requires 24 keys, piano keys would cost 192\$ which is a major part of the budget, therefore the choice to use mechanical keys as it would only cost 48\$ to obtain the keys needed.

Finally, the user interface is effected by cost. Because part of marketing appeal is making the interface interesting to the users eye, it is important to purchase aesthetically appealing controls. For example, a slider tends to be more appealing to the user than a knob. However, sliders are more expensive to purchase on the current budget of the device. Moreover, the interface could include LED lights that could indicate different things to the user to make it more appealing as that is a common feature in most synths, however, that is also an additional cost that cannot be tolerated by the current budget as well.

### Size

Size is another major constraint to the design due to a number of reasons. The main reason this was chosen was for portability purposes which is a marketing requirement. If the *NepTune* is too bulky it would be too inconvenient to carry, weight also being a constraint. However, this leads to multitude of issues with the design first there is a constraint when it comes to PCB and schematic design, The limited space means limiting the amount of components needed to be manipulated.

For instance, one of the main parts that had to be reduce due to size is the amount voices implemented. Since *NepTune* is a polyphonic synth each voice requires its own path into the eventual mixer and amplifier, which means separate VCFs and LFOs are needed for each voice, as well as two oscillators per voice. The goal for *NepTune* is set to be four voices as of right now because of the amount of space each voices path would take, the future goal is to increase the amount of voices to 6, nonetheless, this remains a stretch goal due to the multitude of circuitry required to fit.

Another issue due to size constraints is the inclusion of more features. One of the stretch goals is the inclusion of an LCD screen, the constraint of size then becomes a huge deterrent to this goal. That is because the layout is already limited to all of the current controls that utilize the current span of the design. Therefore, with the use of the 32 buttons and knobs a goal of including a screen becomes difficult due to the user interface being crowded as well as the fact that the LCD screen would require a serious amount of circuit routing that would require more space.

Size also offered a limitation for the amount of keys present. Because *NepTune* is limited in size. The design was conducted with the goal of creating an instrument that had 4 octaves an octave is reflected in keys as 12 keys, so 48 keys are needed to generate the octave goal. However, due to size constraints the team is unable to fit 48 keys onto the instrument. Therefore, the design was changed to include only 24 keys, and the octaves would be changed by a switch. The keyboard was also affected because the choice of keys, while majorly influenced by cost, was also determined by size. Keyboard keys would take up space that is needed for more significant components, and designing specific keys would be too costly therefore there was no choice except utilizing keyboard switches because they offer the smallest price, and the smallest key option that is readily available. Another choice

dictated by size was the shape selection of the keyboard, part of the initial conception included an ergonomic keyboard, however, that is not doable because it would require a larger design to allow for the expansion of the keyboard in that specific manner.

Size was also constraint in regards to the sound output choice. Initially, a speaker was a design element that was aimed to be included, but since space is limited due to size there was a limit on to what output was possible. Line out and headphone

output allow for the most efficient space usage. A speaker circuit would utilize at least 4in of space and would require excessive connectivity that is not feasible with the limit of the size of the device.

Size also limits the project ability in other multitudes because it does not allow for a larger interface. A larger interface would allow for other features that utilize user control, for instance an arpeggiator, multiple filters, and more presets. All of these, while also limited by cost, are severely limited by the lack of space due to complexity of the design, space for the PCBs, and space for user interface.

### Time

Time is a major issue for *NepTune* that is because there are a multitude of things to consider, design, and build, but the time for the conception of this project is limited. Typically, a student senior design project is allotted 32 Weeks for completion. However, in this case the project is only allotted 28 weeks due to it taking place during the summer and winter semester.

This poses a threat as the report, initial designs, and all the planning must be written and completed in approximately 11 weeks. An entire month is lost because of the summer, and multiple days due to thanksgiving break. This all takes away from the time to work on the project. This also leaves only 15 weeks for all design alteration.

### 5.2.2 Minor Constraints

#### Maintainability

A key concern about any product is whether or not it will stand the test of time, and this consideration is what often keeps products from being helpful to being staples of someone's production tools. A synthesizer that uses contemporary, available parts which have been tested and documented, as well as a chassis and construction that will provide stable working conditions for a reasonably long time. Because of the complexity of a device such as a synthesizer, components are required to be accessible for maintenance and replacement to keep the overall cost of sustaining *NepTune* to a minimum for the user.

When considering which parts would be included for the design, the team made sure that nothing was too niche or old to obtain within reason, and that the parts included were tested and prevalent in the music production market. And with careful consideration for the

readability of the PCBs, a user would reasonably be able to identify and replace missing and broken parts with minimal effort. This ability was ensured by using only through-hole and simple solder components, avoiding complex mounting requirements such as Ball-Grid Array (BGA) and Quad Flat No-Lead (QFN). To explain, BGA requires balls of solder to be placed below the IC and then mounted, meaning that if there is a short or misaligned connection, troubleshooting and solving the issue is massively more difficult. QFNs have similar difficulties with mounting as the lack of leads makes accurate soldering difficult, further complicating the user's ability to maintain and expand upon the device.

Other small but overall useful considerations that the team had in mind during development were the ease of access to parts such as removable backplate, secure holdings, and multiple purpose independent ICs that could be replaced individually. By offering access to the PCBs as a whole unit and replaceable, users that prefer simpler albeit more costly replacements have the option to do so. Additionally, the keys to the keyboard, which are the only significant moving parts of NepTune, are hot-swappable, allowing for easy replacement.

### Latency

Real-time responsiveness is critical in any musical instrument, especially in synthesizers where the performer expects immediate auditory feedback from their input actions. Latency in this case is the delay between user interaction (such as pressing a key or turning a knob) and the corresponding sound output. In order for a synthesizer to maintain playability and expressive control, latency must be minimized.

In NepTune, latency originates from multiple sources: ADC sampling of potentiometers and control voltage inputs, digital signal processing through the Daisy Seed's ARM Cortex-M7 MCU, DAC conversion, and communication with external components like the PCM3060 codec. To ensure responsiveness, our system targets a round-trip latency below 10 milliseconds, a common benchmark for professional audio systems. This ensures that even rapid performance gestures translate to near-instantaneous sound output, preserving both rhythmic precision and the illusion of acoustic immediacy.

Achieving this requires several coordinated design decisions. First, the system is clocked at 96 kHz, providing sufficient temporal resolution for low-latency audio processing. Second, we use efficient signal routing and processing pipelines in the DaisySP framework, avoiding unnecessary computational overhead. Third, modulation sources like envelopes and LFOs are calculated at the audio rate when needed, instead of using slower control-rate updates that could cause noticeable stepping or delay.

NepTune also avoids relying on USB audio or any computer-based processing during sound generation. All sound and modulation are handled directly on the built-in hardware, which helps eliminate the unpredictable delays caused by operating system audio buffers. Additionally, using the PCM3060 audio codec helps keep latency low thanks to its fast I2S communication (or HW mode) and minimal internal processing delay.

By reducing delay at every stage, NepTune provides a responsive and natural playing experience that meets the expectations of both analog enthusiasts and digital musicians.

### Safety

A significant concern should be the operating safety of the device; extreme ramifications both legally and morally can be avoided with proper precautions and implementations. Areas of the highest risk are electrical failure, audible oversaturation levels, and physical safety. In order to keep the user from suffering injury and prolonged or sudden exposure to dangerous levels of audio, certain systems need to be tested and designed around such risks.

With electrical faults, shocks and heat can not only damage the internal components but injure users, meaning that research into preventative measures is paramount. One such preventative measure is the secure installation of wires; this can keep shorts, disconnections, and unexpected power from being sent throughout the device. In order to ensure that wires are safely included, fasteners and space must be adequate. Fortunately, NepTune's design does not involve high voltage hardware meaning that a relatively safe, 12V will be sufficient for powering the IC's and can therefore relax the requirements that were possible for safe power delivery. Still, secure connections and ground-bonding from the cable jacks to the chassis will ensure that shocks are not realistically expected throughout the products lifespan. Not only for the quality of sound but for the safety of the user, a positive symptom of including separate PCBs for numerous functions in the device allow the team to implement separate grounds for analog, digital, and power sources reducing risk of noise and electrical looping.

Heat is a major concern with keeping NepTune safe and functional; many ICs are rated for specified currents and voltages which if the design of the circuits connecting these components weren't safely deployed, could introduce overheating and damage to components. Such damage introduces multiple sources of safety concerns, not only can the damage to ICs create opportunities for electrical shocks, but overheating can scald the user, combust dangerous components such as capacitors, and invite risks of fire which can severely harm both the user and the environment that NepTune is used in. Especially for a portable device like NepTune, these considerations must take priority over all other aspects of design. This requires very methodical design and attention to parameters of the components included as well as considerations into the integration with hardware. For example, the team has decided to integrate heat vents on the chassis that face away from the user should they decide to rest the device on their lap or other, air restrictive positions.

Audio presents risk in two different ways, short and loud peaks that can pierce the user's eardrums, and prolonged, high volume outside the safe ranges of what the human body can tolerate. A solution should the issue of loud spikes of sound occurring present itself during testing, can be the integration of *slow-start circuits* meaning that voltages powering audible parts of the circuitry charge slow enough to avoid spikes. For the issue of prolonged damage to user's ears, precautionary documentation should be included advising against operating NepTune at its volume extremes for longer than the general health consensus and

also possibly limiting what the ICs are capable of outputting which will be considered and implemented during testing if necessary.

Another critical aspect of operational safety involves the device's firmware and software safeguards. Since NepTune relies on digital signal processing and amplifier control, software-based protections can be implemented to monitor and regulate output levels in real time. For instance, dynamic range compression or automatic gain control (AGC) could be integrated to prevent sudden volume surges, while a software limiter could enforce a maximum output threshold to protect against prolonged exposure to harmful audio levels. Another precautionary implementation, thermal monitoring via embedded sensors can allow the firmware to throttle performance or shut down components if unsafe temperatures are detected, further mitigating risks of overheating. These software redundancies complement the hardware protections already in place, ensuring a multi-layered approach to user safety without compromising functionality. Advanced programming would be involved in adding these elements to the design which for four electrical engineers and only one computer engineer is a daunting task, but all solutions should be considered when approaching constraints.

# Chapter 6 – Application of LLM Platforms

Large language models are a type of artificial intelligence that can generate human language and perform tasks.[19] These are trained on huge amounts of data which is then analyzed for patterns, and rules of the language. LLM's are relied on to perform tasks such as answering questions, summarizing texts, and writing. The general reliability of these tasks have made their use very popular in academics as they supposedly simplified what used to be very time-consuming tasks that required hours of work.

However, the issue with language models that general users failed to notice is that rather than producing logical answers, they produce answers based on patterns and therefore can create information based on patterns that it believes most likely answers the question. This can produce a large problem when it comes to acquiring information.

In recent years, the rapid advancement of large language models (LLMs) has transformed the way students and researchers approach information gathering, technical writing, and ideation. This chapter explores how the use of LLMs, such as OpenAI's ChatGPT and other competing models, has influenced the development of this report—both positively and negatively. By examining specific examples and comparing different models, this discussion aims to evaluate their role as research aids, assess their limitations, and reflect on their overall impact on the research process.

## 6.1 Comparison of Models

Two main LLMs were uses during the design process of *NepTune* OpenAI and DeepSeek, These were all used as to not rely on a single source, since each model has different strengths and weaknesses. A closer examination of each model and how it was used as follows.

### 6.1.1 OpenAI – ChatGPT

The first model that was used for this project is ChatGPT this is a popular LLM that is often used by wide variety of people to summarize articles, explain and answer questions regarding concepts, and analyze images. However, ChatGPT generates responses based on a pattern and is not an accurate source of information and therefore tends to provide false information often. For *NepTune* this was used to explain unfamiliar concepts regarding synthesizers.

**Case Study 1:** *Question: “What is the difference between a voice and oscillator in a synthesizer?” and this was the response: Reference (Appendix E. Answer 1.)*

In this case study the aim was to test ChatGPT's ability to explain basic concept, since most of the team were unfamiliar with the basics of how a synthesizer works. ChatGPT explained the difference between a voice and oscillator well, but it was a shallow explanation, as the technical definitions is lacking. It did not explain fully how an oscillator functions within a voice and how it contributes to the sound but rather focused on the fact that a voice is made up of an oscillator. Therefore, it was asked to provide more resources

to proof that it had a logical reasoning for its answers. When asked to provide sources for this information

Given that ChatGPT sited both Wikipedia and reddit as a justification for its answers, it is highly unreliable source of information. Wikipedia is a site subject to change based on random people editing, and reddit questions is posted by people not necessarily expert at the topics. Even for a basic question of difference, therefore, no answer taken from here will be accepted without follow up research completed to validate the information. However, this answer provides a good basis for further research and understanding and can be used with solid resources to be able to understand the subject matter correctly.

**Case Study 2:** *Question: “Between notch, Cauer-elliptical, Sallen and key, and unity drive filters which one should I use for my hybrid synthesizer?”*

For this case study it was asked that a comparison is made between all the possible filter topology choices, and for one to be chosen. ChatGPT generated a pleasant graph (refer to Appendix E. Answer 2) however, the reasonings it gave for the Pros and Cons were mostly generic and not very technically inclined. It stated that a bandpass sound is unsuited for synthesizer, yet that is not true in application as it is a common sound used in multitude of synthesizers. The chart also has many inconsistencies when describing the pros and cons, and reasoning for some it described sound of the filter, and other poles that a single filter takes. This is not a consistent way of comparison for a user looking for more comprehensive understanding, and so further questions would be needed on each comparison for true deeper understanding.

Another example, it described and therefore not useful to the project towards the design aspect because after conducting research it is apparent that that some of the filters it denied as inappropriate have been used in other synthesizers and are more highly recommended for synthesizers.

**Case Study 3:** *Question “what is the difference between a linear regulator and switching regulator and which one should I use?”*

In this case ChatGPT listed the comparison very thoroughly, and was accurate to the researched knowledge. It said that the linear regulator was low efficiency which was correct, and the switching a high efficiency which was also correct. It also listed the other important fact which is that linear regulators are terrible at heat dissipation, but good at producing low noise.

Next, to answer the question of when to use each regulator, it stated that a linear regulator was best used for when the input voltage is slightly higher than output voltage, and best used for low noise for analog which is accurate to the research. It also stated the benefit of switching which is higher efficiency but is best for larger input-output differences.

### 6.1.2 DeepSeek – DeepSeek R1

DeepSeek AI is a LLM that was mostly used in this project because of its more advanced abilities to answer questions accurately in comparison to ChatGPT, moreover, the fact that

unlimited questions can be asked is also an important ability when compared to other resources, as you have the ability to constantly refine your questions to obtain clearer answers. The drawback of DeepSeek however is that it cannot create files outside of text, and will need redirecting at times.

### **Case Study 1 – Digital Processing Systems**

*Question: “I am designing a hybrid digital/analog synthesizer. Discuss the pros and cons of going with an MCU vs an FPGA”*

DeepSeek responded, “Choosing between an MCU (like STM32) and an FPGA for a hybrid digital/analog synthesizer involves critical trade-offs in performance, flexibility, development effort, and cost. Here's a detailed comparison.”

It then proceeded to list out the pros and cons of both and MCU and FPGA

| When requesting a comparison between the practical applications of a microcontroller or an FPGA for synthesizer design, DeepSeek provided a pros and cons list for each technology, outlining parameters deemed critical to consider in the design process.

### **Case Study 2- Question: “ List standards that apply to synthesizers”**

In this question the assignment was to look for inspiration regarding the possible standards that could apply for synthesizers. Upon initial view DeepSeek gave great organized lists of possible standards to examine. Nonetheless, many of the standards listed were outdated, such as UL 60065 and IEC 60950 which no longer exist and have been replaced by newer standards. When asked for site sources many of the ones given were dead links that did not lead to any substantial material in relation to its answer.

Moreover, some of these standards are more of a stretch in regard to applying to the synthesizer, such as the WEEE-Waste Electrical and Electronic Equipment recycling compliance, and other standards which were not necessarily applicable to a synthesizer. However, this provides a good introduction to potential standards to research because UL 60065 was used to find the most recent UL standard that apply to our synthesizer.

### **Case Study 3: Question “Which op-amps are best suited for analog audio devices?**

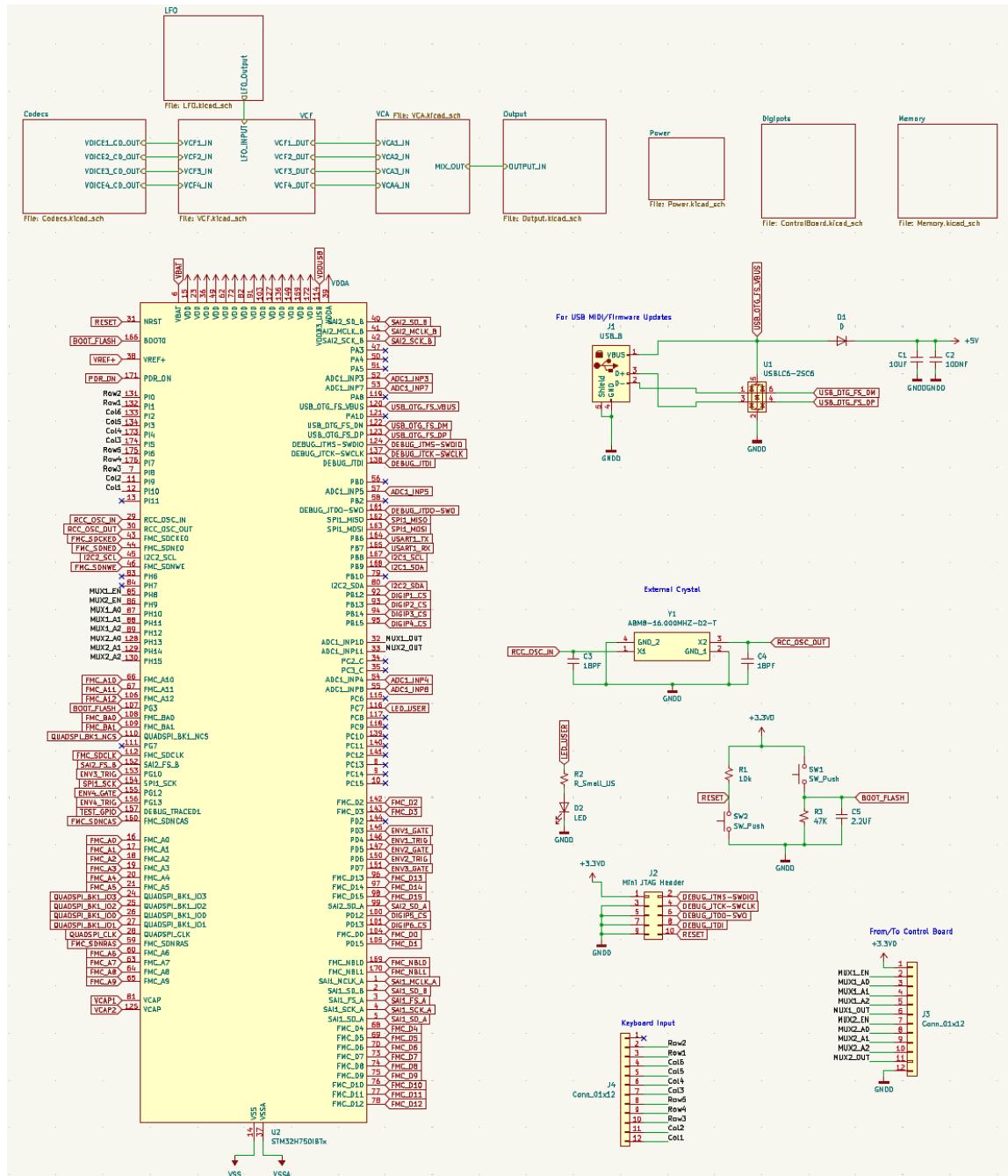
For this question, DeepSeek’s information accuracy was being tested. Since our device is distortion sensitive, the question was whether DeepSeek’s information was as accurate as our research. DeepSeek’s split its recommendations based on brands rather than performance, which isn’t ideal since selection would be preferred on quality, but it’s not a terrible organization. It also gave a recommended usage of each op-amp it suggested which included categories for mics preamps. Headphone amps. Guitar pedals, hi-fi dacs/amps,

and budget audio. These were found to be a rather specific applications that does not necessarily describe why these op-amps are best suited for analog applications.

Although, it did recommend the op-amp that *NepTune* is using which is TL072. It describes it as good for guitar effect which is an odd description for motivation to choose the device for analog devices design. Therefore, while this list was extensive in possible options for usage of op-amps, it offers a simple list of recommendation rather than actual facts and reasoning to make a logical choice.

## Chapter 7 – Hardware Design

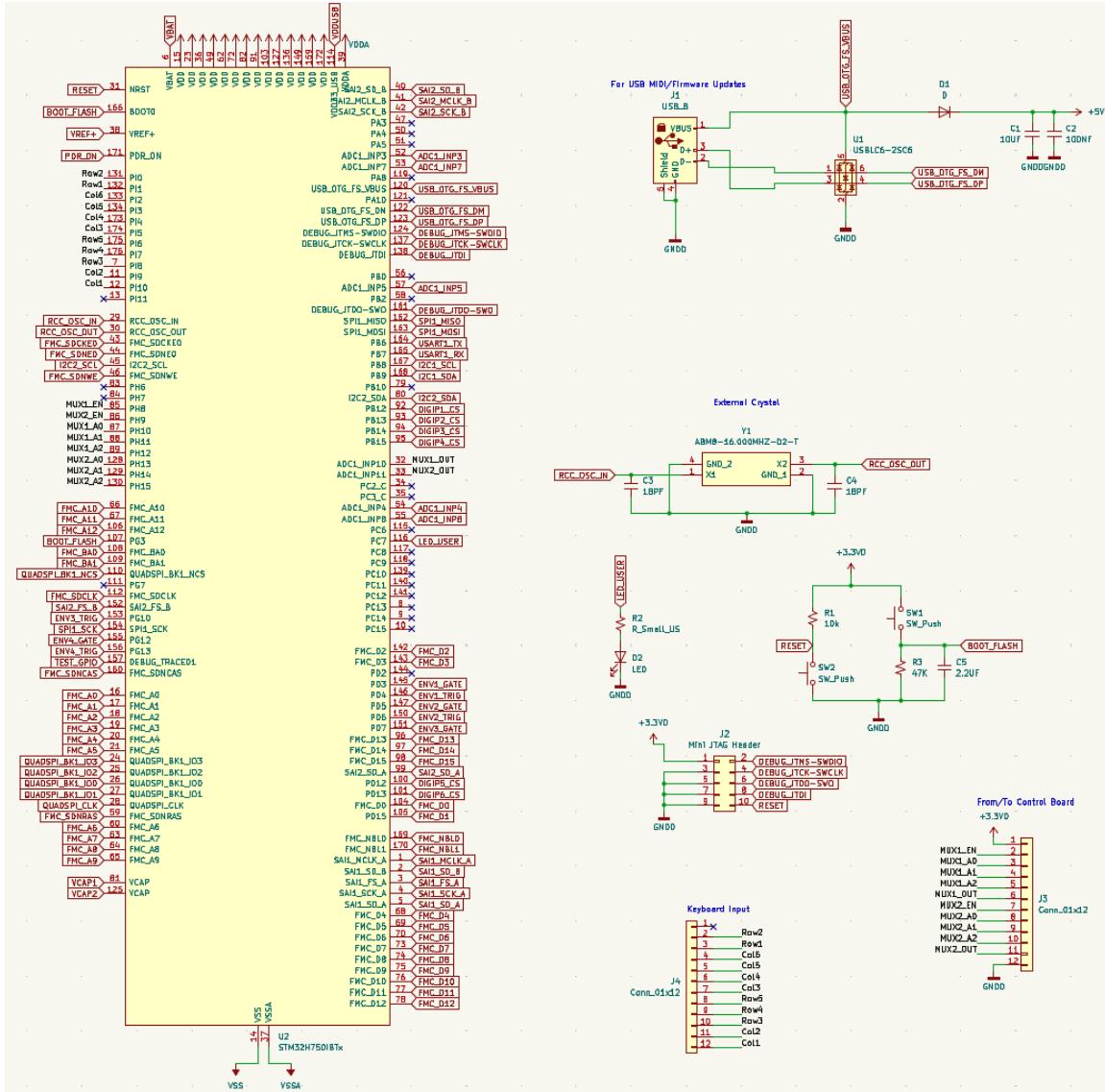
## 7.1 Overall Schematic



**Figure 7.1:** Overall schematic of synthesizer shown in hierachal format. Smaller subsystems are represented by blocks in the top portion of the image.

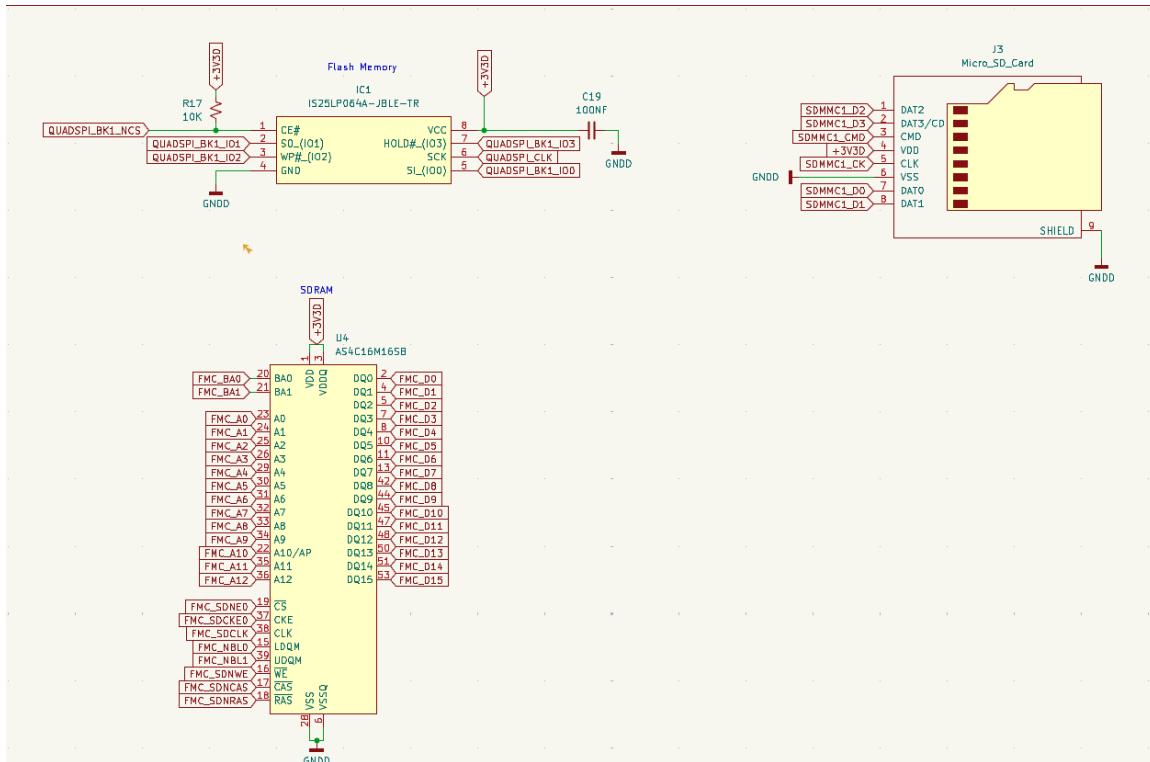
## 7.2 MCU Circuit

The following schematic covers the pins utilized in the STM32H750, a USB-B connection for potential USB MIDI implementation and MCU programming and a mini-JTAG header for direct programming of the MCU. A 16MHz, 18PF external crystal oscillator is connected to the MCU per its datasheet. The Keyboard connector connects this board to the keyboard circuit used for playing notes.



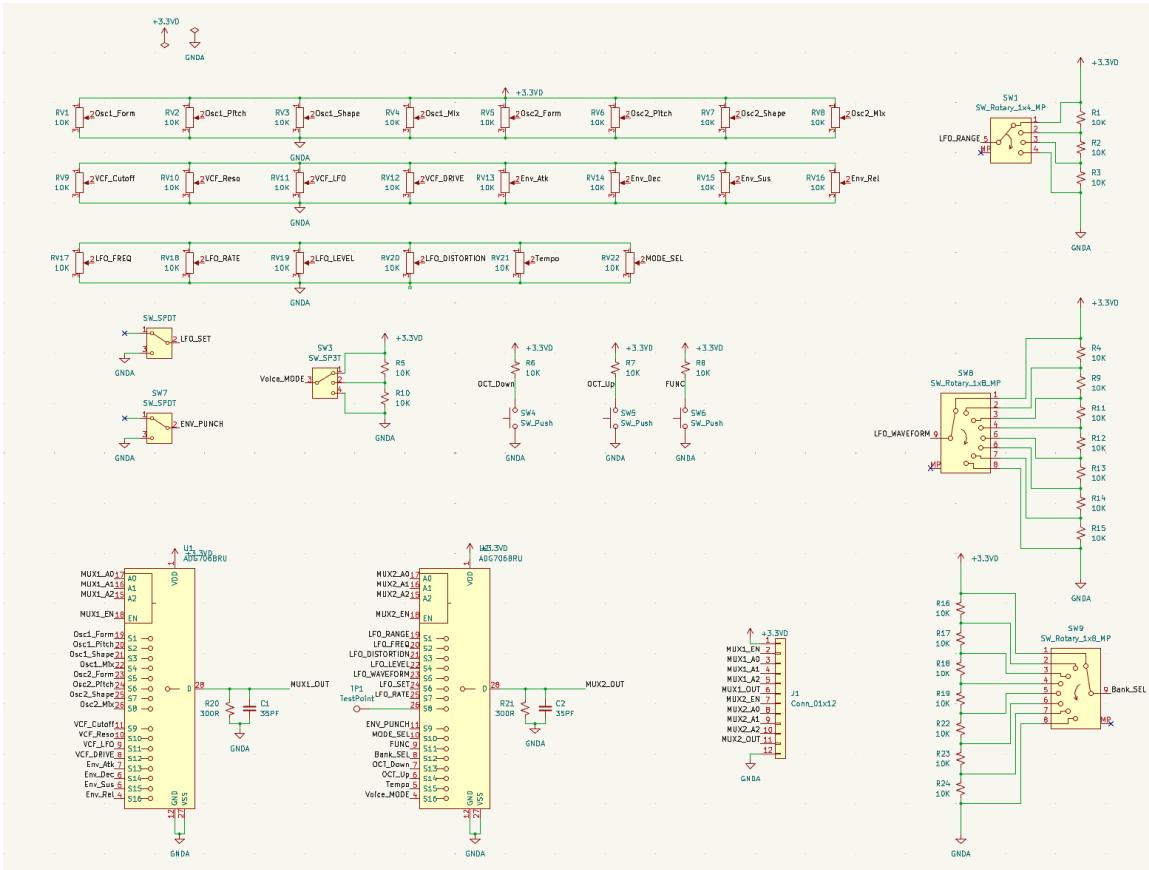
**Figure 7.2:** MCU Schematic.

## 7.3 Memory Circuit



**Figure 7.3:** Schematic for flash memory.

## 7.5 User Interface Circuit



**Figure 7.4:** Schematic for interface controls. These are the buttons and knobs on the face of the board that the user will interact with.

## 7.6 CODEC Circuit

The following circuit is responsible for the digital-to-analog conversion of the sound signal generated by the MCU. As *NepTune*'s digital circuit design is inspired by the open-source Daisy Seed dev-board, the CODEC circuit is modeled after the one found on the Daisy Seed, with the main difference being the inclusion of a second PCM3060 CODEC, allowing potential for 4 voices. The PCM3060 can produce two differential analog outputs; however, it has been configured below for single-ended output. The MODE pin has been configured in a pull-down position via a 220K resistor as stated in the datasheet for hardware mode, single-ended output.

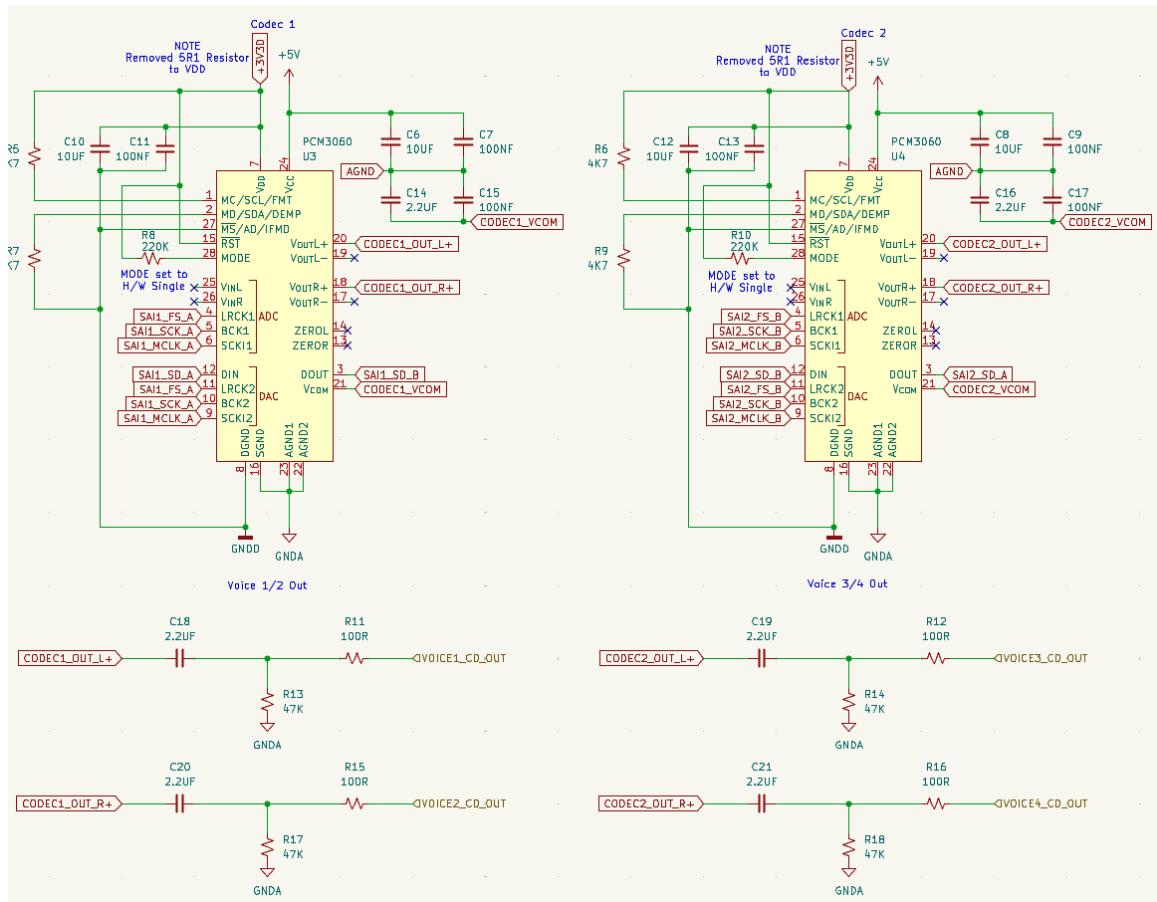
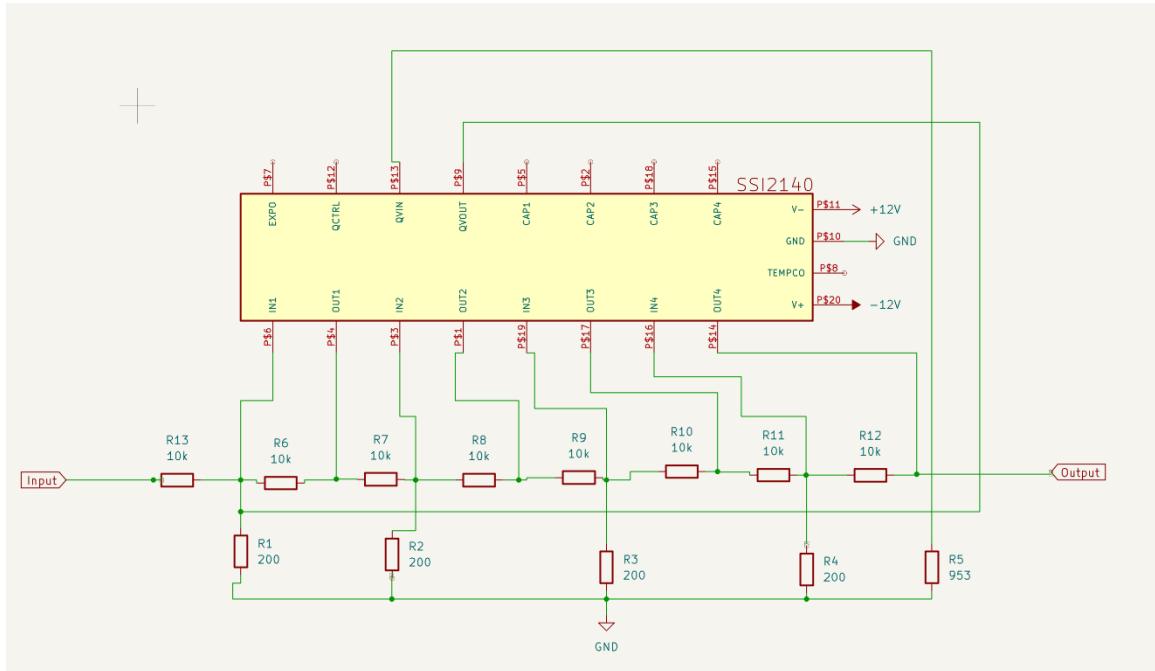


Figure 7.5: Schematic for both CODECs.

## 7.7 VCF Circuit



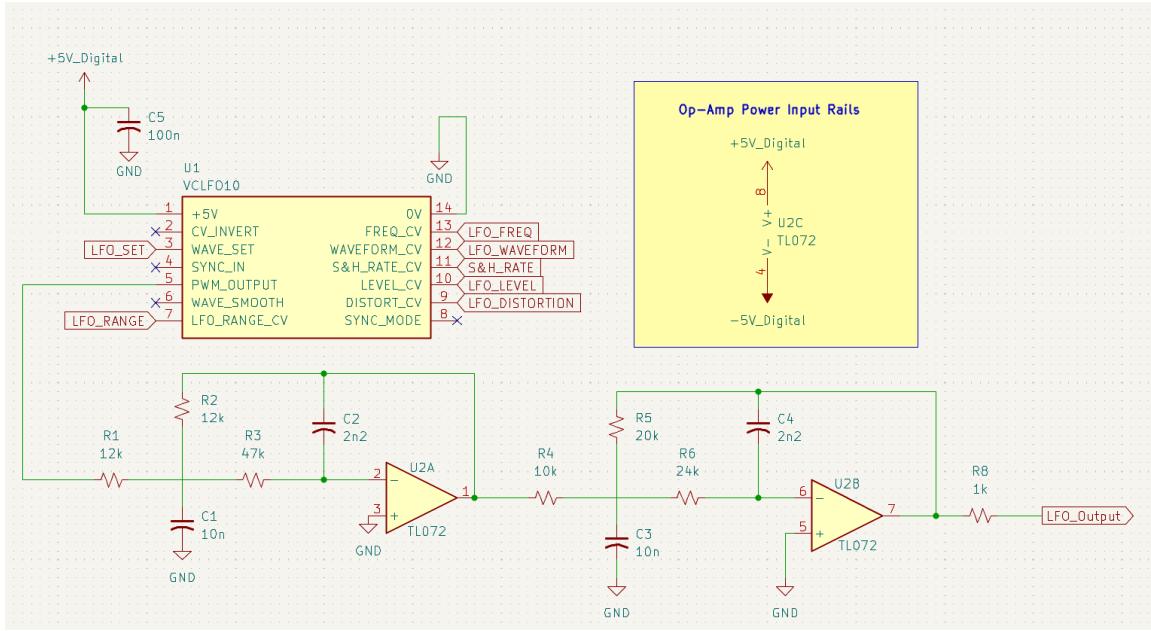
**Figure 7.6:** Schematic for voltage-controlled low-pass filter with output gain Q-compensation.

There are two possible configurations that are being considered for usage in *NepTune* a 4-pole low-pass filter, and a unity gain overdrive filter. Both of these configurations utilize the Q compensation system within the filter that is responsible for controlling resonance of the circuit. This is done through an external resistor that is responsible for controlling the loops gain, which in turn impacts the resonance of the filter. A potentiometer will be used to control the resonance.

The frequency control of the filter is determined by exponential frequency control input pin. This is controlled via a resistor in series with 1Kohm shaunt resistor. In this case as well the resistance is altered via potentiometer.

## 7.8 LFO Circuit

As previously mentioned, the LFO will be constructed using the VCLFO10 by Electric Druid. The VCLFO10 consists of a PIC microcontroller that generates waveforms via direct digital synthesis (DDS). The waveform is passed to the output pin via a PWM signal. The intended waveform is embedded within this PWM signal and can be extracted with a low-pass filter. Passing the signal through a low-pass filter will remove the high order harmonics in the PWM signal, leaving behind the desired waveform. The filter design below is based closely on the reference circuit from the VCLFO10 datasheet, utilizing a multiple feedback filter.



**Figure 7.7:** Schematic for LFO circuit including the VCLFO10 IC and output filters.

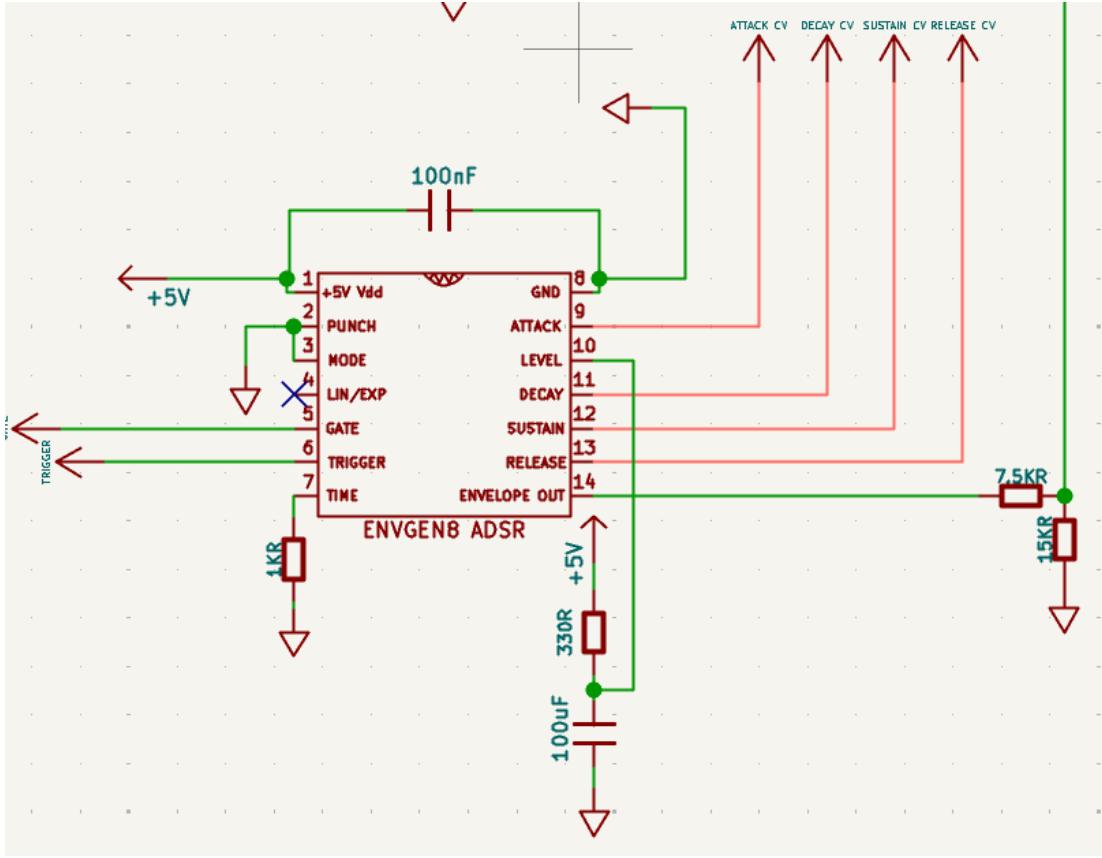
## 7.9 VCA, Envelope Generator, and Mixer Circuit

One of the most important considerations for developing the schematics of this board is whether each connection has been properly joined and labeled. With other boards being designed by other members of the team, clear and concise labels and connections will be extremely useful. Risk of miscommunication on names of connections, locations of connections, and routes of signals all exist in this development. To reduce this, the team worked together to keep ideas between interconnections of the PCBs clear and names of obvious intention.

Opportunities for error are only expanded when development of PCBs occurs. Boarding will bring positions of components to the front of the team's priorities and possible mistakes. NepTune's modular PCBs for different functions increase the number of out-of-trace connections, thus carrying the risk of design failure. That being said, the design of boarding the ADSR, VCA, and SSI2190 audio mixer together would be approached with all mentioned pitfalls in mind. This action translates into clear labeling, clear and effective connections, and present communication during design, meeting regularly to share ideas and provide feedback.

Starting with the ENVGEN8 envelope generator, this proved to be the most time-consuming aspect of the design of the PCB. Complexity was not a particular issue because of the very helpful schematic present in the datasheet, for example integration with another part, the AS3360 VCA. Understanding the purposes of each connection would be necessary for properly naming the connections to the user's control board. What proved to be time-consuming was that the ENVGEN8 IC was not identified in the stock library so to properly display the part on the schematic, a custom icon created by the team would have to be realized. Important aspects of this custom icon for ENVGEN8 were proper pin labelling to

reduce risk of misconnection and actual pinpoints for connections. This will allow the connections to appear when PCB creation occurs, simplifying the process.



**Figure 7.8:** Schematic for ADSR including ENVGEN8.

The only significant difference between NepTune's design and the one present in the example schematic is the source of the control voltages for each of the four values that the user changes according to the sound wanted. Here you see connections from the external User Interface Circuit shown in Figure 7.5. This will allow the user to turn knobs to change the control voltage entering the board through a wire and ultimately affect the timing and amplitude of the ADSR values as described previously in the research section.

After four ADSRs have been sectioned, they pass through voltage dividers which reduce the 0-5V envelope out signal in a 0 – 3.3V which matches most of the available voltages that can affect the VCA's operation amplifiers. The SSI2164 has a -33mV/dB conversion rate meaning that at its maximum rate of 100dB, a maximum voltage of 3.3V will be required, so additional voltage beyond will not contribute to the signal's amplification which could lead to unintended results in the sound produced, hence why the team felt necessary to reduce the voltage range on the line leading out of the ENVGEN8 chip.

All four outputs of the ADSR chips are inserted into the SSI2164 Quad VCA which is controlled by said envelope generator. These four values change the amplification behavior of the VCA so that they follow the envelope generated by the user's selected values. Then

the audio signals coming from the VCF are fed through the amplifiers and modulated according to the ADSR's values. This final modulation of the sound then follows into the audio mixer SSI2190. Here is where the four voices of NepTune are combined and output as a single signal. This will then lead to NepTunes speakers and audio jack.

#### Important notes for implementation of the ADSR envelope generators:

The ADSR is a middle ground of these ICs as the features that were available were fairly limited but still significantly available for customization. Likely due to the complexity of envelope generation itself, this left the team with little to consider for NepTunes features stemming from this IC. One feature that was considered is the PUNCH feature. This pin, when connected to ground, adds a flavor between the ATTACK and DECAY stages in which the sound jumps up to add a *heavier* sound when a key is pressed. The only way to truly translate what this sounds like on paper is to describe the sound as "fuller" for the moment directly after a key is pressed. Because the team was not sure of the benefit or possible drawbacks of leaving this feature on or leaving it off, it was decided to include a user-controlled switch that can either power or ground the option.

Another pin-related feature is the MODE selection, which gave the option to switch the regular ADSR behavior into gated and LFO looping behavior. This would have been exploited by the design if a LFO was not already included for looping purposes or if the scope of the synthesizer was far larger. For this team's purposes, leaving it connected to ground would keep it in ADSR standard mode which is desirable for this medium complexity design.

The TIME feature allows the user to reduce the time that ATTACK, DECAY, and RELEASE take to complete from x1 to x0.01 which is too niche of a feature for the scope of NepTune's design. Another remarkable but relatively low impact decision made when drawing the ENVGEN8s schematics was the Linear/Exponential envelope option which was very clearly leaning towards exponential inclusion due to the enhanced audio results.

#### Important notes for implementation of the VCA:

The VCA was the simplest of the ICs, likely due to the very straightforward but nonetheless important operation. Other than being able to select the class of operation for the VCA, the SSI2164 has little to no customization for amplification purposes. This is not a downside as plenty of other components understandably hold these customization options instead.

Class A operation is selectable by allowing current into the pin and emphasizes lowest total harmonic distortion, which is a fairly unique aspect that many other components do not offer. Class AB operation instead prioritizes low noise on the signal which certainly is also important for the sound of NepTune, however this operation is included in many other components in the signals path, so Class A operation was selected. Additionally, one of the major engineering specifications was a less than %2 THD, which also encourages the design to be Class A rather than Class AB.

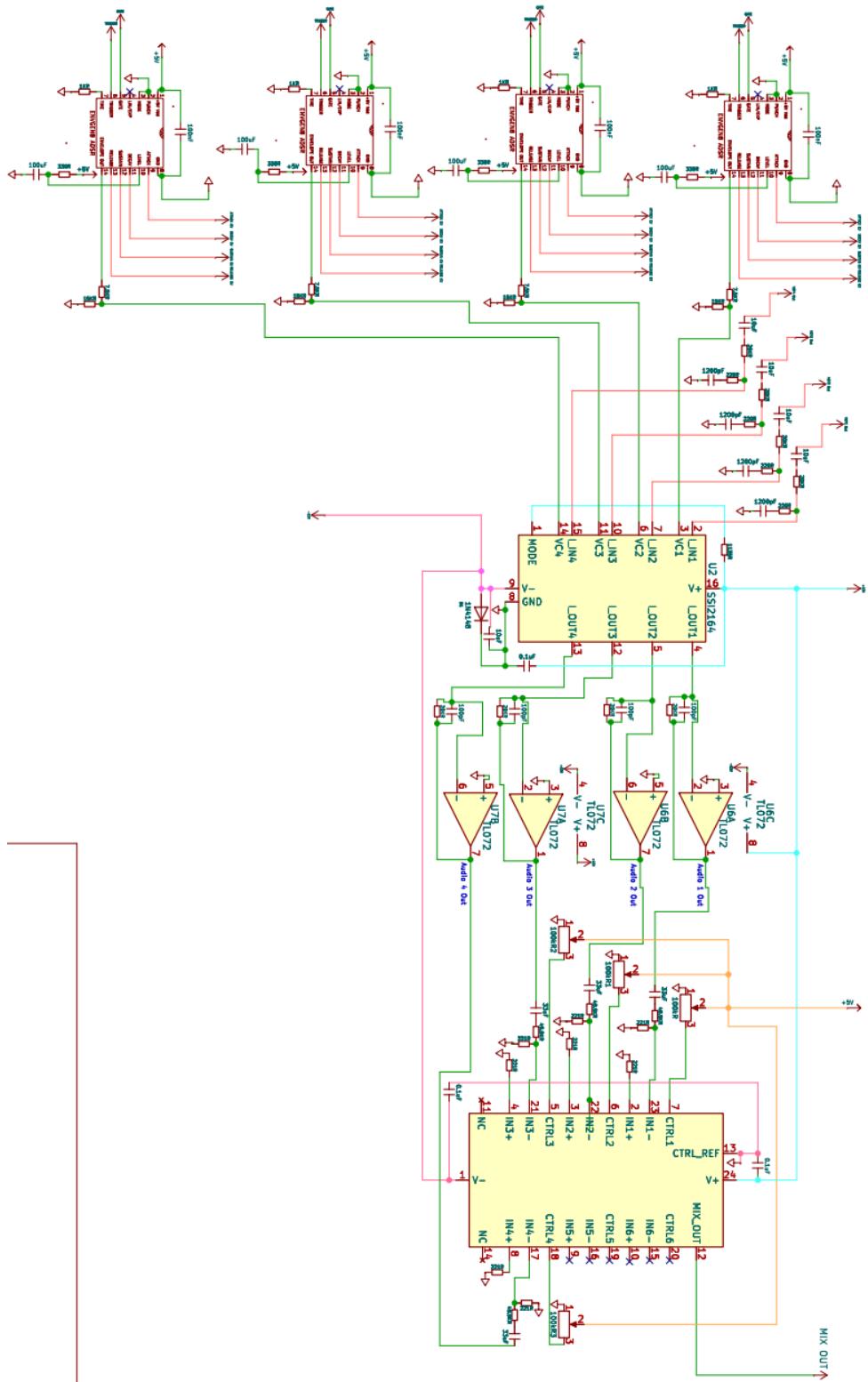
What is notable about SSI2164 but not necessarily relevant to the current design of NepTune is its sheer versatility. This IC, which is intended for audio mixing, can be

configured to operate as a 4-to-1 mixer, an accurate temperature compensated voltage to current converter, and could have been configured into an Ultra Low Noise VCA increasing the general  $-97\text{dBu}$  operation for a single channel to  $-103\text{dBu}$ . The only reason the ultra-low noise operation was not considered for NepTunes design, was that it reduces the available channels from 4 to 1.

While not entirely relevant, because there are similar operations capable from this IC, in the case that another component the team has chosen refuses to integrate correctly into the overall device, it would be possible to supplement with the SSI2164 given the correct configuration and integration. Having an IC on retainer that can achieve multiple relevant operations will be useful as a backup in the worst-case scenario of an inoperable mixer, or unconsidered complications with using a Quad VCA for housing all four of the voices present.

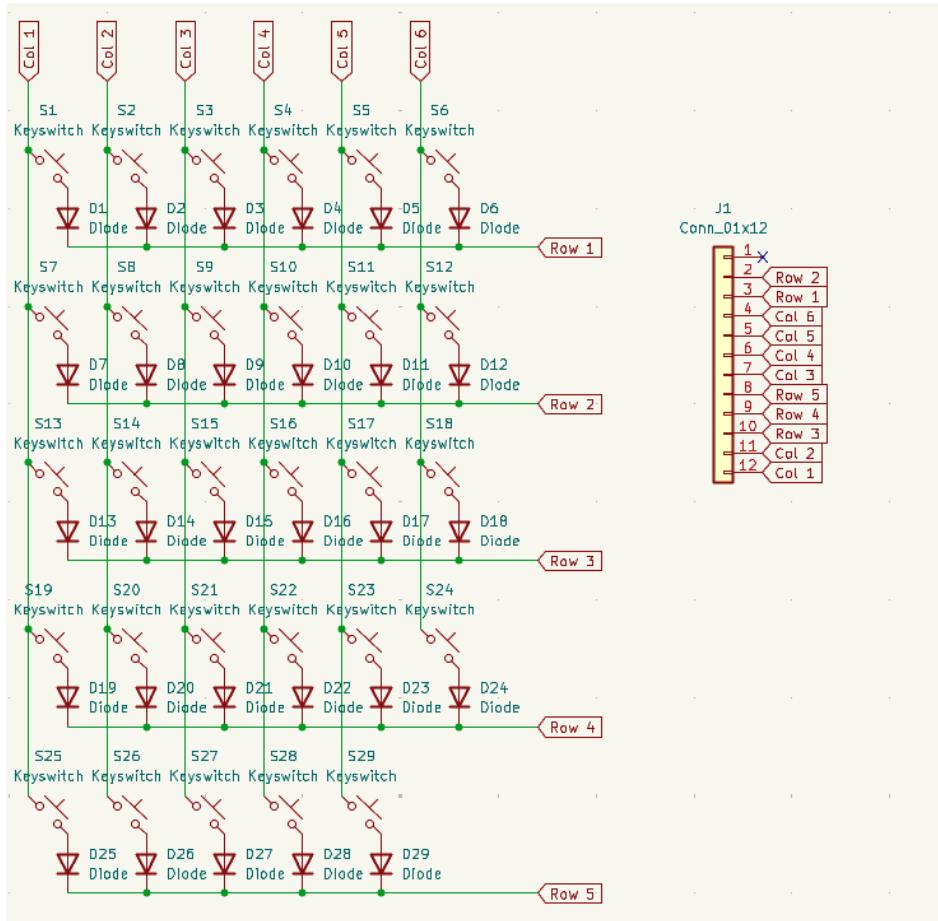
Important notes for implementation of the Mixer:

The SSI2190 mixer does not include many configurable features but does have a complex integration process which requires acute attention to detail. Because NepTune only supports four voices, and SSI2190 supports six voices, two missing voice connections need to be accounted for. Simply leaving no connection to these pins is unadvised as it can draw current and overheat the IC. In order to remove this risk, joining the positive and negative input pins of each voice would be required. The related control pins can be left grounded without interference.



**Figure 7.9:** Schematic of ADSR Envelope Generators (ENVGEN8) and Quad VCA IC (SSI2164) and the Mixer IC (SSI2190).

## 7.10 Keyboard Circuit



**Figure 7.10:** Schematic for keyboard matrix. This will be the MIDI keyboard on which the user plays.

## 7.11 Output Amplifier Circuit

The audio output circuit is important to this design. Careful attention must be made to design the output circuit so that the sound quality is not limited by audio output. To ensure that *NepTune* remains competitive with other synthesizers on the market, there will be two audio outputs: One headphone output, and one line-level output. The headphone output will enhance the synthesizer's portability, enabling the user to pull it straight out of their backpack and begin playing with little more than a pair of headphones. The line-level out will provide support for use in a studio environment or for performing in front of an audience. In either case, the sound quality must not be limited by these output options. The following section deals with the design of the output amplifier circuit.

### 7.11.1 Headphone Driver

The headphone driver circuit will be discussed first. Before designing it is necessary to start with a set of requirements to design for; however, designing for a specific range of headphones can be difficult, since there are little standards associated with headphones. For example, headphone impedances can range from  $8\Omega$ s to  $600\Omega$ s, making it difficult to design for the full range. The wide variety of headphone characteristics is illustrated in the chart below:

**Table 7.1: Sensitivity, Impedance, Peak Voltage, and Peak Current of Headphones (Permission has been requested for this table)**

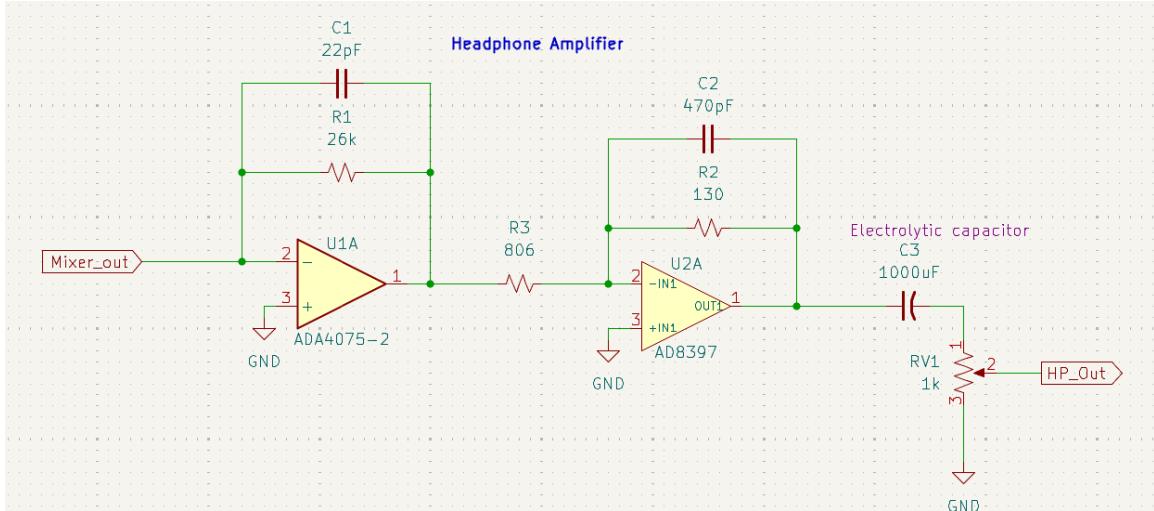
Manufacturer	Model	Sensitivity (dB/mW)	Impedance ( $\Omega$ )	Frequency (Hz)	Average Power (mW)	Peak Power (mW)	Peak Voltage (V)	Peak Current (mA)
SONY	XBA-4	108	8	3 to 28000	0.158	15.849	0.356	44.510
Audio-Technica	ATH-CHX7	100	16	15 to 22000	1.000	100.000	1.265	79.057
Shure	SE215	107	20	22 to 17500	0.200	19.953	0.632	31.585
Apple	Earpod	109	23	5 to 21000	0.126	12.589	0.538	23.396
Grado	Alice M1	100	32	20 to 22000	1.000	100.000	1.789	55.902
Creative	AURVANA Air	102	32	20 to 20000	0.631	63.096	1.421	44.404
KOSS	PP	101	60	10 to 25000	0.794	79.433	2.183	36.385
Sennheiser	HD650	98	300	10 to 39500	1.585	158.489	6.895	22.985
Beyer Dynamic	DT880	96	600	5 to 35000	2.512	251.189	12.277	20.461

As a result, it is necessary to choose a smaller range of headphones to target. Most consumers have headphones that are less than  $50\Omega$ s. This is a result of the rise in popularity of low-voltage devices such as smart phones and laptops. These low voltage devices work best to drive low impedance devices ranging from  $8\Omega$ s to  $32\Omega$ s [47]. Therefore, it makes sense to target a load of  $32\Omega$ s in this design. In addition, the peak power for devices in this range can reach as high as 100mW so a circuit with a higher power tolerance will be required. To keep a safety buffer, *NepTune* will incorporate a 200mW peak power capability. The voltage output of the circuit should also be  $\sim 2.5V_p$  (1.77Vrms). In addition, headphones are not purely resistive. They contain some reactive component. As a result, the output amplifier should be able to handle loads that have capacitances as high as 400pF.

The design criteria, which also pulls from the engineering specifications (Table 1.1), is outlined in the table below:

**Table 7.2: Headphone Driver Design Criteria**

Voltage out (headphone out)	~2.5 Vp	Voltage output for headphone jack (Vp).
Output Impedance (headphone out)	8Ωs-32Ωs	Output impedance for headphone jack.
Peak output power	200mW	Power output that headphone jack should be capable of.
Capable Capacitance load	400pF	Capacitive load that headphone jack should be capable of supplying.



**Figure 7.11: Headphone driver circuit.**

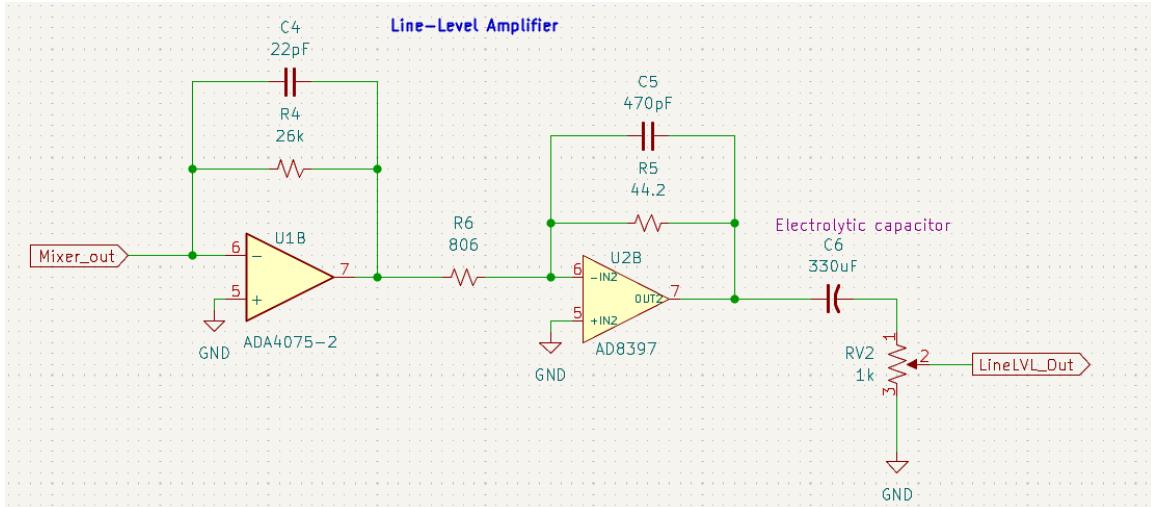
The headphone amplifier circuit above consists of two stages. The first stage is the transimpedance amplifier and is known as the I-V (current-to-voltage) stage. It takes the output current that the preceding mixer produces and converts it into a voltage. In the configuration shown in Figure 7.2, it is set up to provide unity gain and has a transfer function  $-R_1$  [46]. It produce an output voltage that swings between  $\pm 5V_p$  in order to keep with the signal path convention. The I-V amplifier fits well in this location because the preceding mixer stage requires a virtual ground immediately at the output (in order to supply a stable current).

The second stage in Figure 7.2 is the headphone driver and is what converts the voltage into the  $2.5V_{pp}$  that is required for the load. The second stage is set up to provide 0.25 gain to bring the signal down from a  $\pm 5V_p$  swing. It is also a low-noise op-amp that is capable of supplying 200mW of power to the output if necessary. The  $1000\mu F$  capacitor is

electrolytic and serves to block DC currents to protect against short-circuit conditions. Short circuit conditions may occur when the 1/4" jack is improperly inserted.

### 7.11.2 Line-Level Output Driver

There is an option between professional grade (+4dBu) 0.775Vrms or (-10dBV) 0.446Vrms. NepTune will be -10dBV since this product caters to the introductory/novice audience who will typically not have access to professional grade equipment.



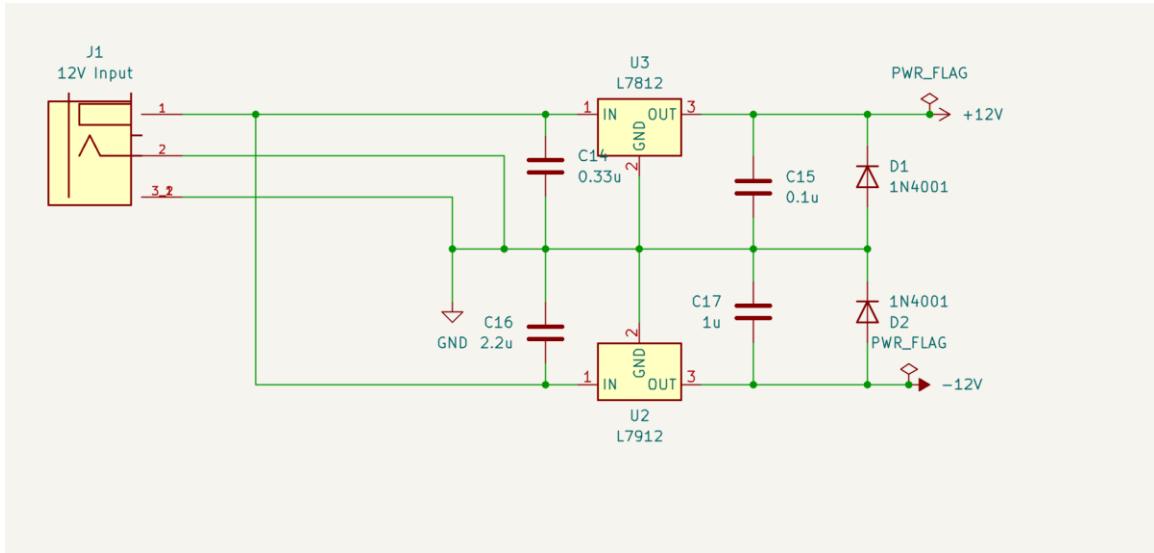
**Figure 7.12:** Line-level driver circuit.

The 330uF capacitor is electrolytic and serves to block DC currents to protect against short-circuit conditions. Short circuit conditions may occur when the 1/4" jack is improperly inserted.

## 7.12 Power Circuit

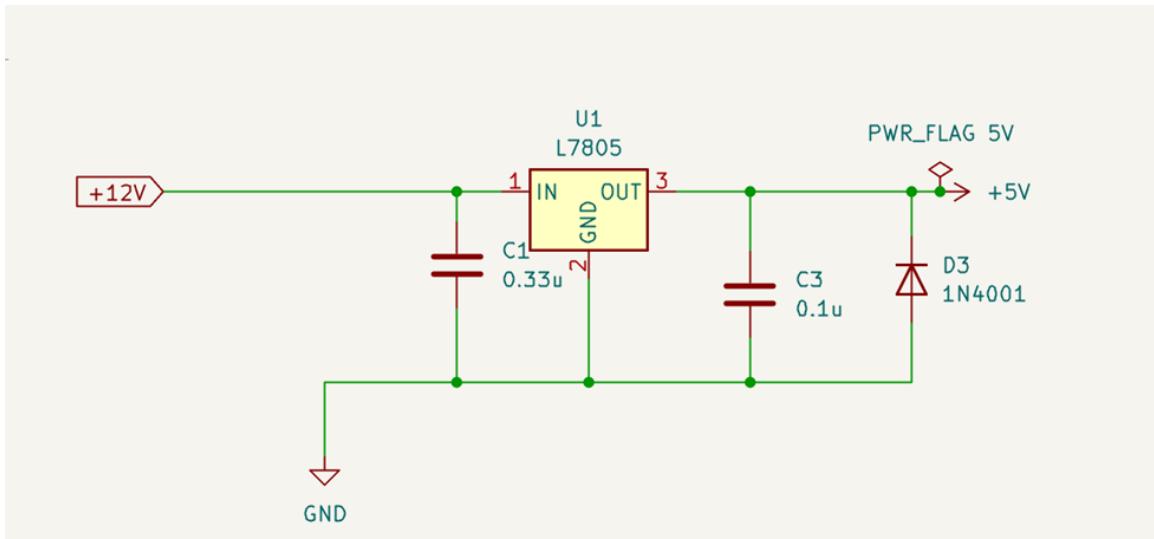
The power circuit design was difficult due to the separation of ground that needs to take place. However, initially the design begins with the 15V input through a barrel jack. The barrel jack that was chosen is an AC/DC source meaning that it takes the power sourced from the wall wart that is in AC form and converts it to a DC power source. The current that is generated by the power source then passes through a linear load regulator that ensures that cleans the signal and allows for a steady 15V that is stepped down to 12V to be outputted to the board. Since this is an analog source its important to have both +12V and -12V as outputs. This output is then past to multiple of the analog ICs as described below. The ground that connects to these regulators will be separated from the digital ICs grounds.

### 7.12.1 Analog Board Power Circuit



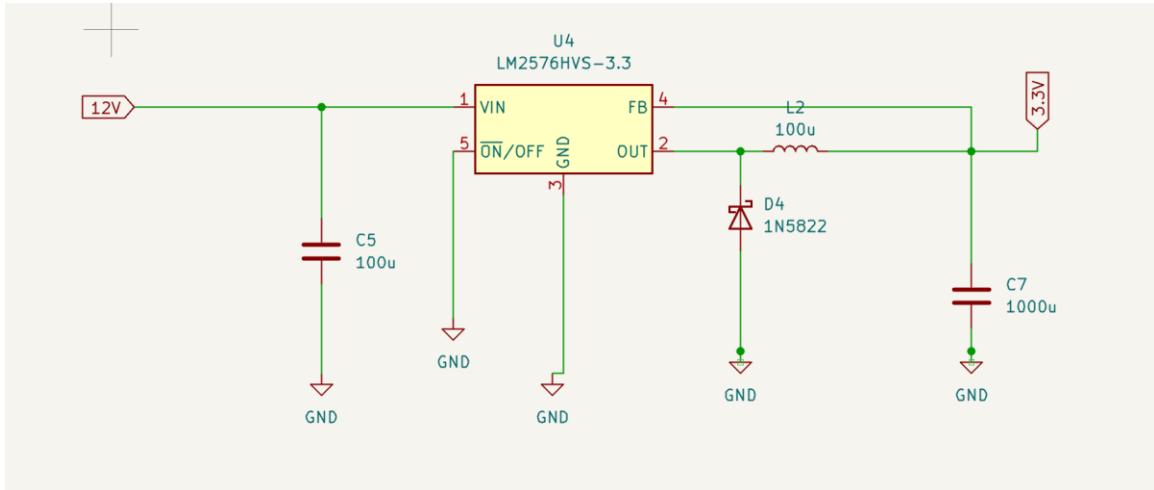
**Figure 7.13:** +12V and -12V Input Voltage Regulator

The same concept is applied for the 5V regulator as it is mainly used for the digital portion of the PCB. The 5V regulator is also utilized for analog circuits and therefore the grounds between the 12V and the 5V are connected, and the grounds between the two 5V regulators is also connected. There are decoupling capacitors present for both power supplies, and a diode to ensure no current flows in the wrong direction.



**Figure 7.14:** +5V and -5V Regulator

### 7.12.1 Digital Board Power Circuit



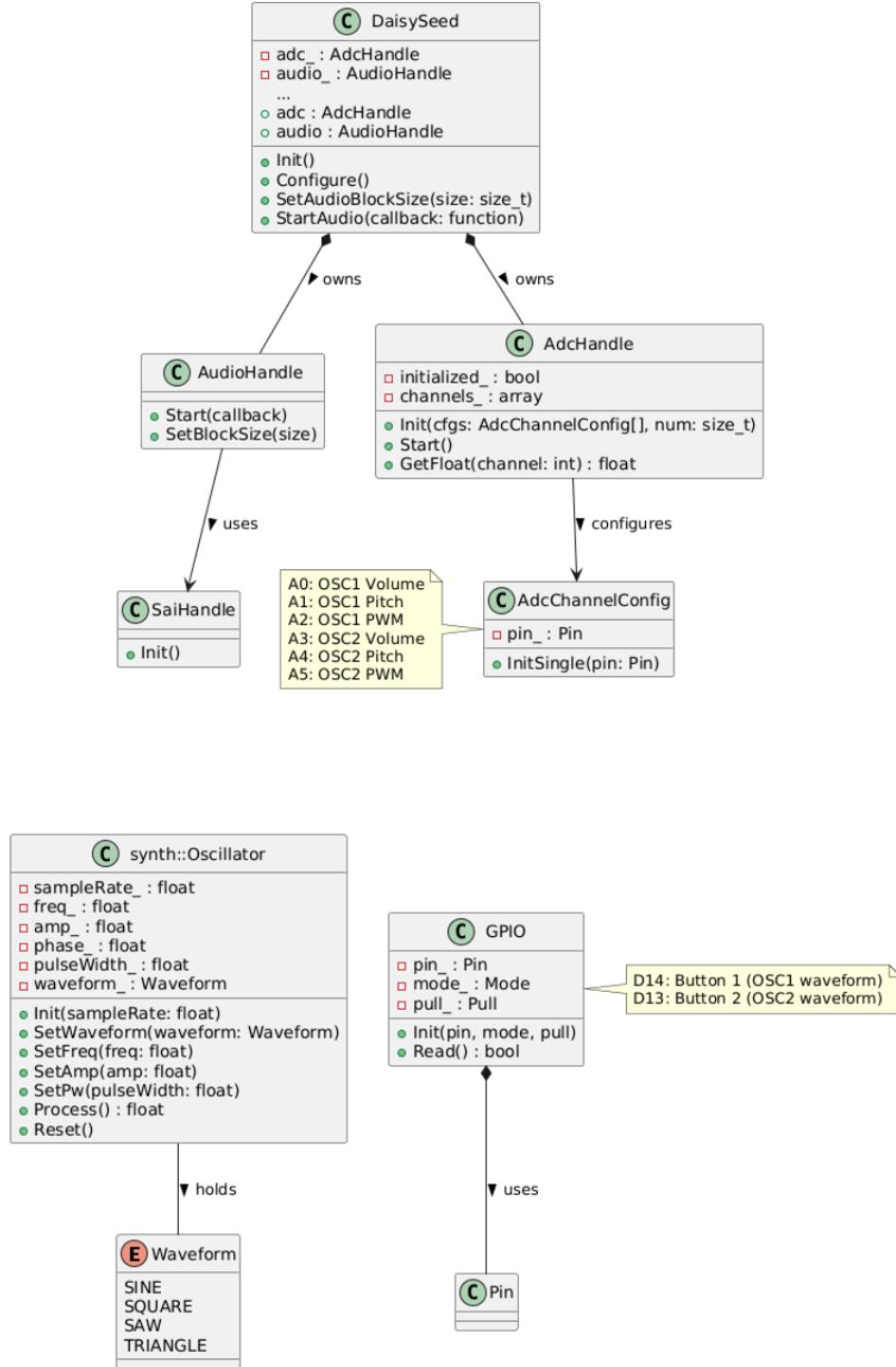
**Figure 7.15:** Power circuit schematic.

The digital board uses mostly 3.3V connection to all the parts, therefore, a switching regulator is used to power this portion of the board. This is also a separate ground from the analog circuit to limit noise.

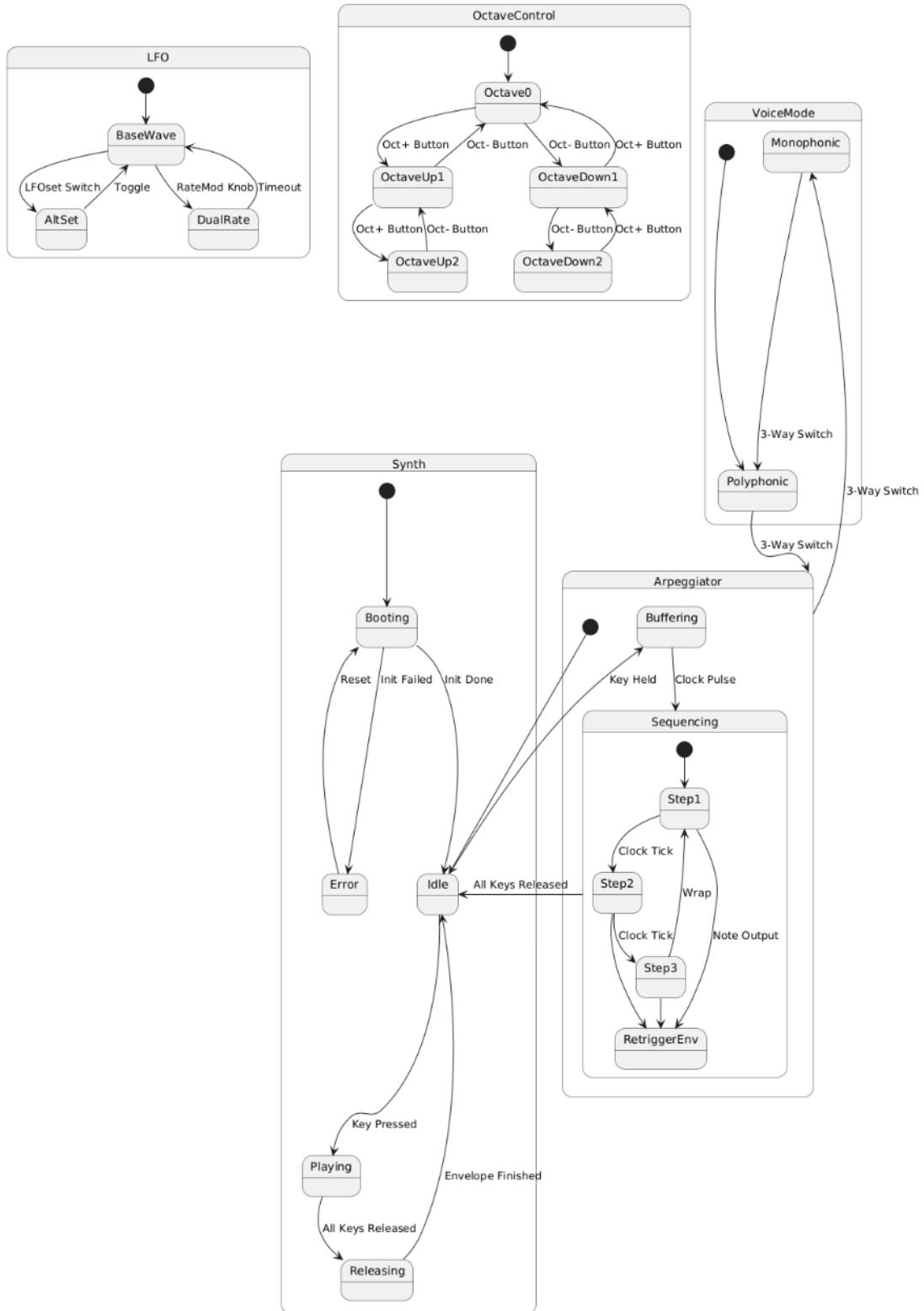
## Chapter 8 – Software Design (Senior Design 2 Revised)

The NepTune synthesizer's system behavior and architecture are modeled using several complementary diagrams that collectively capture its overall design. The class diagram defines the software architecture and logic of core components like oscillators, envelope generators, filters, and hardware interfaces. The state diagrams describe runtime behavior and transitions between system states such as booting, playing, or releasing, while also detailing subsystem-specific logic like voice mode switching, LFO waveform cycling, and envelope progression. A use case diagram outlines how the user interacts with the system, identifying primary actions such as playing notes, designing patches, and modulating parameters, with clear inclusion and extension relationships. The module interconnection diagram visualizes how all subsystems (oscillators, filters, envelopes, LFOs, and clock generators) exchange control and signal data, making dependencies explicit. Supporting this is a detailed control and module summary table, listing all user-facing controls (knobs, switches, buttons) with their associated functions and subsystem mappings. Together, these

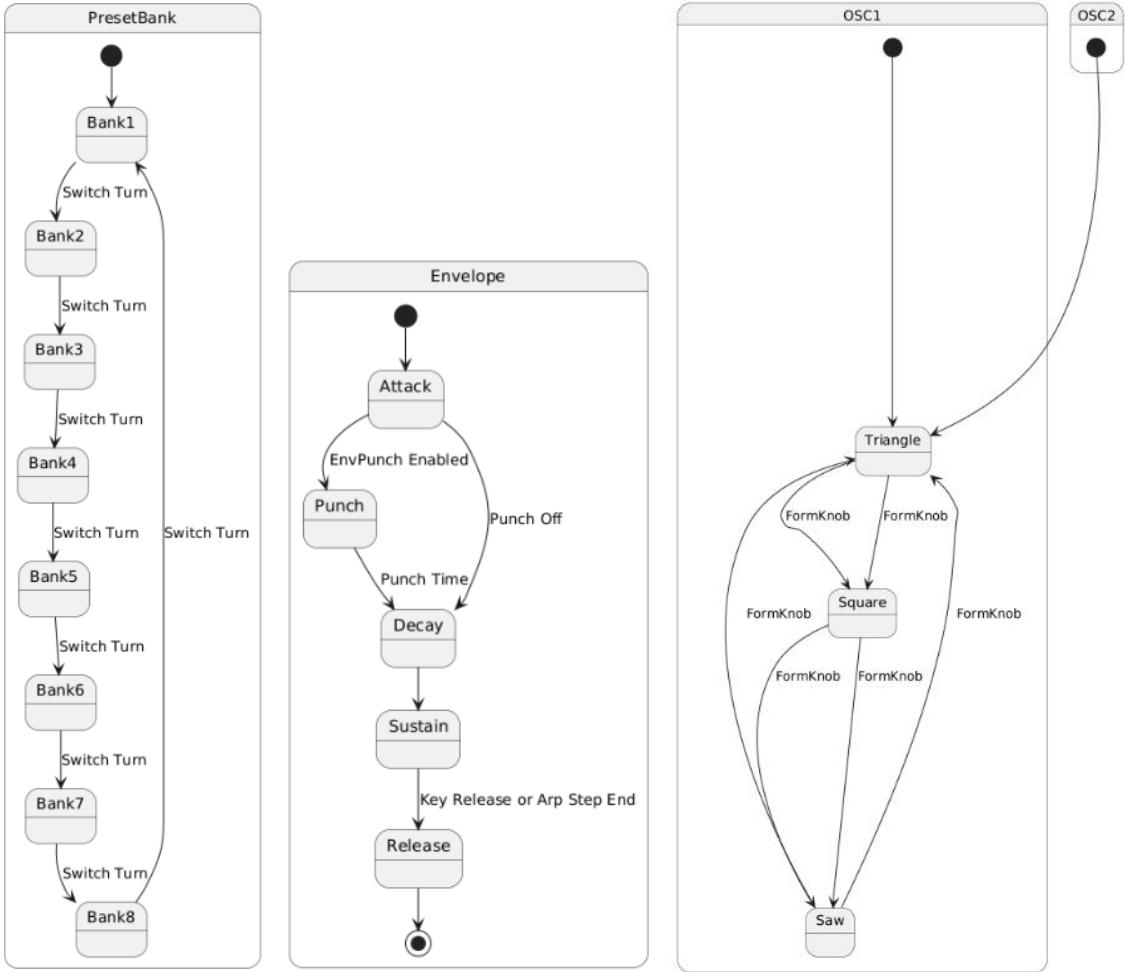
diagrams provide a holistic view of NepTune's firmware design, user experience flow, and audio processing architecture.



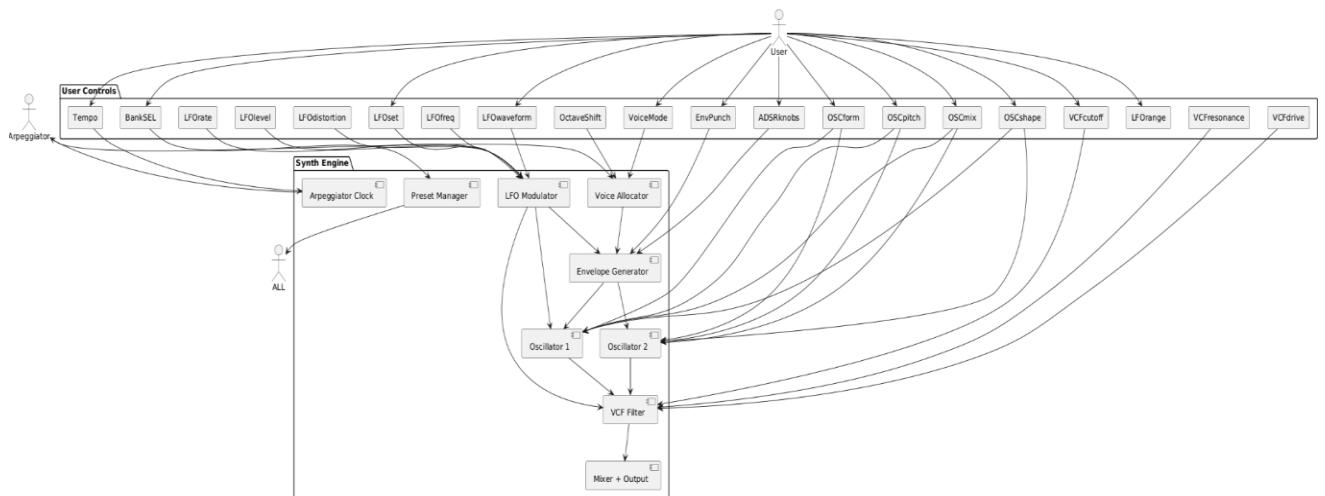
**Figure 8.1:** Embedded software class diagram of NepTune.



**Figure 8.2:** Separate state diagrams of LFO manipulation, octave control system, VoiceMode switch, arpeggiator, and state of keys pressed on synth with arpeggiator.



**Figure 8.3:** Remainder state diagrams of the 8 presets, envelope generator (ADSR), and waveform cycle of the 2 oscillators.

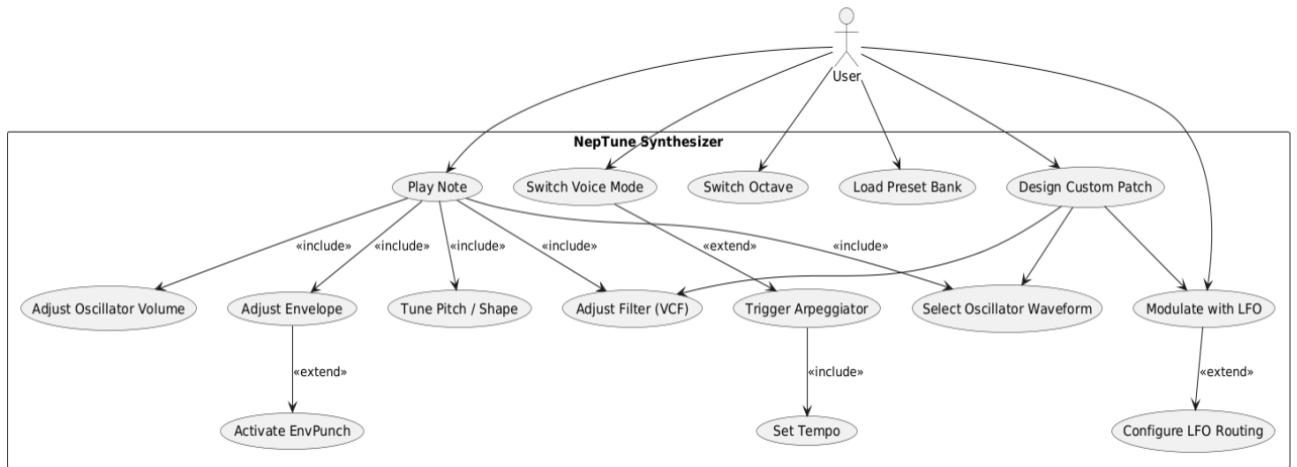


**Figure 8.4:** NepTune synthesizer module interconnection diagram.

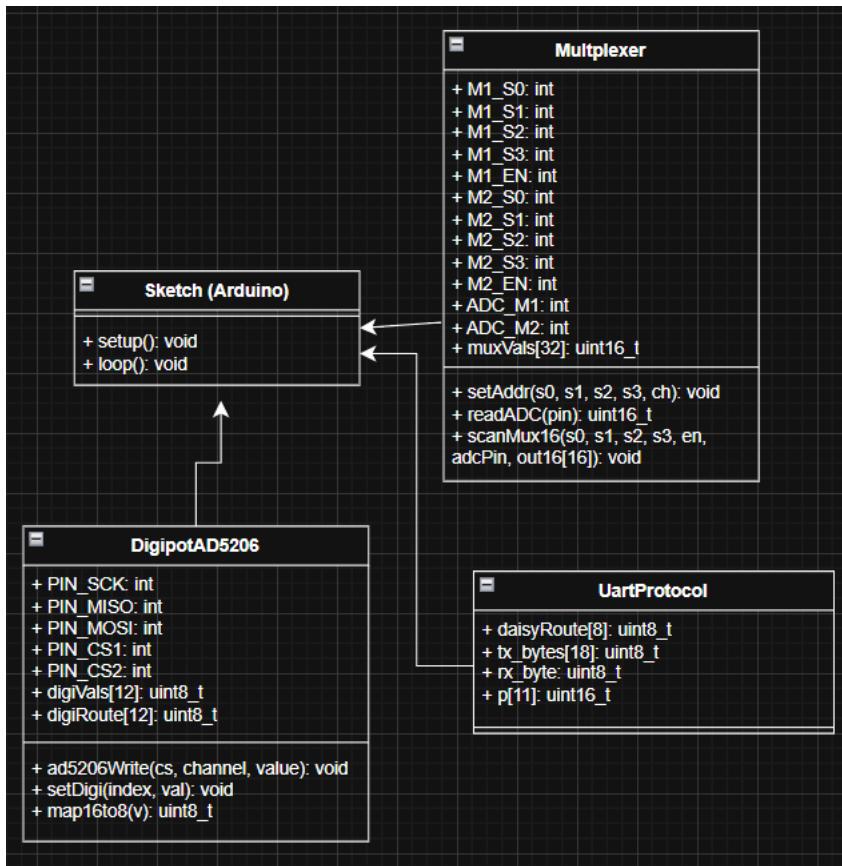
**Table 8.1: Control and Module Summary Table**

Name	Type	Subsystem	Description
OSCpitch	Knob	Oscillator	Controls pitch of both oscillators
OSCform	Knob	Oscillator	Selects waveform (Triangle, Saw, Square)
OSCshape	Knob	Oscillator	Alters oscillator shape/modulation
OSCmix	Knob	Oscillator	Controls amplitude mix level
VCFcutoff	Knob	Filter (VCF)	Sets low-pass filter cutoff frequency
VCFresonance	Knob	Filter (VCF)	Boosts frequencies near the cutoff point
VCFdrive	Knob	Filter (VCF)	Adds gain/distortion after the filter
LFOfreq	Knob	LFO	Sets the base modulation frequency
LFOOrange	Switch (4x)	LFO	Selects LFO frequency range
LFOwaveform	Switch (8x)	LFO	Selects waveform for LFO modulation
LFOset	Switch (2x)	LFO	Switch between LFO waveform banks
LForate	Knob	LFO	Speed of morphing between LFO waveforms
LFOlevel	Knob	LFO	Controls modulation intensity
LFOdistortion	Knob	LFO	Adds distortion to the LFO signal

VoiceMode	Switch (3x)	Voice Engine	Selects between Polyphonic, Mono, or Arpeggiator voice handling
Tempo	Knob	Clock	Sets tempo of the arpeggiator (clock ticks)
EnvPunch	Button	Envelope Generator	Enables extra transient boost during attack phase
ADSRknobs	Knob Group	Envelope Generator	Sets Attack, Decay, Sustain, Release phases
OctaveShift	Button (+/-)	Voice Engine	Transposes key input by $\pm 1$ or 2 octaves
BankSEL	Switch (8x)	Preset Manager	Selects one of 8 saved sound patches
Voice Allocator	Engine Unit	Synth Engine	Routes notes to envelopes/oscillators based on voice mode
Envelope Generator	Module	Synth Engine	Shapes volume/frequency of notes via ADSR
Oscillator 1/2	Module	Synth Engine	Generates sound waves per note (shaped by LFO & ENV)
VCF Filter	Module	Synth Engine	Processes signal with filtering (modulated by knobs & LFO)
LFO Modulator	Module	Synth Engine	Sends low-frequency modulation to pitch, filter, or amp
Arpeggiator Clock	Module	Synth Engine	Generates rhythmic steps in arpeggiator mode
Preset Manager	Module	Synth Engine	Loads and applies preset values to all subsystems



**Figure 8.5:** NepTune Use Case Diagram



**Figure 8.6:** ESP32 class diagram (Senior Design 2)

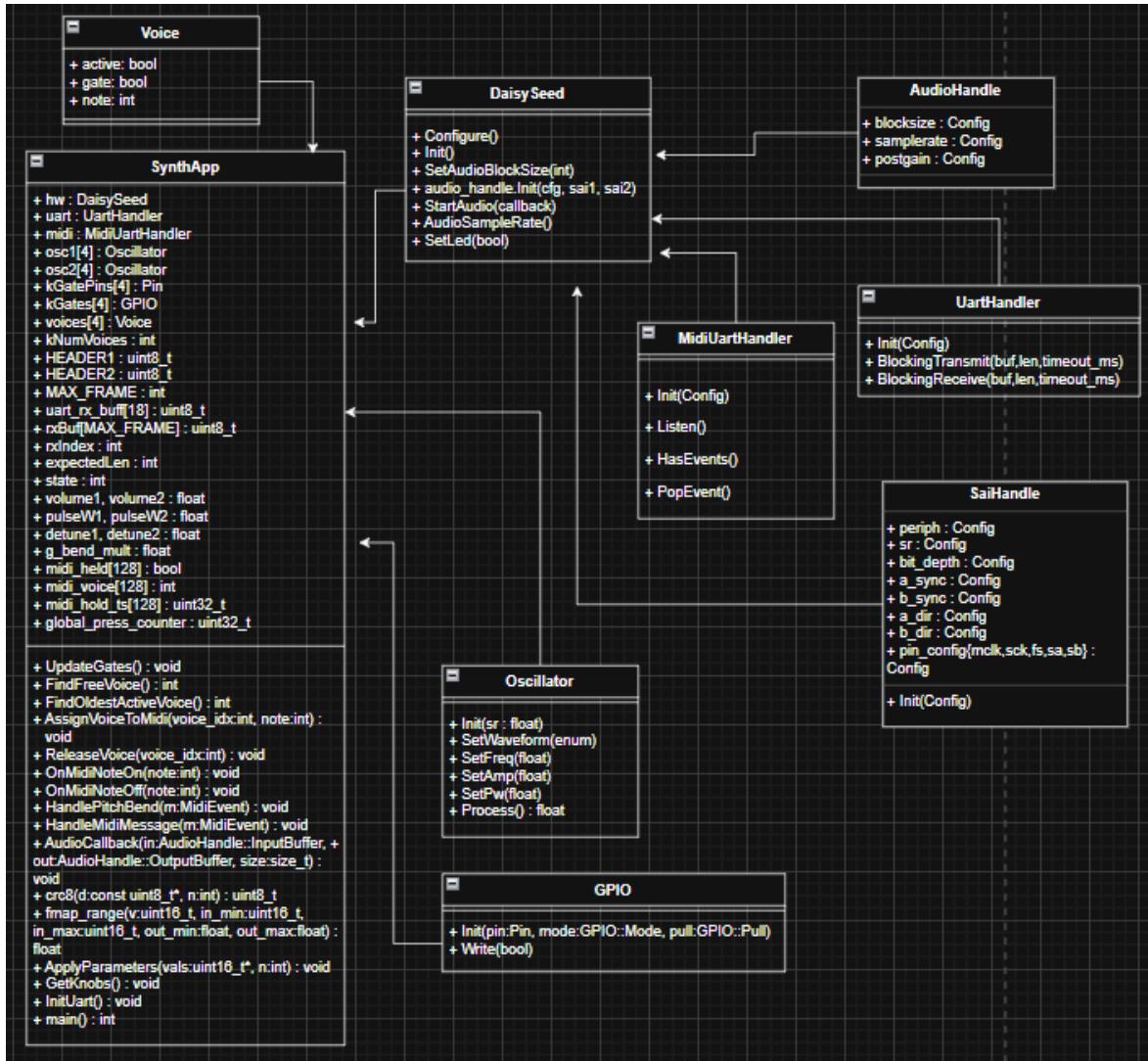


Figure 8.7: DaisySeed/STM32 class diagram (Senior Design 2)

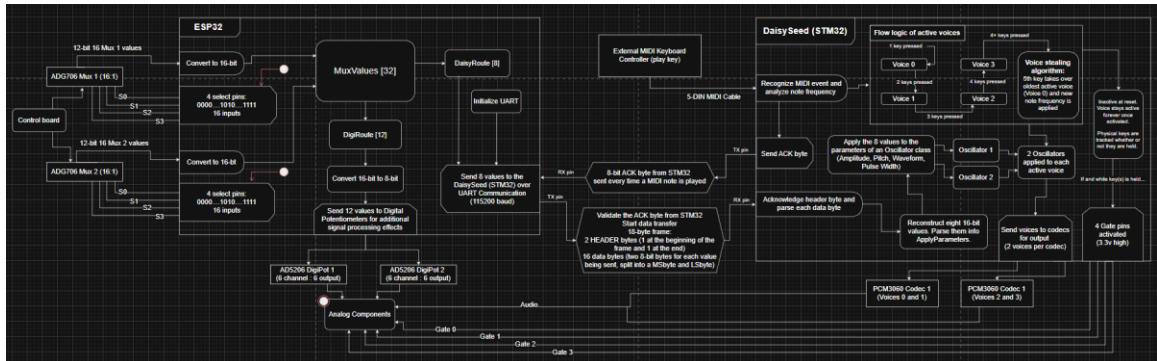
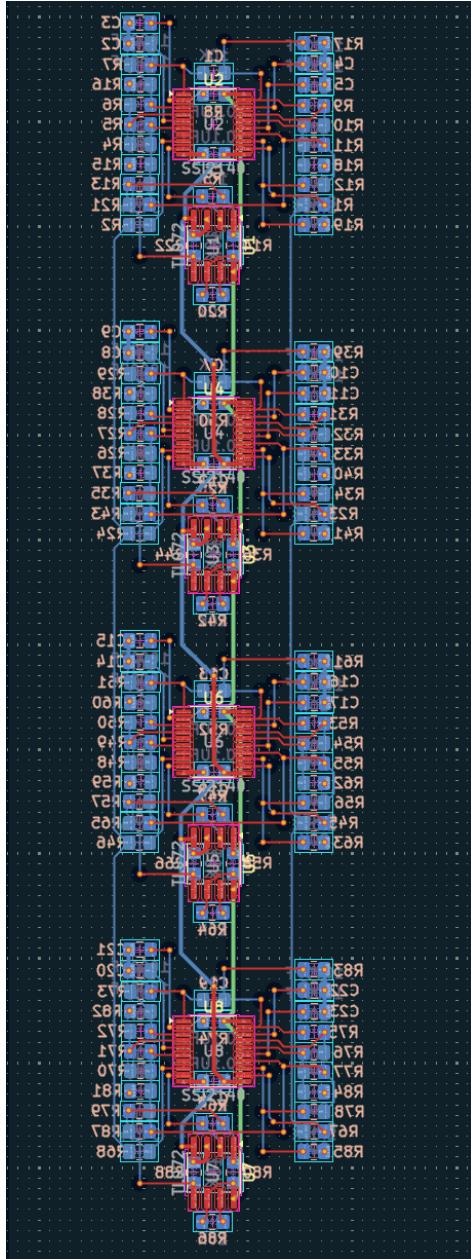


Figure 8.8: Hardware to Software State (functionality) diagram (Senior Design 2). Zoom in for full view.

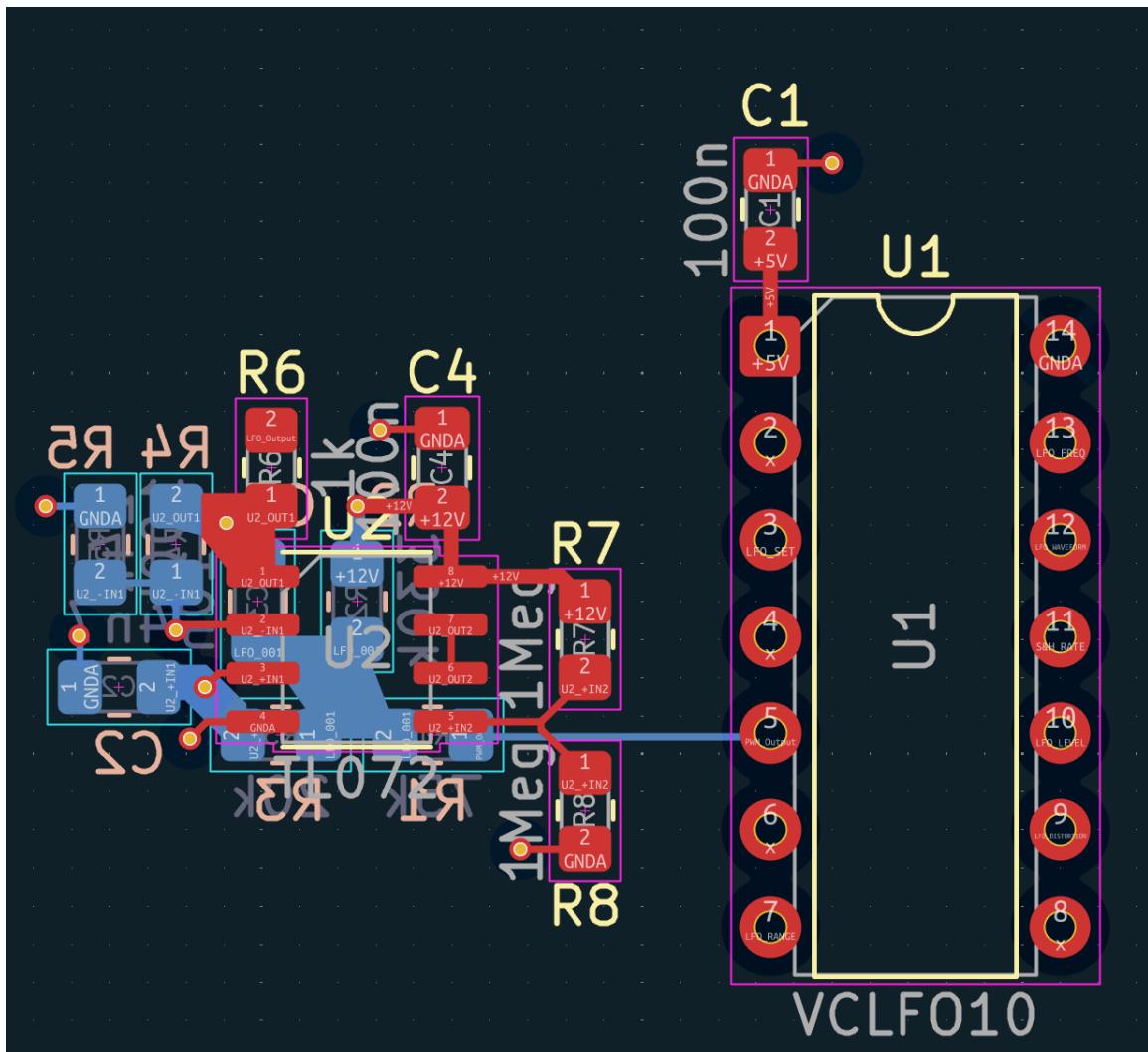
# Chapter 9 – System Fabrication & Prototype Construction

## 9.1 Printed Circuit Board (PCB) Layouts

### 9.1.1 VCF PCB

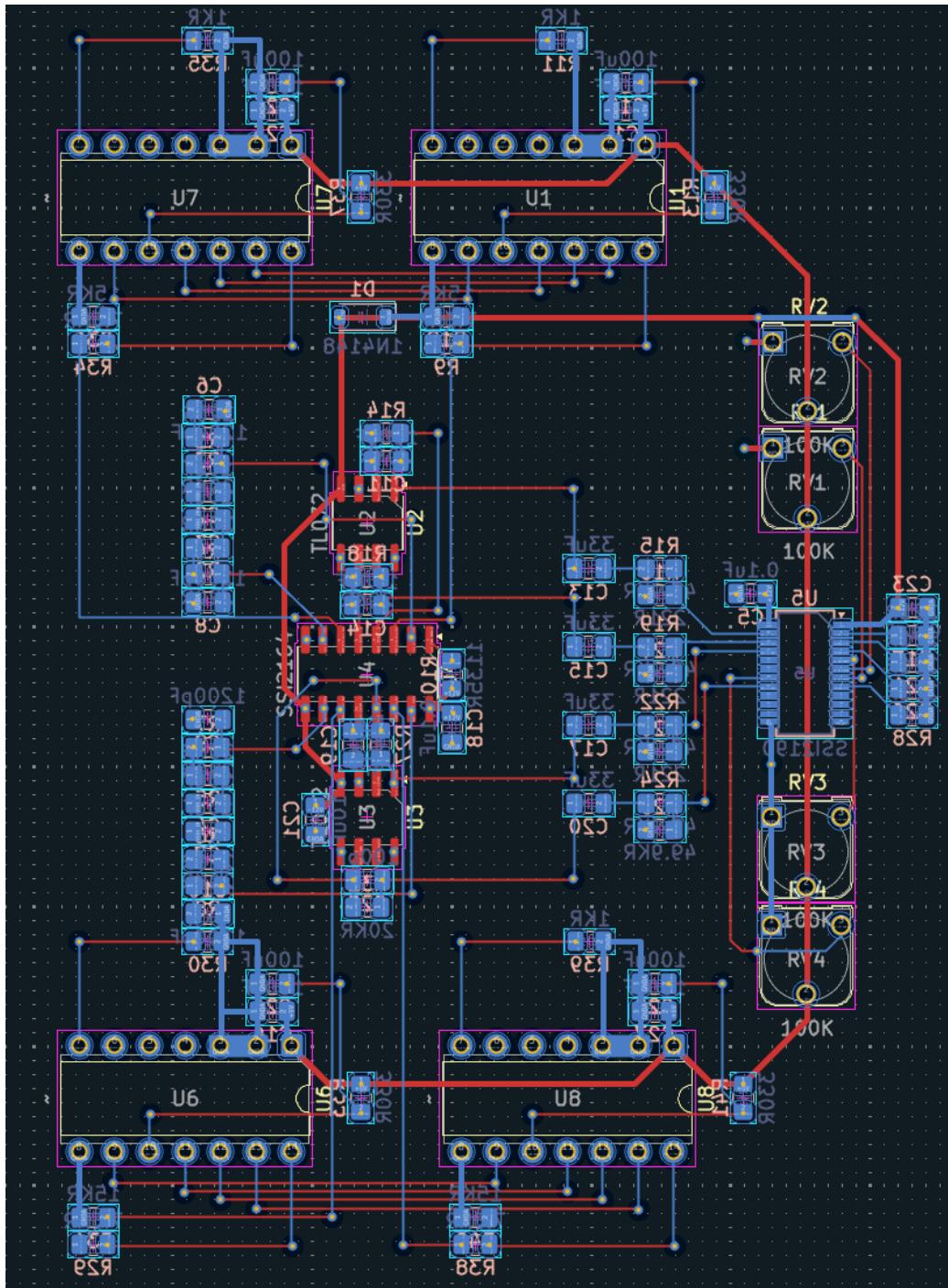


### 9.1.2 LFO PCB



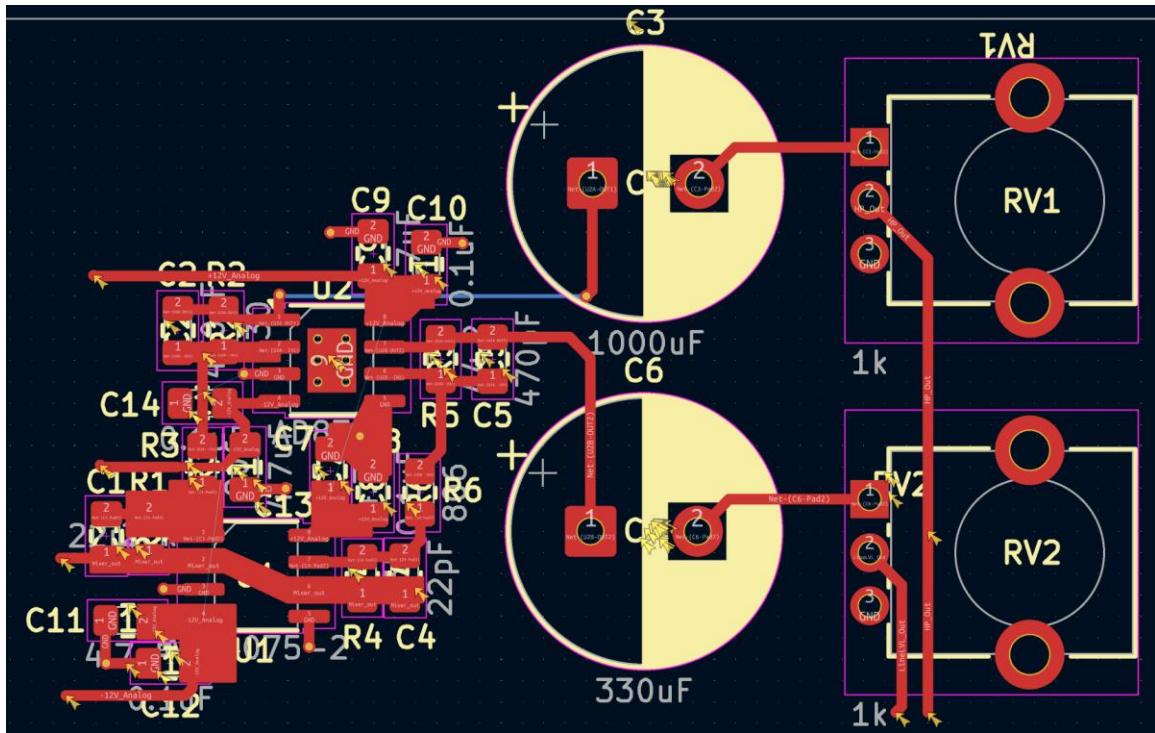
**Figure 9.2:** PCB layout containing the LFO circuit. U1 is Electric Druid's VCLFO10, and U2 is the TL072 which filters the output of the LFO.

### 9.1.3 VCA, Envelope Generator, and Mixer PCB



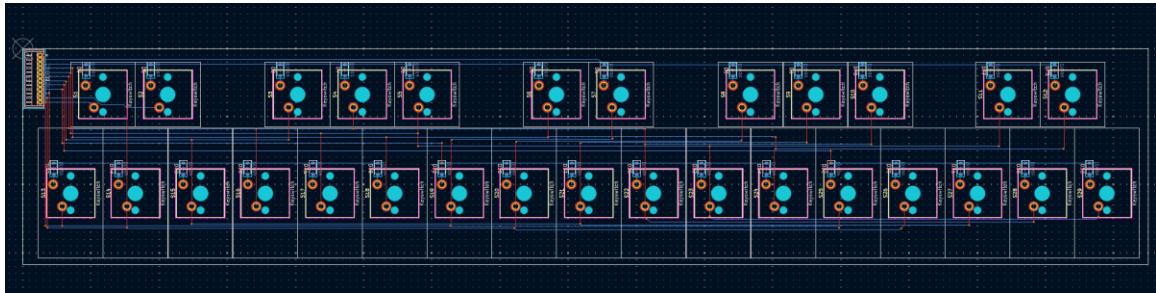
**Figure 9.3:** PCB layout containing the VCA and ADSR envelope generators for each of the four voices. The mixer which combines the four voices into one signal is also included in this PCB layout.

#### 9.1.4 Audio Output PCB



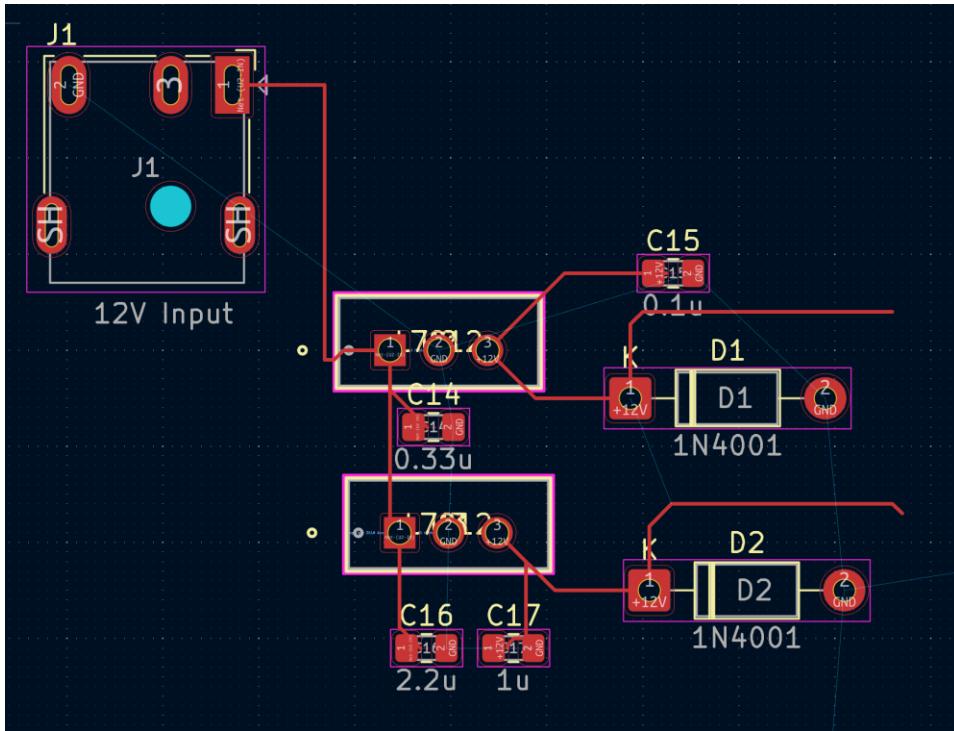
**Figure 9.4:** Output amplifier circuit. This layout contains the amplifier circuits for both the line-level and headphone outputs.

### 9.1.5 Keyboard PCB

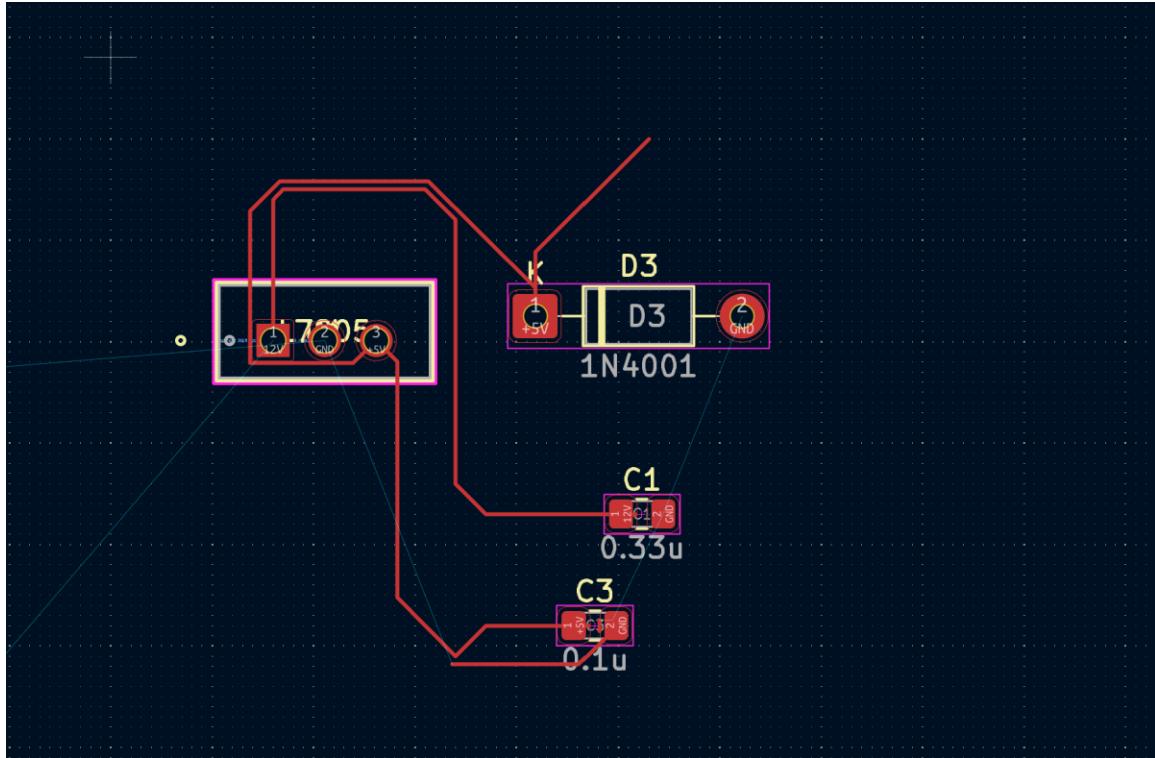


**Figure 9.5:** MIDI Keyboard PCB. This is the layout for the keyboard the user will play on.

### 9.1.6 Analog Power PCB

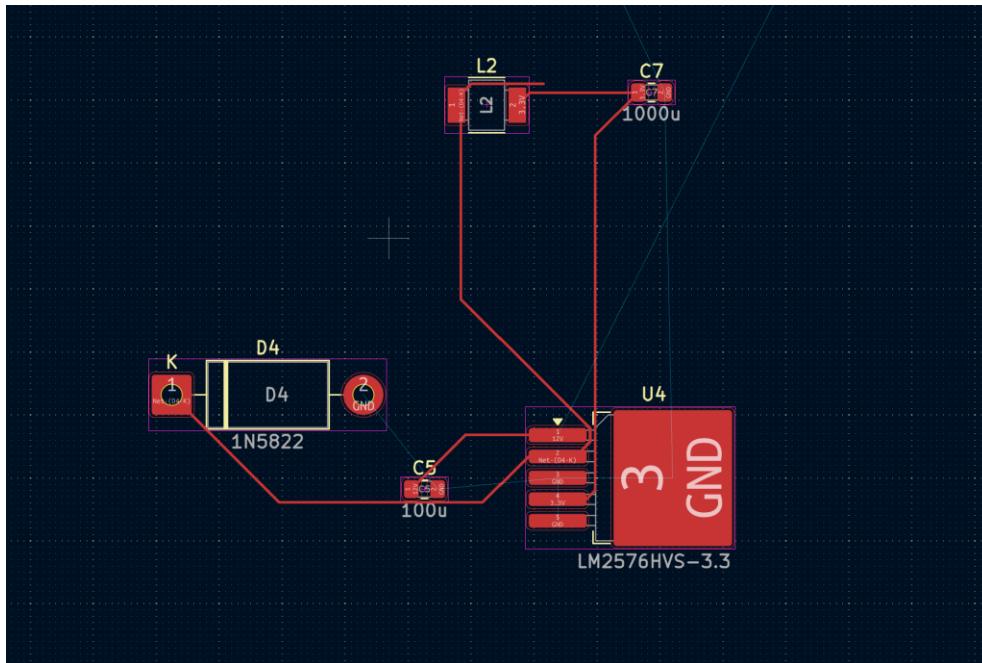


**Figure 9.6:** -12V Analog Power to be connected to ICs



*Figure 9.7: -5V Analog Power to be connected to ICs*

### 9.1.7 Digital Power PCB



*Figure 9.8: -5V Analog Power not connected to ICs*

# Chapter 10 – System Testing and Evaluation

## 10.1 Hardware Testing

Hardware testing is heavily based around constructing circuits on breadboards. There are various pitfalls surrounding the construction of audio circuits on breadboards. Typically, the jumper wires are highly susceptible to noise, which makes debugging circuits difficult.

### 10.1.1 Daisy Seed

In order to test that the Daisy Seed development boards were working correctly, they were plugged into the local device and confirmed to have their power LED on. Upon pressing the boot button and reset buttons at the same time, they should switch to DFU boot mode. In this mode, they were recognized by the local devices to be available for programming. GPIO pin voltages were validated through measurements with a digital multimeter.

### 10.1.2 SSI2140

In order to test the SSI2140's filtering capabilities, a simple 1<sup>st</sup>-order lowpass filter was implemented according to the datasheet. A 0.5Vpp 1kHz sine wave was sent to the input of the filter, and the cutoff frequency was controlled via the EXPO\_CTRL pin. Upon successfully controlling the cutoff frequency of signal from the output pin and monitoring it on an oscilloscope, the SSI2140 was confirmed functional.

### 10.1.3 SSI2164

The SSI2164 VCA will be tested by observing the gain of each signal entered. This gain can be configured based on the necessary value required by the NepTune synthesizer. It will be required for the testing team to observe not only the operating behavior of the chip when connected to the VCF but also when modulated by the ADSR envelope generator, and when operating with multiple signals being amplified at once because of the SSI2164's quad amplifier capabilities in which NepTune exploits.

Additionally, the required operational amplifiers, the TL072 will be required for testing as the signal will be better modulated after the output of the operation amplifier.

### 10.1.4 SSI2190

The SSI2190 audio mixer will be tested to see the ability to mix the four voices of the NepTune board. Being able to use potentiometers to test the volume configuration that should be present to the user is also required. Proper sound should be outputted by the chip meaning that audio interface will also be present for testing. There are numerous discrete components that will be implemented alongside the IC for breadboard testing.

### 10.1.5 ENVGEN 8D

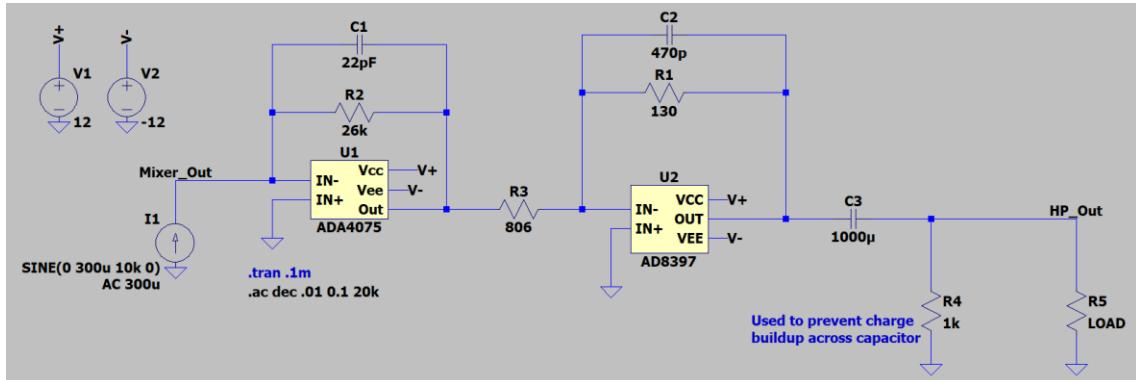
The Envgen8 will require four potentiometers along with an output coming from the VCF so that modulation can be observed on an oscilloscope. Changing the values to match the behavior expected will be the major issue for testing. Connecting all other discrete components as described by the schematics of Envgen8's hardware design will also be required and possible via lab resources. This can be tested with a simple sine sound signal entering and as testers press the key for sound to be generated, other members can examine the waveform to observe the matching Attack, Decay, Sustain and Release waveform behaviors.

### 10.1.6 VCLFO 10

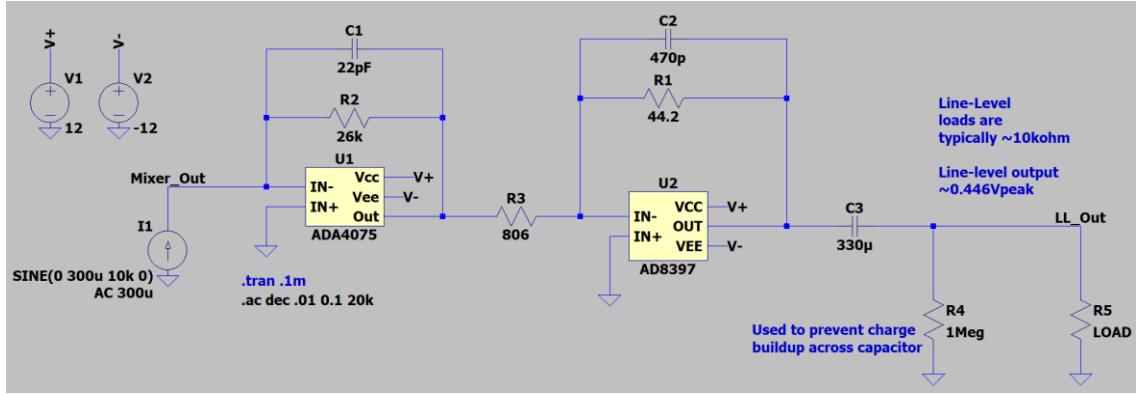
The LFO circuit has went through various tests on a breadboard. No PCB design tests could be made yet. However, construction on a breadboard has yielded promising results. The circuit functions well off of a 5V switching power supply. The output filters successfully extract the intended waveform from the PWM signal. The complete output waveform is a strong 5V signal, which is sufficient to drive the VCF control voltages.

### 10.1.7 Audio Output Amplifier

Since the audio output amplifier is constructed from discrete components from Analog Devices, a circuit simulation on both output circuits can be performed via LTspice. Analog Devices often makes SPICE models for their devices readily available on the product page, which greatly accelerates prototyping. The circuits used in the simulation are shown in the following figures:



(a)



(b)

**Figure 10.1:** LTspice simulation schematic of audio output amplifier. (a) Headphone amplifier circuit. (b) Line-level output circuit.

In both simulations, the load is modeled as the resistor, R5, denoted as “LOAD”. The headphone amplifier circuit is shown in Figure 10.1 (a). The simulation was ran with the headphone modeled as both an  $8\Omega$  and a  $32\Omega$  load. With the load as  $8\Omega$ , the cutoff frequency is 20Hz, providing acceptable gain for low frequencies. For  $32\Omega$  headphones, the response is much greater, offering a cutoff frequency at 5Hz. The line-level amplifier circuit is shown in Figure 10.1 (b). The load is modeled as a  $10k\Omega$  resistor and the low frequency response is excellent, offering a cutoff frequency at 0.048Hz. Thus, the circuit performs best when connected to external audio equipment.

## 10.2 Software Testing

To test the capabilities of the Daisy Seed, the GitHub repository was cloned onto the local device and the example code was modified in various ways to adjust the different sounds that could be made. Testing of code, writing of code, and building of firmware was done in Visual Studio Code utilizing the Cortex-Debug extension (see appendix Figure E.1).

# Chapter 11 – Administrative Content

As with any group project, successful execution depends on administrative planning. While these considerations may not directly influence the features of *NepTune*, they are essential to the project's success. It is necessary for the team to agree on the expectations of each member early on in the project to eliminate any surprises or confusion. These expectations include budget, timelines, task distribution, and team organization.

## 11.1 Budget and Financing

While the goal of *NepTune* is to produce a low-cost device, there will inevitably be unexpected and additional costs that come as a result of prototyping. One of the goals for *NepTune* is to produce a final product from parts costing less than \$500. However, this price does not include the cost of prototype circuit boards, circuit components, and equipment that will need to be bought in order to complete the project. As a result, the team expects to spend a bit more.

*Table 11.1: Budget distribution*

Item	Description	Budget allocation
Daisy Seed	Daisy Seed development board by Electrosmith. This will be used to accelerate prototyping.	\$65
Keyboard	Keyboard keys and switches.	\$60
Buttons & Knobs	Buttons and potentiometers the user needs to interact with the synthesizer.	\$15
ICs	Sound processing ICs, power ICs, external DACs, memory, MCUs, and more.	\$280
Discrete components	Miscellaneous components such resistors, capacitors, and inductors.	\$40
PCBs	All PCBs required for this build.	\$125
ST Links	Required for programming the STM32.	\$20

Screen	Screen and related hardware.	\$35
Other	Miscellaneous and unexpected costs.	\$110
<b>TOTAL:</b>		<b>\$750</b>

## 11.2 Project milestones for SD1 and SD2

When defining the goals of the project, one of the key considerations was the limited time available. This senior design project is being completed over the Summer and Fall semesters, which together represent one of the shortest possible timelines. The Summer term is especially compressed, typically four weeks shorter than a standard semester, with no extended breaks or holidays to leverage. Although the Fall semester includes several breaks, they often coincide with major holidays, making it difficult to dedicate time to the project. Furthermore, choosing the Summer–Fall schedule eliminates the opportunity to use the month-long winter break for additional work. Given these constraints, it was essential to set realistic project goals from the beginning.

This first table lists the Senior Design I milestones. The timeline has been organized in two sections. Table 11.2 lists the project and team milestones, while Table 11.3 lists the deadlines for deliverables.

The project milestones below have been organized to reflect an achievable timeline. The timelines are tight, and it will be very easy to fall behind, but the team is determined to meet these deadlines.

*Table 11.2: Senior Design I Project and Team Milestones*

Time	Date	Task
	5/02/25	Classes begin (Summer Sem.)
1 week	5/12/25 – 5/17/25	Recruitment and Introduction to idea
2 weeks	5/18/25 – 5/31/25	Research, project refinement, task distribution
3 weeks	6/01/25 – 6/21/25	Individual and collaborative research and start of design
2 weeks	6/22/25 – 7/05/25	Design finalization
1 week	7/06/25 – 7/12/25	Remaining device & component acquisition
2 weeks	7/13/25 – 7/26/25	Prototyping & debugging

3 days	7/27/25 – 7/29/25	Finalizing prototype for presentation
--------	-------------------	---------------------------------------

Similar to the project milestones, the deliverable deadlines are also extremely tight. However, unlike internal deadlines, these are non-negotiable and not determined by the team. As a result, there is little flexibility and the team must not let deadlines creep up. The deadlines are shown below:

*Table 11.3: Senior Design I Deliverable Deadlines*

Date	Task
5/16/25	Project proposal documents and team organization
5/30/25	Divide and Conquer report
6/06/25	Divide and Conquer report revision
7/07/25	100-page report
7/18/25	100-page report revision
7/29/25	150-page report
7/29/25	Final demonstration video

To meet the 150-page report requirement, the team will maintain a continuous writing process, even in the absence of immediate feedback. While feedback is valuable, the compressed timeline demands steady forward momentum throughout the semester. This is especially true for the time period after the 100-page deadline, where the team only has three weeks to write the remaining 50 pages, complete a prototype, and produce a demonstration video.

The following table illustrates the milestones for Senior Design II. While it is very early and deadlines for the second semester are tentative, creating a schedule helps keep the team on track.

**Table 11.4: Senior Design II Milestones**

Time	Date	Task
2 weeks	7/30/25 – 8/18/25	Continue PCB design
	8/18/25	Classes Begin (Fall Sem.)
	8/18/25	Finalize and order PCBs
2 weeks	8/18/25 – 8/30/25	Build case & keys & screen
10 weeks	8/31/25 – 11/01/25	Test fit & debug
3 weeks	11/02/25 – 11/20/21	Final Adjustments
	11/21/25	Video Demo
	11/24/25	Final Presentation
	11/25/25	GRADUATE!?

*\*Dates for Senior Design II are not finalized\**

### 11.3 Division of Labor

When dividing tasks, one of the most important considerations was each team member's availability. The availability of team members varied significantly, with the busiest member working a job, taking two classes, and taking senior design, to the least busy member focused solely on senior design. While limited availability is not considered a reason for taking on less responsibility, it is essential to understand each member's time constraints early on in the semester. One effective strategy was to define project goals concurrently with task distribution, allowing the team to assess the feasibility of the project and allocate work accordingly.

**Table 11.5: Division of Labor**

It is important to note that this task division table is not a rigid assignment of responsibilities. Throughout the development of the project, the team has maintained a flexible and collaborative approach, with members supporting each other where needed. In fact, this table has changed several times, as team members discovered interest in certain areas of the project or recognized they had taken on too much. Therefore, it is expected that this table will continue to evolve as the team adapts to meet the project's needs.

## 11.4 Team Organization

Finally, it is important to outline how the team will stay organized and what communication channels are used to ensure steady progress. Given the tight timeline, minimizing delays caused by miscommunication or misplaced files is critical. Clear, consistent communication and well-managed documentation are key to keeping the project on track.

First and foremost, the team has committed to meeting twice a week at minimum, with at least one of these meetings being in-person. This is critical to ensure that progress is made and that ideas are not lost through poor communication. Each meeting is planned to last approximately 2-3 hours. During these meetings one person is responsible for performing “minutes” and recording important decisions. To maximize efficiency, an agenda is compiled throughout the week and reviewed at the start of each meeting. Any additional meetings outside of the regular two are spontaneous and informal. As a result, minutes will not be taken unless the circumstances warrant it.

For online communication the team has chosen to use Discord. Discord is great for establishing multiple topics of communication. It is available on mobile and desktop devices, with compatibility across all operating systems. It is useful for sharing small files, links, photos, and drawings. Outside of text-based conversation, Discord also offers robust video call features with screen sharing. It excels at video calls which enable screensharing. It even has whiteboard capability through its built-in apps, though this feature can sometimes be clunky to use.

For file sharing, the team has agreed to use OneDrive to host files such as the 150-page report, datasheets, diagrams, and any other files that will be routinely accessed. The team also has a shared “working document” hosted on OneDrive, which acts as a centralized space for storing related ideas, links, tables, and more.

For programming the microcontroller and website, the team has agreed to use GitHub. This will streamline the coding process and provide a way to establish version control. Although the team is relatively new to collaboration via Git, the learning curve is considered a worthwhile time-investment for the benefits it offers in workflow organization.

## Chapter 12 – Conclusion

NepTune started as a design project with the simple idea of making a music synthesizer and given that senior design consists of both EE and CpE students, the combination of an analog and digital synthesizer was obvious. Digital would allow easy implementation of difficult features as well as simplifying portions of difficult circuitry meanwhile, the analog portion can provide complex and tactile interaction with the user attracting both lovers of manual playing and the customization of slot ready circuitry. The team has grown a strong passion to see NepTune come to realization as many members hold close ties to music and all of the team feels connected to the complex and intricate nature of music synthesis.

During the design of NepTune, it was important to keep track of which design goals had been met, 2 oscillators per voice would add enough richness to the sound. With 4 voices that allows 4 keys to be held down at once, ADSR, VCF, and Audio mixer customization from the user. A size that can be carried anywhere for mobile use as well as a low power 12V power input. The groundwork for additional features that can be developed has been laid as well.

Not every design detail came smoothly however, and the image of what NepTune has changed as progress has been made. Challenges initially took the form of size constraints as well as individual IC capacity. For every voice more space would be required, and each IC only could handle between 1 – 6 voices, each meaning rising cost of production. Niche digital/analog signal combinations affected the audio signal in a unique way that each member needed to become familiar with. High voltage parts would interrupt and cause noise meaning low power was a priority.

The team used a multitude of tools to better the outcome of both time and resources spent during production. Initially, communication was centered around a message board that the team could upload information to discuss topics both over text and voice chat as well as project information in real time to discuss. On top of this, a OneDrive and GitHub containing all of the teams' relevant files were shared to keep information organized. Schematics were formed on KiCad and Mermaid design suites.

Experience communicating within NepTunes design team was an effective one, quickly the team was able to realize issues as they arose and communicate contributions required. The team faced several scheduling troubles as many if not all members live busy lives and had to hold realistic expectations from their own capabilities. As the team consists of 4 EEs and a single CpE student, the priorities NepTune would have followed what most of the team was comfortable, moving the digital side of synthesizers to combine with analog synthesis. Other challenges were how to properly distance the different low and high frequency signals, and how to manage all of the different digital/analog signals being sent on NepTune's multiple modular PCBs. NepTune proved to be a difficult overall design task but with the testing completed so far and the expertise practiced in preparation for construction, NepTune has a bright future.

Another fantastic property about NepTune's design is that the room for expansion on not only voice capacity but for housing more features, is vast. Ideally, as the project is

developing, opportunities to bring the features the team has planned to fruition will be rewarding to see expanded on. Multiple members hope to have boards by the end of this process and to be able to know how to interictally expand on the modular design that has been dedicated to the user's benefit.

This senior design project has brought a team of 5 members together to build and expand not only their knowledge but also their interest in the field of signal modulation and sound synthesis. To create a product that is both technically sound and practical for the mass market of users. It appeals to the needs of users often associated with analog music synthesis while keeping a low enough entry cost to operate. Safe and effective design has been the central focus so that NepTune can be the best synth that the team can present.

## Chapter 13 - Reflections

As the project went through prototyping phase in Senior Design II, the project changed drastically from the initial goals. The team faced many difficulties including hardware issues, software issues, and went through many changes. Below are a list of changes the team incorporated.

### 13.1 Hardware

Throughout the prototyping phase, our project underwent several significant design changes as we adapted to practical constraints and lessons learned. Initially, we attempted to implement our system using a bare-metal STM32 microcontroller, but we encountered substantial difficulties related to low-level configuration, toolchain complexity, and development time. To keep the project on schedule while still meeting performance needs, we transitioned to the Electrosmith Daisy Seed platform for audio processing, which provided a more accessible development environment and robust DSP libraries. To continue satisfying the Senior Design requirement of incorporating a discrete microcontroller, we integrated an ESP32 module into the system architecture and used it for control and interface tasks. This decision allowed us to partition responsibilities more cleanly between high-level control and real-time audio, at the cost of increased effort in inter-device communication.

Several hardware and firmware subsystems also required iteration. Our ADSR envelope generators, implemented using PIC microcontrollers, initially exhibited functional issues that we traced back to firmware behavior. We resolved these by revising the PIC code and reprogramming the devices using a PICkit, restoring proper envelope shaping. We also encountered persistent challenges in establishing reliable UART communication between the ESP32 and the Daisy Seed, requiring multiple rounds of debugging and optimization to achieve stable parameter updates without disrupting audio performance. On the analog side, we originally selected the AD8397 as the output audio amplifier, but this choice proved difficult to work with in our application due to stability and implementation constraints. We ultimately replaced it with the LM386, which offered a

simpler, more forgiving solution for our single-ended audio output stage. Collectively, these changes reflect an iterative design process driven by real-world constraints, where we balanced technical ambition with reliability, simplicity, and the need to deliver a fully functioning prototype.

## 13.2 Software

For changes to software in Senior Design II, refer to the respective sections in this paper below.

1. 2.8 Senior Design II Hardware Block Diagram
2. 4.2.1 Digital Processing System / Digital Processing System Part Selection / Utilization of STM32H750 and Reselection of MCU Part (Senior Design 2 Revised)
3. 4.3.5 UART / UART between ESP32 and DaisySeed/STM32 (Senior Design 2 Revised)
4. Chapter 8 – Software Design (Senior Design 2 Revised)
5. Appendix D – Software Coding and Debugging (Senior Design 2 Revised)

# Appendices

## Appendix A – References

- [1] “Arturia - MicroFreak.” [Online]. Available: <https://www.arturia.com/products/hardware-synths/microfreak/overview>.
- [2] Magnolia International Ltd., “Behringer | Product | MS-1 MK II-GY.” [Online]. Available: <https://www.behringer.com/product.html?modelCode=0722-ACM>.
- [3] E. Tagi, “Learning synthesis: oscillators,” *Perfect Circuit*, Feb. 28, 2019. [Online]. Available: <https://www.perfectcircuit.com/signal/learning-synthesis-oscillators>.
- [4] “Getting started - Daisy.” [Online]. Available: <https://daisy.audio/tutorials/cpp-dev-env/>.
- [5] Electrosmith, “Daisy Seed,” *Electrosmith*. [Online]. Available: <https://electro-smith.com/products/daisy-seed>.
- [6] Electro-Smith, “GitHub - electro-smith/libDaisy: Hardware Library for the Daisy Audio Platform,” *GitHub*. [Online]. Available: <https://github.com/electro-smith/libDaisy>.
- [7] Electro-Smith, “GitHub - electro-smith/DaisySP: A Powerful DSP Library in C++,” *GitHub*. [Online]. Available: <https://github.com/electro-smith/DaisySP>.
- [8] K. Wada, “Bit-banging a digital-to-analog converter,” *Embedded*, Mar. 27, 2013. [Online]. Available: <https://www.embedded.com/bit-banging-a-digital-to-analog-converter-2>.
- [9] W. Storr, “Band Stop Filter and Notch Filter design tutorial,” *Basic Electronics Tutorials*, Mar. 24, 2023. [Online]. Available: <https://www.electronics-tutorials.ws/filter/band-stop-filter.html>.
- [10] W. Storr, “Sallen and key filter design for second order filters,” *Basic Electronics Tutorials*, Oct. 27, 2022. [Online]. Available: <https://www.electronics-tutorials.ws/filter/sallen-key-filter.html>.
- [11] Matsusada Precision Inc., “Linear vs. Switching Power Supplies | Tech | Matsusada Precision,” *Matsusada Precision*. [Online]. Available: [https://www.matsusada.com/column/linear\\_switching\\_difference.html](https://www.matsusada.com/column/linear_switching_difference.html).
- [12] S. Campbell, “Basics of the I2C communication Protocol,” *Circuit Basics*, Nov. 11, 2024. [Online]. Available: <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>.

- [13] “SPI vs I2C Protocol Differences and Things to Consider | Saleae Articles,” *Saleae Articles*. [Online]. Available: <https://articles.saleae.com/logic-analyzers/spi-vs-i2c-protocol-differences-and-things-to-consider>.
- [14] M. Hahn, “What is MIDI? How to use the most powerful tool in music,” *LANDR Blog*, Mar. 04, 2024. [Online]. Available: <https://blog.landr.com/what-is-midi/>.
- [15] G. Tabari, “What is MIDI? | The Basics, Benefits, Uses, & Applications in Music Production,” *ProShot Media*, Mar. 11, 2025. [Online]. Available: <https://proshotmediagroup.com/blog/what-is-midi-the-basics-benefits-uses-and-applications-in-music-production/>.
- [16] James, “All you need to know about TRS MIDI connections,” *Morningstar Engineering*, Jul. 12, 2022. [Online]. Available: <https://www.morningstar.io/post/all-you-need-to-know-about-trs-midi-connections>.
- [17] J. Johnson, “MIDI Connectors 101,” *Perfect Circuit*, Oct. 29, 2020. [Online]. Available: <https://www.perfectcircuit.com/signal/midi-connectors>.
- [18] C. Anderton, “What is MIDI? The Essential Guide,” *Sweetwater.com*, Jun. 07, 2019. [Online]. Available: <https://www.sweetwater.com/insync/midi-essential-guide/>.
- [19] Songstuff, “Midi Message Format - Songstuff,” *Songstuff*, Jul. 19, 2024. [Online]. Available: <https://www.songstuff.com/recording/article/midi-message-format>.
- [20] “Summary of MIDI 1.0 messages,” *MIDI.org*, Feb. 14, 2024. [Online]. Available: <https://midi.org/summary-of-midi-1-0-messages>.
- [21] “MIDI Tuning (Updated Specification),” *MIDI.org*, Feb. 14, 2024. [Online]. Available: <https://midi.org/midi-tuning-updated-specification>.
- [22] “Pure data - the Bela knowledge base.” [Online]. Available: <https://learn.bela.io/using-bela/languages/pure-data/>.
- [23] Warwick, “What are 8-Bit AVR microcontrollers? A beginner’s guide,” *AVR8 - A resource for 8-bit AVR microcontrollers*, Apr. 17, 2025. [Online]. Available: <https://avr8.com/what-are-8-bit-avr-microcontrollers/>.
- [24] “What is the CPU usage difference between C++ and Pure Data? - Bela.” [Online]. Available: <https://forum.bela.io/d/1703-what-is-the-cpu-usage-difference-between-c-and-pure-data>.
- [25] “Teensy Audio Library, high quality sound processing in Arduino sketches on Teensy 3.1.” [Online]. Available: [https://www.pjrc.com/teensy/td\\_libs\\_Audio.html](https://www.pjrc.com/teensy/td_libs_Audio.html).
- [26] A. Øland and R. B. Dannenberg, “FM Synthesis,” *Introduction to Computer Music Carnegie Mellon University*, Sep. 2017, [Online]. Available: [https://www.cs.cmu.edu/~music/icm-online/readings/fm-synthesis/fm\\_synthesis.pdf](https://www.cs.cmu.edu/~music/icm-online/readings/fm-synthesis/fm_synthesis.pdf)

[27] S. Rise “Wavetable Synthesis | The Synthesizer Academy,” *The Synthesizer Academy | Learn All About Synthesizers*, Mar. 08, 2016.  
<http://synthesizeracademy.com/wavetable-synthesis/>

[28] Jondent, “Microcontrollers & Synths.”  
<https://djjondent.blogspot.com/2019/02/microcontrollers.html>

[29] J. Schneider and I. Smalley, “Microcontroller,” *Topics*, Apr. 16, 2025.  
<https://www.ibm.com/think/topics/microcontroller>

[30] “Picking the right tool for the job: MCU, SBC or FPGA? | Bench Ta.”  
<https://www.mouser.com/blog/picking-the-right-tool-for-the-job-mcu-sbc-or-fpga>

[31] J. Schneider and I. Smalley, “Field programmable gate array,” *Topics*, Jul. 22, 2025.  
<https://www.ibm.com/think/topics/field-programmable-gate-arrays>

[32] Guest, “Waldorf Kyra to be the first fully FPGA powered synthesizer,” *Synthtopia*, Jun. 08, 2018. <https://www.synthtopia.com/content/2018/06/08/waldorf-kyra-to-be-the-first-fully-fpga-powered-synthesizer/>

[33] “Ultra-low latency trading: How low can you go?,” *WaterTechnology.com*, Jan. 29, 2025. [Online]. Available: <https://www.watertechnology.com/market-access/data/7952118/ultra-low-latency-trading-how-low-can-you-go>

[34] A. Aravindan, “Flash 101: NAND Flash vs NOR Flash,” *Embedded*, Aug. 23, 2022. [Online]. Available: <https://www.embedded.com/flash-101-nand-flash-vs-nor-flash/>

[35] B. Lutkevich, B. Posey, and R. Brown, “NOR flash memory,” *Search Storage*, Jun. 09, 2023. <https://www.techtarget.com/searchstorage/definition/NOR-flash-memory>

[36] “Bits on chips,” *Google Books*.  
[https://books.google.com/books?id=7XhhDwAAQBAJ&dq=nor+random+access&pg=P\\_A113#v=onepage&q=nor%20random%20access&f=false](https://books.google.com/books?id=7XhhDwAAQBAJ&dq=nor+random+access&pg=P_A113#v=onepage&q=nor%20random%20access&f=false)

[37] “Binary Weighted Resistor DAC | Analog-Integrated-Circuits || Electronics Tutorial.” <https://www.electronics-tutorial.net/analog-integrated-circuits/data-converters/binary-weighted-resistor-dac/>

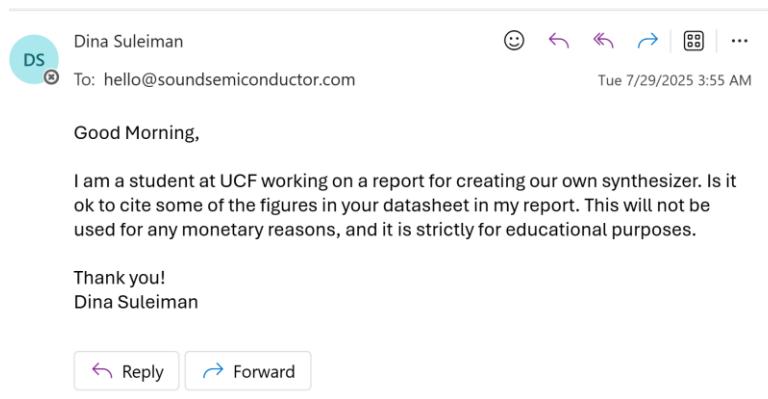
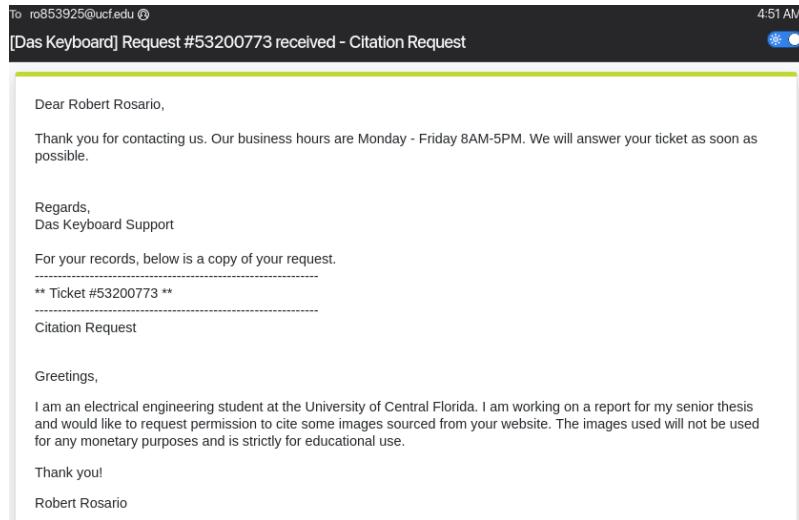
[38] W. Kester, “Basic DAC Architectures II: Binary DACs,” Analog Devices, MT-015. [Online]. Available: <https://www.analog.com/media/en/training-seminars/tutorials/MT-015.pdf>

- [39] STMicroelectronics, “What does precision mean for an op amp?” YouTube, Apr. 27, 2023. [Online]. Available: <https://www.youtube.com/watch?v=i8dcN-avksg>.
- [40] Analog Devices, “Op Amp Common-Mode Rejection Ratio (CMRR),” Analog Devices, MT-042, 2009. [Online]. Available: <https://www.analog.com/media/en/training-seminars/tutorials/MT-042.pdf>.
- [41] Texas Instruments, “What You Need to Know about Input Bias Current – and Why,” SSZTAJ8, Texas Instruments, Dec. 2016. [Online]. Available: <https://www.ti.com/document-viewer/lit/html/SSZTAJ8>
- [42] C. Poole and I. Darwazeh, “Chapter 14 - Low-noise amplifier design,” in *Microwave Active Circuit Analysis and Design*, C. Poole and I. Darwazeh, Eds., San Diego, CA: Academic Press, 2016, pp. 475–517. [Online]. Available: <https://doi.org/10.1016/B978-0-12-407823-9.00014-7>
- [43] Analog Devices, “Low Noise Amplifier: Selection Guide for Optimal Noise Performance,” Analog Devices, Application Note AN-940, 2011. [Online]. Available: <https://www.analog.com/en/resources/app-notes/an-940.html>
- [44] N. C. Braga, “How they work: Thermal drift (ART287E),” *INCB Tech.* [Online]. Available: <https://www.incbtech.com/articles/16-how-they-work/1909-how-they-work-thermal-drift-art287e.html>.
- [45] “Discrete circuit vs integrated circuits – what you know,” *Electronics Workshops*, Oct. 23, 2024. [Online]. Available: <https://electronicsworkshops.com/2024/10/23/discrete-circuit-vs-integrated-circuits-what-you-know/>
- [46] P. Horowitz and W. Hill, “4x.3 Transresistance Amplifiers,” *The Art of Electronics*, 3rd ed., 2019. [Online]. Available: [https://x.artofelectronics.net/wp-content/uploads/2019/11/4xp3\\_TIA.pdf](https://x.artofelectronics.net/wp-content/uploads/2019/11/4xp3_TIA.pdf)
- [47] nwavguy, “Headphone impedance explained,” *nwavguy’s blog*, Feb. 16, 2011. [Online]. Available: <https://nwavguy.blogspot.com/2011/02/headphone-impedance-explained.html>.
- [48] A. Pini, “The basics of direct digital synthesizers (DDSs) and How to Select and Use Them,” *Digi-Key Electronics*, Mar. 3, 2019. [Online]. Available: <https://www.digikey.com/en/articles/the-basics-of-direct-digital-synthesizers-ddss>
- [49] Adrian. “Unlocking Precision: How Discrete Components Optimize Analog Circuit Design”, *AllPCB Blog*, Jun. 17, 2025. [Online]. Available: <https://www.allpcb.com/blog/pcb-assembly/unlocking-precision-how-discrete-components-optimize-analog-circuit-design.html>

[50] E. Murphy and C. Slattery, “Ask the Application Engineer-33: All About Direct Digital Synthesis,” *Analog Dialogue*, vol. 38, Aug 2004. [Online]. Available: <https://www.analog.com/en/resources/analog-dialogue/articles/all-about-direct-digital-synthesis.html>

[51] “DDS3: Direct Digital Synthesis — Example Problem,” *FPGA4Fun*. [Online]. Available: <https://www.fpga4fun.com/DDS3.html>.

## Appendix B – copyright permission



Permission request for figures 4.4-4.6



Anthony DeRosa

To: contact@electronics.org

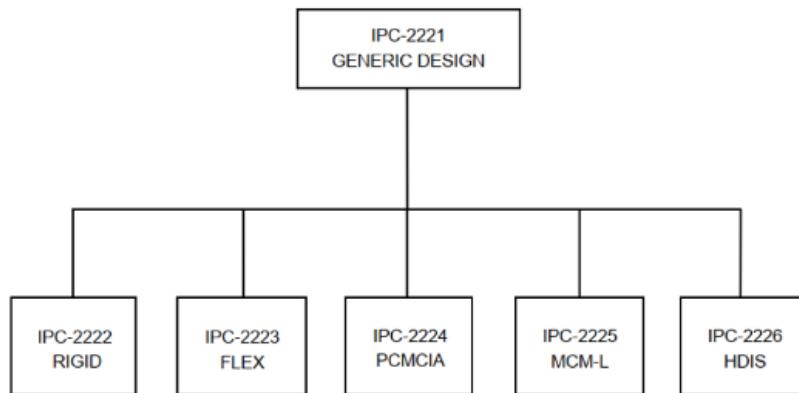
😊 ⏪ ⏴ ⏵ | 📄 | ...

Tue 7/29/2025 7:48 AM

Hello,

I am part of a group of university students working on a report as part of our final project before graduation.

We are requesting permission to use the following diagram found in the IPC-2221A documentation as part of a chapter explaining standards.



Is this acceptable? We appreciate the help.

Thank you,  
Anthony DeRosa

Permission request for IPC-2221A diagram, Figure 5.1.

AD

Anthony Derosa

To: distribution.literature@analog.com



Tue 7/29/2025 7:56 AM

Hello,

I am part of a group of university students working on a report as part of our final project before graduation.

We are requesting permission to use the following table found in the **Application Note AN-1429: "Design Considerations and Solutions for Headphone Drivers in Mobile Phones" By David Gua.**

The table in question is Table 1, as seen below:

**Table 1. Sensitivity, Impedance, Peak Voltage, and Peak Current of Headphones**

Manufacturer	Model	Sensitivity (dB/mW)	Impedance ( $\Omega$ )	Frequency (Hz)	Average Power (mW)	Peak Power (mW)	Peak Voltage (V)
SONY	XBA-4	108	8	3 to 28000	0.158	15.849	0.356
Audio-Technica	ATH-CHX7	100	16	15 to 22000	1.000	100.000	1.265
Shure	SE215	107	20	22 to 17500	0.200	19.953	0.632
Apple	Earpod	109	23	5 to 21000	0.126	12.589	0.538
Grado	Alice M1	100	32	20 to 22000	1.000	100.000	1.789
Creative	AURVANA Air	102	32	20 to 20000	0.631	63.096	1.421
KOSS	PP	101	60	10 to 25000	0.794	79.433	2.183
Sennheiser	HD650	98	300	10 to 39500	1.585	158.489	6.895
Beyer Dynamic	DT880	96	600	5 to 35000	2.512	251.189	12.277

Is this acceptable? We appreciate your help.

Thank you,  
Anthony DeRosa

Permission request for Table 7.1

## Appendix C – data sheet

[https://www.coolaudio.com/docs/Cool%20Audio\\_Data\\_Sheet\\_V3320\\_VCF.pdf](https://www.coolaudio.com/docs/Cool%20Audio_Data_Sheet_V3320_VCF.pdf)

<https://www.soundsemiconductor.com/downloads/ssi2140datasheet.pdf>

<https://www.ti.com/lit/ds/symlink/pcm3060.pdf>

[https://www.ti.com/lit/ds/symlink/pcm3060.pdf?ts=1753723221505&ref\\_url=https%253A%252F%252Fwww.mouser.de%252F](https://www.ti.com/lit/ds/symlink/pcm3060.pdf?ts=1753723221505&ref_url=https%253A%252F%252Fwww.mouser.de%252F)

<https://u.dianyuan.com/bbs/u/51/1174974780.pdf>

<https://sunsite3.icm.edu.pl/packages/linux-alsa/manuals/usb/midi10.pdf>

<https://www.nxp.com/docs/en/nxp/data-sheets/IMXRT1060CEC.pdf>

## Appendix D – Software Coding and Debugging (Senior Design 2 Revised)

```
Memory region      Used Size  Region Size %age Used
    FLASH:        75152 B   128 KB   57.34%
    DTCMRAM:      0 GB     128 KB   0.00%
    SRAM:         13004 B   512 KB   2.48%
    RAM_D2:       16704 B   288 KB   5.66%
    RAM_D3:       0 GB     64 KB    0.00%
    BACKUP_SRAM:  12 B     4 KB    0.29%
    ITCMRAM:      0 GB     64 KB    0.00%
    SDRAM:        0 GB     64 MB    0.00%
    QSPIFLASH:    0 GB     8 MB    0.00%
arm-none-eabi-objcopy -O ihex build/oscillator.elf build/oscillator.hex
arm-none-eabi-objcopy -O binary -S build/oscillator.elf build/oscillator.bin
dfu-util -a 0 -s 0x08000000:leave -D build/oscillator.bin -d ,0483:df11
dfu-util 0.11

Copyright 2005-2009 Weston Schmidt, Harald Welte and OpenMoko Inc.
Copyright 2010-2021 Tormod Volden and Stefan Schmidt
This program is Free Software and has ABSOLUTELY NO WARRANTY
Please report bugs to http://sourceforge.net/p/dfu-util/tickets/

dfu-util: Warning: Invalid DFU suffix signature
dfu-util: A valid DFU suffix will be required in a future dfu-util release
Opening DFU capable USB device...
Device ID 0483:df11
Device DFU version 011a
Claiming USB DFU Interface...
Setting Alternate Interface #0 ...
Determining device status...
DFU state(2) = dfuIDLE, status(0) = No error condition is present
DFU mode device DFU version 011a
Device returned transfer size 1024
DfuSe interface name: "Internal Flash"
Downloading element to address = 0x08000000, size = 75152
Erase      [=====] 100%          75152 bytes
Erase done.
Download   [=====] 100%          75152 bytes
Download done.
File downloaded successfully
Submitting leave request...
```

Figure F.1

```
#include <Arduino.h>
#include <SPI.h>
#include <Preferences.h>
```

```

// ====== UART CONFIG ======
//static const uint32_t CTRL_BAUD = 115200; // Daisy RX=16, TX=17

// ====== MUX CONFIG ======
// we'll KEEP these so the rest of your code still compiles
const int M1_S0 = 4, M1_S1 = 2, M1_S2 = 32, M1_S3 = 33, M1_EN = 21;
const int M2_S0 = 25, M2_S1 = 26, M2_S2 = 27, M2_S3 = 14, M2_EN = 22;
const int ADC_M1 = 36; // ADC1_CH0
const int ADC_M2 = 39; // ADC1_CH3

// ====== DIGIPOT CONFIG ======
const int PIN_SCK = 18;
const int PIN_MISO = 19;
const int PIN_MOSI = 23;
const int PIN_CS1 = 5;
const int PIN_CS2 = 15;

const int N_MUX = 32;
const int N_DIG = 12;
uint16_t muxVals[N_MUX];
uint8_t digiVals[N_DIG];

// which MUX inputs map to digipots and Daisy
uint8_t digiRoute[N_DIG] = {4,5,14,13,12,11,30,25,28,26,27,29};
uint8_t daisyRoute[8] = {0,1,2,3,18,16,17,19};

// ====== UTILITIES ======
inline void setAddr(int s0,int s1,int s2,int s3,int ch){
    digitalWrite(s0,(ch>>0)&1);
    digitalWrite(s1,(ch>>1)&1);
    digitalWrite(s2,(ch>>2)&1);
    digitalWrite(s3,(ch>>3)&1);
}

uint16_t readADC(int pin){
    int raw = analogRead(pin); // 0-4095 (12-bit)
    return (uint16_t)((raw<<4) | (raw>>8)); // expand to 16-bit
}

```

```

}

// EN is still a parameter so calls don't change, but we do NOTHING with it
void scanMux16(int s0,int s1,int s2,int s3,int en,int adcPin,uint16_t* out16){
    digitalWrite(en, HIGH);
    for(int ch = 0; ch < 16; ++ch){
        setAddr(s0,s1,s2,s3,ch);
        delayMicroseconds(20);
        (void)analogRead(adcPin); // throwaway sample
        delayMicroseconds(20);
        out16[ch] = readADC(adcPin);
    }
}

// ===== DIGIPOT DRIVER =====
void ad5206Write(int cs,uint8_t channel,uint8_t value){
    digitalWrite(cs,LOW);
    SPI.transfer(channel & 0x0F);
    SPI.transfer(value);
    digitalWrite(cs,HIGH);
}

void setDigi(uint8_t index,uint8_t val){
    if(index >= N_DIG) return;
    uint8_t chip = index / 6;
    uint8_t chan = index % 6;
    int cs = (chip == 0) ? PIN_CS1 : PIN_CS2;
    if(digiVals[index] != val){
        digiVals[index] = val;
        ad5206Write(cs, chan, val);
    }
}

static inline uint8_t map16to8(uint16_t v){ return v >> 8; }

// ===== SETUP / LOOP =====
void setup() {

```

```

// UART to Daisy (TX=17, RX=16)
Serial2.begin(76800, SERIAL_8N1, 3, 1);

pinMode(M1_S0,OUTPUT); pinMode(M1_S1,OUTPUT);
pinMode(M1_S2,OUTPUT); pinMode(M1_S3,OUTPUT);
pinMode(M2_S0,OUTPUT); pinMode(M2_S1,OUTPUT);
pinMode(M2_S2,OUTPUT); pinMode(M2_S3,OUTPUT);
// we still declare EN pins, but we don't drive them
pinMode(M1_EN,OUTPUT);
pinMode(M2_EN,OUTPUT);
digitalWrite(M1_EN,HIGH);
digitalWrite(M2_EN,HIGH);

analogReadResolution(12);

// just in case you need that extra pin
pinMode(12, OUTPUT);
digitalWrite(12, HIGH);

SPI.begin(PIN_SCK, PIN_MISO, PIN_MOSI, PIN_CS1);
pinMode(PIN_CS1,OUTPUT);
pinMode(PIN_CS2,OUTPUT);
digitalWrite(PIN_CS1,HIGH);
digitalWrite(PIN_CS2,HIGH);

// init digipots
for(int ch = 0; ch < 6; ++ch){
    ad5206Write(PIN_CS1, ch, 128);
    ad5206Write(PIN_CS2, ch, 128);
    digiVals[ch] = 128;
    digiVals[ch+6] = 128;
}
uint8_t tx_bytes[18] =
{
    0xAA,
    0x00,
    0x00,

```



```

rx_byte = Serial2.read();

if(rx_byte == 0x69)

{

    for(uint8_t i = 0; i < 8; i++)

        p[i] = muxVals[daisyRoute[i]];


    tx_bytes[0] = 0xAA; // Start marker

    for(uint8_t i = 1; i < 17; i++)

    {

        if((i % 2) == 1) // MSByte

            tx_bytes[i] = (uint8_t)((p[(i-1) >> 1] & 0xFF00) >> 8); // Copy top half of 16bit

knob to first byte

        else // LSByte

            tx_bytes[i] = (uint8_t)(p[(i-1) >> 1] & 0x00FF); // Copy bottom half of 16bit

knob to second byte

    }

    tx_bytes[17] = 0x55; // End marker

    Serial2.write(tx_bytes, 18);

}

}

delayMicroseconds(100);

}

```

**Figure D.2:** ESP32 code (Arduino IDE)

```

// File: src/oscillator.cpp
// Purpose: 4-voice synth with Analog Envelopes + PITCH BEND support.
//
// FIXED: Truncation error (restored missing main loop).
// FIXED: "Defined but not used" warnings.
// ADDED: Pitch Bend (+/- 2 Semitones).

#include "daisysp.h"
#include "daisy_seed.h"
#include "hid/midi.h"

using namespace daisy;
using namespace daisysp;
using namespace seed;

// ===== Hardware =====

```

```

DaisySeed hw;
static constexpr int kNumVoices = 4;

UartHandler uart;

// ----- Constants for framing -----
static constexpr uint8_t HEADER1 = 0xAA;
static constexpr uint8_t HEADER2 = 0x55;
static constexpr int MAX_FRAME = 128;

// ----- Receive buffer variables -----
const size_t uart_buff_size = 18;
uint8_t DMA_BUFFER_MEM_SECTION uart_rx_buff[uart_buff_size];

uint8_t rxBuf[MAX_FRAME];
int rxIndex = 0;
int expectedLen = 0;
int state = 0;

// ===== Params =====
float volume1 = 1.0f, volume2 = 1.0f;
float pulseW1 = 0.5f, pulseW2 = 0.5f;
float detune1 = 0.5f;
float detune2 = 0.5f;

// ===== Pitch Bend State =====
// 1.0 = no bend. >1.0 = sharp, <1.0 = flat.
float g_bend_mult = 1.0f;

// ===== Synthesis =====
Oscillator osc1[kNumVoices];
Oscillator osc2[kNumVoices];

// ===== MIDI =====
MidiUartHandler midi;

// ===== Voice/Allocation State =====
struct Voice {
    bool active = false;
    bool gate = false;
    int note = -1;
};

Voice voices[kNumVoices];

```

```

// Per-MIDI-note state
static constexpr int kNumMidiNotes = 128;
bool      midi_held[kNumMidiNotes] = {false};
int       midi_voice[kNumMidiNotes];
uint32_t  midi_hold_ts[kNumMidiNotes];

uint32_t global_press_counter = 0;

// --- GATES D15-D18 ---
static const Pin kGatePins[4] = { D15, D16, D17, D18 };
static GPIO      kGates[4];

// ----- Gate Helpers -----
static void UpdateGates()
{
    bool any_gate_active = false;
    for(int i = 0; i < 4; ++i)
    {
        kGates[i].Write(voices[i].gate);
        if(voices[i].gate) any_gate_active = true;
    }
    hw.SetLed(any_gate_active);
}

static inline int FindFreeVoice()
{
    for(int v = 0; v < kNumVoices; ++v)
        if(!voices[v].gate) return v;
    return -1;
}

static int FindOldestActiveVoice()
{
    int oldest_vi = -1;
    uint32_t oldest_ts = 0;
    for(int v=0; v<kNumVoices; ++v)
    {
        if(!voices[v].gate) continue;
        const int n = voices[v].note;
        const uint32_t ts = midi_hold_ts[n];
        if(oldest_vi < 0 || ts < oldest_ts) { oldest_ts = ts; oldest_vi =
v; }
    }
    return oldest_vi;
}

```

```

}

static void AssignVoiceToMidi(int voice_idx, int note)
{
    int old_note = voices[voice_idx].note;
    if(old_note >= 0 && midi_voice[old_note] == voice_idx) {
        midi_voice[old_note] = -1;
    }

    voices[voice_idx].active = true;
    voices[voice_idx].note = note;
    voices[voice_idx].gate = true; // PIN HIGH
    midi_voice[note] = voice_idx;
}

static void ReleaseVoice(int voice_idx)
{
    const int note = voices[voice_idx].note;
    if(note >= 0 && midi_voice[note] == voice_idx)
        midi_voice[note] = -1;

    voices[voice_idx].gate = false; // PIN LOW
}

static void OnMidiNoteOn(int note)
{
    if(note < 0 || note >= kNumMidiNotes) return;
    midi_held[note] = true;
    midi_hold_ts[note] = ++global_press_counter;

    int free_v = FindFreeVoice();
    if(free_v >= 0)
    {
        AssignVoiceToMidi(free_v, note);
        UpdateGates();
        return;
    }

    int steal_vi = FindOldestActiveVoice();
    if(steal_vi < 0) steal_vi = 0;

    AssignVoiceToMidi(steal_vi, note);
    UpdateGates();
}

```

```

static void OnMidiNoteOff(int note)
{
    if(note < 0 || note >= kNumMidiNotes) return;
    midi_held[note] = false;
    int owned_v = midi_voice[note];
    if(owned_v >= 0)
        ReleaseVoice(owned_v);

    UpdateGates();
}

// ===== Pitch Bend Handler =====
static void HandlePitchBend(MidiEvent m)
{
    // PitchBend is 14-bit: Data1 (LSB) + Data2 (MSB)
    // Value range: 0 to 16383. Center (no bend) is 8192.
    uint16_t val = (m.data[1] << 7) | m.data[0];

    // Normalize to -1.0 to 1.0
    float norm = (float)(val - 8192) / 8192.0f;

    // Set Bend Range: +/- 2 Semitones
    float semitones = norm * 2.0f;

    // Convert semitones to frequency multiplier: 2^(semitones/12)
    g_bend_mult = powf(2.0f, semitones / 12.0f);
}

// ===== MIDI event handling =====
static void HandleMidiMessage(MidiEvent m)
{
    if(m.type == NoteOn && m.data[1] > 0)
        OnMidiNoteOn(m.data[0]);
    else if(m.type == NoteOff || (m.type == NoteOn && m.data[1] == 0))
        OnMidiNoteOff(m.data[0]);
    else if(m.type == PitchBend) // Handle Pitch Bend Events
        HandlePitchBend(m);
}

// ===== Audio Callback =====
static void AudioCallback(AudioHandle::InputBuffer in,
AudioHandle::OutputBuffer out, size_t size)
{
    // const float cents1 = (detune1 - 0.5f) * 400.0f;
    // const float cents2 = (detune2 - 0.5f) * 400.0f;
}

```

```

// const float detuneFactor1 = powf(2.0f, cents1 / 1200.0f);
// const float detuneFactor2 = powf(2.0f, cents2 / 1200.0f);
const float detuneFactor1 = powf(2.0f, ((2*detune1) - 1)); // Octave
ranging code
const float detuneFactor2 = powf(2.0f, ((2*detune2) - 1));
const float mix_scale = 1.f / (2.f * kNumVoices);

for(int v = 0; v < kNumVoices; ++v)
{
    int note = voices[v].note >= 0 ? voices[v].note : 60;
    const float f = mtod(static_cast<float>(note));

    // Apply g_bend_mult to the frequency
    osc1[v].SetFreq(f * detuneFactor1 * g_bend_mult);
    osc2[v].SetFreq(f * detuneFactor2 * g_bend_mult);

    osc1[v].SetAmp(volume1);
    osc1[v].SetPw(pulseW1);

    osc2[v].SetAmp(volume2);
    osc2[v].SetPw(pulseW2);
}

for(size_t i = 0; i < size; ++i)
{
    float mix[kNumVoices];
    for(int v = 0; v < kNumVoices; ++v)
    {
        if(voices[v].active)
            mix[v] = osc1[v].Process() + osc2[v].Process();
        else
            mix[v] = 0.0f;
        mix[v] *= mix_scale;
    }

    out[0][i] = mix[0];
    out[1][i] = mix[1];
    out[2][i] = mix[3];
    out[3][i] = mix[2];
}
}

// ----- CRC8 -----
uint8_t crc8(const uint8_t* d, int n)
{

```

```

    uint8_t c = 0;
    for(int i = 0; i < n; i++) {
        c ^= d[i];
        for(int b = 0; b < 8; b++)
            c = (c & 0x80) ? (c << 1) ^ 0x31 : (c << 1);
    }
    return c;
}

static inline float fmap_range(uint16_t v, uint16_t in_min, uint16_t
in_max, float out_min, float out_max)
{
    if(in_max == in_min) return out_min;
    float t = (float)(v - in_min) / (float)(in_max - in_min);
    if(t < 0.0f) t = 0.0f;
    if(t > 1.0f) t = 1.0f;
    return out_min + t * (out_max - out_min);
}

void ApplyParameters(uint16_t* vals, int n)
{
    //if(n < 11) return;

    float waveSel1 = fmap_range(vals[0], 0, 65535, 0.0f, 4.99f);
    detune1 = fmap_range(vals[1], 0, 65535, 0.0f, 1.0f);
    pulseW1 = fmap_range(vals[2], 0, 65535, 0.0f, 1.0f);
    volume1 = fmap_range(vals[3], 0, 65535, 0.0f, 1.0f);

    float waveSel2 = fmap_range(vals[4], 0, 65535, 0.0f, 4.99f);
    detune2 = fmap_range(vals[5], 0, 65535, 0.0f, 1.0f);
    pulseW2 = fmap_range(vals[6], 0, 65535, 0.0f, 1.0f);
    volume2 = fmap_range(vals[7], 0, 65535, 0.0f, 1.0f);

    int waveIndex1 = static_cast<int>(waveSel1);
    int waveIndex2 = static_cast<int>(waveSel2);
    for(int v = 0; v < kNumVoices; ++v)
    {
        switch(waveIndex1) {
            case 0:
                osc1[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SAW);           break;
            case 1:
                osc1[v].SetWaveform(Oscillator::WAVE_POLYBLEP_TRI);      break;
            case 2: osc1[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SQUARE); break;
            case 3: osc1[v].SetWaveform(Oscillator::WAVE_SIN);       break;
        }
    }
}

```

```

        case 4:
osc1[v].SetWaveform(Oscillator::WAVE_RAMP);           break;
    }
    switch(waveIndex2) {
        case 0:
osc2[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SAW);      break;
        case 1:
osc2[v].SetWaveform(Oscillator::WAVE_POLYBLEP_TRI);      break;
        case 2: osc2[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SQUARE);
break;
        case 3: osc2[v].SetWaveform(Oscillator::WAVE_SIN);     break;
        case 4:
osc2[v].SetWaveform(Oscillator::WAVE_RAMP);           break;
    }
}

// static void TryApplyAll()
// {
//     if(g_ctrl_parts == 0x0F)
//     {
//         ApplyParameters(g_ctrl_buf, 11);
//         g_ctrl_parts = 0;
//     }
// }

// void ProcessFrame(uint8_t* data, int n)
// {
//     if(n < 1 + 1 + 4 + 1) return;
//     int p = 0;
//     uint8_t len  = data[p++];
//     uint8_t type = data[p++];
//     p += 4; // skip seq

//     int param_bytes = len - (1 + 4);
//     int nparms = param_bytes / 2;
//     if(nparms <= 0) return;

//     switch(type)
//     {
//         case 0x01:
//             for(int i = 0; i < nparms && i < 3; i++) {
//                 g_ctrl_buf[0 + i] = data[p] | (data[p+1] << 8);
//                 p += 2;
//             }
//     }
}

```

```

//           g_ctrl_parts |= 0x01;
//           break;
//       case 0x02:
//           for(int i = 0; i < nparams && i < 3; i++) {
//               g_ctrl_buf[3 + i] = data[p] | (data[p+1] << 8);
//               p += 2;
//           }
//           g_ctrl_parts |= 0x02;
//           break;
//       case 0x03:
//           for(int i = 0; i < nparams && i < 3; i++) {
//               g_ctrl_buf[6 + i] = data[p] | (data[p+1] << 8);
//               p += 2;
//           }
//           g_ctrl_parts |= 0x04;
//           break;
//       case 0x04:
//           for(int i = 0; i < nparams && i < 2; i++) {
//               g_ctrl_buf[9 + i] = data[p] | (data[p+1] << 8);
//               p += 2;
//           }
//           g_ctrl_parts |= 0x08;
//           break;
//       }
//   TryApplyAll();
// }

// void ParseByte(uint8_t b)
// {
//     switch(state)
//     {
//         case 0: if(b == HEADER1) state = 1; break;
//         case 1: if(b == HEADER2) state = 2; else state = 0; break;
//         case 2:
//             expectedLen = b + 1;
//             rxIndex = 0;
//             state = 3;
//             break;
//         case 3:
//             rxBuf[rxIndex++] = b;
//             if(rxIndex >= expectedLen)
//             {
//                 uint8_t crc_recv = rxBuf[expectedLen - 1];
//                 uint8_t crc_calc = crc8(rxBuf, expectedLen - 1);
//                 if(crc_recv == crc_calc)
//             }
//     }
// }

```

```

//                                ProcessFrame(rxBuf, expectedLen - 1);
//                                state    = 0;
//                                rxIndex = 0;
//                            }
//                            break;
//                        }
//                    }

// void ReceiveLoop()
// {
//     //uart.BlockingReceive(uart_rx_buff, sizeof(uart_rx_buff), 30);
//     // while(uart.BlockingReceive(b, 1, 0) == UartHandler::Result::OK)
//     // {
//     //     ParseByte(b);
//     // }
// }

void GetKnobs() { // function that polls ESP32
    uint8_t command = 0x69; // command to send
    for(uint8_t i = 0; i < 18; i++)
        uart_rx_buff[i] = 0x00;
    uart.BlockingTransmit(&command, 1, 10); // Send a knob request
    // command to ESP32
    uart.BlockingReceive(uart_rx_buff, 18, 200); // Wait (up to 200ms) for
    // ESP32 to reply with knob data (18bytes)
    uint16_t knob_values[8] = {0x0000};
    for(uint8_t i = 0; i < 8; i++)
        knob_values[i] = (uart_rx_buff[(2 * i) + 1] << 8) |
    uart_rx_buff[(2*i) + 2];
    ApplyParameters(knob_values, 8);
}

void InitUart()
{
    UartHandler::Config cfg;
    cfg.periph = UartHandler::Config::Peripheral::UART_4;
    cfg.mode   = UartHandler::Config::Mode::TX_RX;
    cfg.baudrate = 76800;
    cfg.pin_config.rx = D11;
    cfg.pin_config.tx = D12;
    uart.Init(cfg);
}

int main(void)
{

```

```

hw.Configure();
hw.Init();
hw.SetAudioBlockSize(48);

// 1. Initialize MIDI (activates USART1)
{
    MidiUartHandler::Config midi_cfg;
    midi_cfg.transport_config.periph =
UartHandler::Config::Peripheral::USART_1;
    midi_cfg.transport_config.rx = D30;
    midi_cfg.transport_config.tx = D29;
    midi.Init(midi_cfg);
}

// 2. Initialize GATES (D15-D18)
// CRITICAL: Run AFTER MIDI init to reclaim D15 as GPIO Output
for(int i = 0; i < 4; ++i)
{
    kGates[i].Init(kGatePins[i], GPIO::Mode::OUTPUT,
GPIO::Pull::PULLDOWN);
    kGates[i].Write(false);
}

// 3. Oscillators
for(int v=0; v<kNumVoices; ++v)
{
    osc1[v].Init(hw.AudioSampleRate());
    osc2[v].Init(hw.AudioSampleRate());
    //osc1[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SAW);
    //osc2[v].SetWaveform(Oscillator::WAVE_POLYBLEP_SQUARE);
    osc1[v].SetAmp(1.0f);
    osc2[v].SetAmp(1.0f);
}

for(int n = 0; n < kNumMidiNotes; ++n)
{
    midi_voice[n]    = -1;
    midi_hold_ts[n] = 0;
    midi_held[n]    = false;
}
UpdateGates();

// 4. SAI2 (External PCM3060) config
SaiHandle        sai2;
SaiHandle::Config sc;

```

```

sc.periph    = SaiHandle::Config::Peripheral::SAI_2;
sc.sr        = SaiHandle::Config::SampleRate::SAI_48KHZ;
sc.bit_depth = SaiHandle::Config::BitDepth::SAI_24BIT;
sc.a_sync    = SaiHandle::Config::Sync::SLAVE;
sc.b_sync    = SaiHandle::Config::Sync::MASTER;
sc.a_dir     = SaiHandle::Config::Direction::TRANSMIT;
sc.b_dir     = SaiHandle::Config::Direction::RECEIVE;
sc.pin_config.mclk = D24;
sc.pin_config.sck  = D28;
sc.pin_config.fs   = D27;
sc.pin_config.sa   = D26;
sc.pin_config.sb   = D25;
sai2.Init(sc);

// 5. UART and Audio Start
InitUart();

AudioHandle::Config audio_cfg;
audio_cfg.blocksize  = 48;
audio_cfg.samplerate = SaiHandle::Config::SampleRate::SAI_48KHZ;
audio_cfg.postgain   = 0.5f;

hw.audio_handle.Init(audio_cfg, hw.AudioSaiHandle(), sai2);
hw.StartAudio(AudioCallback);

// 6. Main Loop
while(true)
{
    midi.Listen();
    while(midi.HasEvents())
    {
        HandleMidiMessage(midi.PopEvent());
        GetKnobs();
    }
    //ReceiveLoop();
}
}

```

*Figure D.3: DaisySeed (STM32) code*

## Appendix E – LLM prompts and outcomes

E.1.Answer1:.

“Oscillator

An oscillator is a sound source.

It generates a periodic waveform, such as:

- Sine
- Square
- Sawtooth
- Triangle

This waveform is the raw audio signal.

Synths often have multiple oscillators per voice.

Think of it as the basic tone generator.

Example:

An oscillator might produce a sawtooth wave at 440 Hz (the pitch of A4).

Voice

A voice is a complete signal path that includes at least one oscillator, plus:

- A filter (to shape the tone)
- An amplifier (to control loudness)
- Envelopes (for attack, decay, sustain, release)
- Possibly LFOs (for modulation)

It's the entire chain required to produce a single note.

A polyphonic synth can play multiple voices simultaneously (e.g., 8 voices = 8 simultaneous notes).

A monophonic synth can only play one voice at a time.

Example:

If you press one key on a polyphonic synth, it triggers one voice (which may use 2 or 3 oscillators inside it).

Summary

Term-What it is-Role

Oscillator-Tone generator-Produces raw waveforms (sound source)

Voice-Full sound path for 1 note-Includes oscillator(s), filter, amp, etc.

Analogy:

Imagine a band:

An oscillator is like a guitar string—it makes sound.

A voice is like a guitarist's entire rig—guitar, pedals, amp, everything to play a note.

Let me know if you'd like a diagram or audio example to go with this!

“Polyphonic synths have multiple voices, meaning they can play multiple notes at once—a “6voice” synth can play six simultaneous notes (dailyanalog.com, noiseengineering.us, Gearspace, Reddit).

“Generally it’s how many notes can be played at once. A synth like a Moog Model D is monophonic (one voice) even though it has 3 oscillators...” (Reddit)

“I think of ‘voice’ as ‘sound(s) at a time.’ ... A 1voice synth can play one note at a time... while a twovoice synth is two ‘sounds at a time’...” (Gearspace)

An oscillator is part of a voice—it’s the raw sound generator that produces a continuous waveform (sine, square, sawtooth, pulse, triangle). It defines the pitch and timbre source for a voice (Wikipedia).

“These tone generators play a big role in the final sound. An oscillator generates a steady pitch which can be played musically by a keyboard.” (optoproductions.com)

“A circuit that produces a stable and controllable AC signal. Sound itself is oscillations” (Gearspace)

Term	Definition
Oscillator	Generates raw periodic waveform (sound source). A voice may contain multiple oscillators.
Voice	Complete signal chain for one playable note: includes oscillator(s), filter, envelope, VCA, etc.
Polyphony	Number of voices available to play simultaneous notes.

### In Summary

- A voice = everything you need to produce one note (oscillator + filter + amplifier + envelopes).
- An oscillator = just the sound producing component of a voice—generates waveform.”

## E.2 Answer 2:

Great question! Choosing the right filter topology for your hybrid synthesizer depends on several factors: sound character, circuit complexity, control voltage behavior, and whether your filter is analog, digitally controlled, or fully digital.

Let's compare the filter types you're considering specifically for a hybrid synth context:

Quick Comparison Table

Filter Type	Pros	Cons	Best For
Notch Filter (band-stop)	Useful for removing specific frequencies or creating phasing effects	Not musically useful as a main filter; lacks character	Special effects, phase cancellation
Cauer / Elliptic	Very steep roll-off, precise	Complex, not musically smooth; phase distortion	Steep filtering in digital realm, not good for analog warmth
Sallen-Key	Simple, smooth, musical, easily modifiable	Harder to voltage-control cutoff smoothly; not scalable to high poles	Classic 2-pole analog filters, gentle slopes
Unity Gain / Unity Drive (often part of OTA/SVF circuits)	Can be designed for CV control; stable gain	Needs careful design for dynamic control	OTA-based VCFs (e.g. LM13700), good hybrid control
State Variable Filter (SVF) <i>(not listed, but important)</i>	Offers LP, BP, HP outputs simultaneously; easy CV control	Needs more op-amps, careful tuning	Flexible analog VCFs in hybrid designs

### Recommendation for a Hybrid Synth

Best Choice: OTA-Based Filter (Unity Gain/Drive or SVF)

If your hybrid synth has digital envelopes, LFOs, or a modulation matrix controlling an analog filter, then you need:

Voltage-controlled cutoff

Voltage-controlled resonance

Musical response

Analog character

So the best fit is:

OTA-based Sallen-Key or State-Variable Filter, using LM13700 or CA3080

Why OTA/Sallen-Key/SVF Wins

Easily controlled by DACs or digital CVs

Well-suited to analog signal path + digital modulation

Proven in many classic hybrid and analog synths (Roland, Korg, Yamaha)

Warm analog tone, and with proper feedback, self-oscillating resonance

Can be exponentially controlled using a digital exponential converter or software  
1V/oct CV → DAC

Why Not the Others?

Cauer/Elliptical:

These are great in digital DSP or audio mastering tools.

But in analog circuits, they're hard to tune, non-linear, and harsh — not musical.

Notch Filter:

Not suited for subtractive synthesis as a main tone-shaping tool.

Good for special effects, but not as the main VCF.

Suggested Filter Designs

1. OTA-Based 4-Pole Low-Pass Filter

(LM13700 + exponential converter)

Warm tone, Moog-style slope (24 dB/oct)

Resonance with self-oscillation

CV-controlled cutoff and Q

2. State Variable Filter (SVF)

(Multi-output: LPF, HPF, BPF)

Use digital CVs to control integrator frequency and feedback

Great for multi-mode filters with analog tone

🔧 Want to Build It?

I can provide:

Schematic for a CV-controlled OTA filter

PCB layout (KiCad, EasyEDA)

DAC control interface using Teensy or STM32

Digital envelope → analog filter example

E.Digital 2. Answer 2:

“Synthesizers, whether hardware or software, are subject to various standards that ensure compatibility, safety, and performance. Below are some key standards that apply to synthesizers:

#### 1. Electrical & Safety Standards

- IEC 60950 / IEC 62368 – Safety standards for audio/video and IT equipment (replaced IEC 60950).
- UL 60065 – Safety requirements for audio and video equipment (U.S. equivalent of IEC 60065).
- CE Marking – Indicates compliance with EU safety, health, and environmental standards.
- FCC Part 15 (U.S.) – Regulates electromagnetic interference (EMI) for digital devices.
- RoHS (EU) – Restriction of Hazardous Substances in electrical equipment.
- REACH (EU) – Regulation on chemical safety in manufacturing.

#### 2. Audio & MIDI Standards

- MIDI (Musical Instrument Digital Interface) Standards:
  - MIDI 1.0 (1983) – Original standard for note and control messages.
  - MIDI 2.0 (2020) – Enhanced protocol with higher resolution and bidirectional communication.
  - General MIDI (GM) – Standardized sound set for compatibility across devices.
  - GM Lite – Reduced version for mobile/embedded devices.
  - MIDI over USB – Standard for USB-MIDI device communication.
  - MIDI over Bluetooth (BLE-MIDI) – Wireless MIDI transmission standard.
- Audio Connectivity:
  - TRS/TS (1/4" or 3.5mm jacks) – Common analog audio standards.

- XLR (AES3) – Balanced audio connections (common in professional gear).
- USB Audio Class (UAC) – Standard for digital audio over USB.
- Dante / AVB – Networked audio protocols (used in high-end setups).

### 3. Software & File Formats

- VST (Steinberg) – Plugin standard for DAWs (Windows/macOS).
- AU (Audio Units, Apple) – Plugin format for macOS/iOS.
- AAX (Avid) – Pro Tools plugin format.
- CLAP (2022) – Open-source plugin standard.
- SF2 / SFZ – Sample-based sound formats.
- MPE (MIDI Polyphonic Expression) – Enhances MIDI for multidimensional controllers (e.g., ROLI Seaboard).

### 4. Environmental & Quality Standards

- ISO 9001 – Quality management in manufacturing.
- IP Rating (e.g., IP54) – Dust/water resistance (relevant for portable synths).
- WEEE (EU) – Waste Electrical and Electronic Equipment recycling compliance.

### 5. Synthesis & Compatibility Standards

- Open Sound Control (OSC) – Alternative to MIDI for networked music control.
- CV/Gate (Control Voltage) – Analog synth control standard (e.g., Eurorack compatibility).
- Eurorack Power (Doepfer A-100) –  $\pm 12V$  power standard for modular synths.

Would you like details on a specific standard?"

E. Deepseek. Answer 1:

MCU (e.g., STM32 Cortex-M)

Pros:

1. Ecosystem & Development Speed

- Rich libraries (CMSIS, HAL), IDEs (STM32CubeIDE), and community support.
- Faster firmware development using C/C++.
- Built-in peripherals: USB, DACs, ADCs, MIDI, GPIOs reduce external components.

2. Cost-Effectiveness

- Lower component cost (especially mid-range STM32F4/H7).
- Simpler PCB design (fewer power rails, smaller packages).

3. Power Efficiency

- Optimized low-power modes (critical for portable designs).

4. Real-Time Control

- Excellent for managing UI, presets, MIDI, and non-audio tasks.
- Predictable interrupt handling for control signals (e.g., envelope triggers).

Cons:

1. Processing Limitations

- Fixed clock speed (e.g., 480 MHz on H7) limits complex DSP (e.g., high-voice polyphony, FFT).
- Sequential processing struggles with parallel tasks (e.g., multiple oscillators + filters).

2. Determinism & Latency

- Interrupt jitter can cause audio glitches under load.
- Software pipelines add  $\mu\text{s}$ -ms latency, problematic for tight analog integration.

3. Precision Constraints

- Limited resolution for high-frequency modulation (e.g., <1Hz FM accuracy).

FPGA (e.g., Xilinx Artix, Intel Cyclone)

Pros:

1. True Parallelism & Determinism

- Run multiple DSP cores (oscillators, filters) simultaneously with nanosecond precision.
- Zero jitter for critical signals (e.g., analog CV/gate timing).

2. High Performance

- Sample-accurate processing at extreme speeds (e.g., 500 MHz+).
- Handle complex algorithms (physical modeling, wavetable scanning) in real-time.

3. Custom Hardware

- Implement exotic synthesis (granular, waveguide) with custom data paths.
- Directly interface high-speed ADCs/DACs for minimal latency.

4. Flexible I/O

- Create custom interfaces (e.g., multi-channel CV, digital oscillators).

Cons:

1. Steep Learning Curve

- Requires HDL (VHDL/Verilog) expertise. DSP implementation is lower-level than MCU libraries.

2. Development Complexity

- Longer design cycles (simulation, place-and-route).
- Limited high-level synthesis tools (e.g., Xilinx Vitis) still require hardware knowledge.

3. Cost & Power

- Higher component cost (FPGA + configuration memory, power regulators).
- Power-hungry vs. MCUs (e.g., 1–5W vs. 100mW).

4. Peripheral Limitations

- Often needs companion MCU for USB/MIDI/UI management.

## Appendix E/ ChatGpt.Answer 3:

A **linear regulator** is a simple voltage regulation device that works by dissipating excess input voltage as heat to maintain a constant output voltage. It operates much like a variable resistor controlled by a feedback loop. Because of its simplicity, it requires only a few external components, making it easy to design and implement. Linear regulators provide a very **clean and low-noise output**, which is ideal for

sensitive analog circuits, audio devices, and RF applications. However, they are inefficient when the difference between input and output voltage is large or when supplying high current, as all the voltage drop is converted directly into heat. This can require large heat sinks and leads to significant power loss.

In contrast, a **switching regulator** is more complex and operates by rapidly switching a transistor on and off and using an inductor, capacitor, and diode to convert voltage efficiently. There are several types—**buck** (step-down), **boost** (step-up), and **buck-boost** (step-up/down or inversion). Switching regulators can achieve efficiencies of 70–95%, even with large input-to-output voltage differences, making them ideal for **battery-powered systems**, **high-current loads**, or applications where **heat and energy waste** must be minimized. However, they generate electrical noise due to high-frequency switching, which can interfere with sensitive circuits if not properly filtered.

In choosing between the two, use a **linear regulator** if your system has a small input-output voltage difference, low current demand, and requires low output noise. They are ideal for low-power applications where simplicity, low cost, and clean power matter more than efficiency. On the other hand, choose a **switching regulator** when efficiency is critical, especially in systems with high current draw or a large gap between input and output voltages. While more complex and potentially noisier, switching regulators are far more power-efficient and suitable for modern portable and high-performance electronic systems.

If you have a specific application in mind, such as your input voltage, desired output, and load current, I can help you decide exactly which type to use—or even recommend a specific regulator.

E. DeepSeek. Answer 3:

“Selecting the right op-amp for analog audio devices depends on factors like noise, distortion, bandwidth, power supply requirements, and cost. Here are some of the best op-amps for audio applications:

#### Top Op-Amps for Analog Audio:

##### 1. Burr-Brown (TI) OPA Series (High-End Audio)

**OPA1611/OPA1612** – Ultra-low noise (1.1 nV/ $\sqrt{\text{Hz}}$ ), low distortion, excellent for preamps, mixers, and DACs.

**OPA2134/OPA2604** – FET-input, low noise (8 nV/ $\sqrt{\text{Hz}}$ ), warm sound, great for guitar pedals and Hi-Fi.

**OPA1656** – Very low noise (2.9 nV/ $\sqrt{\text{Hz}}$ ), low distortion, ideal for high-end audio.

## 2. Analog Devices (High Performance)

**AD797** – Ultra-low noise (0.9 nV/ $\sqrt{\text{Hz}}$ ), best for phono preamps and sensitive audio stages.

**AD8599** – Low noise (1.1 nV/ $\sqrt{\text{Hz}}$ ), low distortion, good for DACs and ADCs.

**AD8620** – Zero-drift, low noise, excellent for DC-coupled audio circuits.

## 3. Texas Instruments (General Purpose & High Performance)

**NE5532 (Classic)** – Low noise (5 nV/ $\sqrt{\text{Hz}}$ ), high drive capability, widely used in mixers and consoles.

**LME49720 (Discontinued, replaced by LME49990)** – Ultra-low distortion, high-end audio.

**LM4562 (Similar to LME49720)** – High performance, low noise (2.7 nV/ $\sqrt{\text{Hz}}$ ), great for Hi-Fi.

## 4. NJR (New Japan Radio)

**NJM4556** – High output current, good for headphone amps.

**NJM2068** – Low noise (5 nV/ $\sqrt{\text{Hz}}$ ), similar to NE5532 but with better performance.

## 5. STMicroelectronics

**TL072/TL082 (FET-Input, Budget Option)** – Decent noise (18 nV/ $\sqrt{\text{Hz}}$ ), good for guitar effects and non-critical audio.

### Key Considerations:

**Noise:** For mic preamps, choose ultra-low noise (<5 nV/ $\sqrt{\text{Hz}}$ ) like OPA1611 or AD797.

**Distortion:** For Hi-Fi, use ultra-low THD op-amps like LME49720 or OPA1656.

**Output Drive:** For headphones, NJM4556 or OPA1622 (high current).

**Budget:** NE5532 or TL072 for cost-sensitive designs.

**Single vs. Dual Supply:** Check if the op-amp works with your voltage range (e.g., ±15V or +5V).

**Best by Application:**

**Mic Preamps:** OPA1611, AD797

**Headphone Amps:** OPA1622, NJM4556

**Guitar Pedals:** OPA2134, TL072

**Hi-Fi DACs/Amps:** LME49720, OPA1656

**Budget Audio:** NE5532, NJM2068

Would you like recommendations for a specific circuit?"

Appendix F – etc.