ModuLoop

Conor Curtin CompE, Shawn Colby CompE, Ryan Gordon EE, Evan Rovelli CompE
Source Code & Design Files: github.com/cpcurtin/SDP-Team-28

Abstract—Music creation has evolved over time and a lot of new technology has been incorporated into this process. There are many solutions designed to create backbeats and loops to aid musicians in creating music. However, a lot of these tools only have set features and are quite confusing for an inexperienced user to operate. We designed an open-source solution that is more modular in its features and easier for that user to operate. Our device is able to have varying steps for each beat in a looping measure. There are different sounds and effects that can be swapped by the user to achieve various styles of music. The loops are chainable, so multiple measures of music can be made.

I. INTRODUCTION

The world of musical design and creation is an expression of the skills and talent of musicians that can be traced back to ancient civilizations thousands of years in our past. The first composed musical phrase dates back to 14th century B.C., and was written in cuneiform. Many consider cuckoo clocks to be the first music sequencer, invented all the way back in the 1600s. Since the invention of the first digital sequencers in the 1970s, they have played a vital role in music creation, allowing musicians to generate complex rhythms and beats to be used standalone or in other devices, like a synthesizer. While these devices have grown more powerful with advancements in electronics technology, many examples are quite large and not intuitive to learn. Our goal for this design was to create a device that lessens the learning curve for more novice musicians while also making the device more portable.

A. Significance

Music creation is a complex art form to learn. Between the size and non-intuitive nature of this kind of device, it can be very challenging for a novice user to get over the learning curve and down to making music. Furthermore, the more entry-level of these devices utilized fixed sound libraries and beat timing, limiting the capabilities of the tools most accessible to new users. With members of our team having experience with sequencers and having copious amounts of friends that also make music, we understand the challenge that comes with using these devices.

B. Context and Survey of Similar Solutions

In today's market, hardware sequencers that seamlessly converge ease of use, mobile accessibility, sound customization and effects are not available to musicians without serious drawbacks such as required companion software, predetermined step sequences, or price.

In the past, musicians relied on dedicated hardware sequencers, such as the Roland TB-303 or the Akai MPC series. These machines allowed users to input and manipulate musical sequences directly using physical interfaces. While limited in functionality compared to modern software sequencers, they were powerful tools for music production.

With the advent of personal computers, software sequencers became increasingly popular. Programs like Steinberg Cubase, Ableton Live, and Propellerhead Reason offered flexible and comprehensive solutions for music composition and sequencing. These software sequencers provided features like MIDI editing, virtual instrument integration, and automation.

The Pocket Operator by Teenage Engineering

The Pocket Operator is a compact yet powerful micro sampler and sequencer, boasting a 16-step sequencer that's perfect for music production on the go. With two 3.5mm connectors and a clear LCD screen, it offers intuitive control and connectivity. This pocket-sized wonder comes with 16 distinct sounds split into eight melodic sample slots and eight drum slots, providing a versatile sonic palette for your creativity. What sets it apart are the 16 onboard effects, from octave shifting to stutter and quantization, allowing precise sound manipulation. It's a portable music-making companion with a built-in speaker and is powered by two AAA batteries. Measuring a mere 1.2 x 2.4 x 4.9 inches and weighing only 0.2 pounds, the Pocket Operator fits comfortably in your pocket, making it an essential tool for both seasoned producers and newcomers looking to delve into electronic music.

Type:	Micro sampler & sequencer
Sequencer:	16 step sequencer
I/O:	2x 3.5mm connector, LCD screen
Sounds:	16x (8 melodic sample & 8 drum slots)
Effects:	16x (Octave up & down, stutter, quantize,...)
Speaker:	Built-in
Power Source:	2x AAA Batteries
Size:	1.2 x 2.4 x 4.9 Inches
Weight:	0.2 LB

Maschine Mikro MK3 by Native Instruments

The Mikro MK3 is a versatile pad controller, sequencer, and sampler in a single package, offering both step sequencing and real-time recording sequencing capabilities for synthesis of intricate beats and melodies. It has a singular point of connectivity for both power and data transfer over USB-B and features an LCD screen for sound and effect tag selection. The MK3 contains a wide range of sound options from samples, one-shots, loops, instruments, drum kits, bass, and drum synths as well as 33 onboard effects like stutter, burst, reverb, and compress-ion. The device provides comprehensive tools for sound manipulation. While it lacks a built-in speaker, it's powered via USB, making it a portable and convenient option for music producers. With compact dimensions of 1.77 x 12.6 x 6.96 inches and a weight of 2.47 pounds, the Mikro MK3 is a versatile choice for both experienced and novice music creators. The MK3 has an additional software companion app, MASCHINE Essentials, for advanced sampling functions, VST and AU plug-in hosting, effects sidechaining, and a DAW-style mixer.

Type:	Pad Controller with Sequencer and Sampler
Sequencer:	Step Sequencer, Realtime Recording Sequencing
I/O:	USB 2.0-B, LCD, velocity-sensitive button pads, Dual-touch Smart Strip, 39 controlling buttons, and one endless rotary encoder.
Sounds:	Samples, one-shots, loops, instruments, drum kits, bass/drum synths
Effects:	33 total Effects (stutter, burst, reverb, compression,...)
Speaker:	None
Power Source:	USB bus powered
Size:	1.77 x 12.6 x 6.96 Inches
Weight:	2.47 lbs

C. Societal Impacts

ModuLoop was designed to improve on these similar devices. The main audience for our device is individuals who make music currently with sequencers and individuals who want to make music with sequencers. ModuLoop allows any user to make loops for their application. Users don't have to spend an inordinate amount of time to get used to a new device and it doesn't lock them into preset sounds. That is what we focused on as we designed ModuLoop.

D. Goals, Specifications and Testing Plan

Specification	Test Plan
Ensure tempo precision within a ±2% tolerance across the entire range of step intervals.	Measure the delay between steps and view output on an oscilloscope to measure the distance between peaks
High and low pass filters can be applied to the final audio mix	Read the effect of HPF & LPF in the frequency domain using the FFT oscilloscope feature
BPM is adjustable through a user input	Measure the delay between steps and view output on an oscilloscope to measure the distance between peaks
A graphical user interface can be used to navigate preset & custom sound libraries and saved pattern configs	Navigate sound libraries and saved configurations
The system is operable by both rechargeable battery and power supply	Independently demonstrate system powered by external power supply and battery
Allow users interchangeable custom sound files	Play custom sound files from external storage

Achieve at least four sounds layered per step	Mix a minimum of four sounds concurrently
Preset and custom sounds can be mixed	Output mixed preset and custom sounds
System outputs to built-in speakers or line-out	Display system sound through headphones or speakers
The device under the dimensions of a large tablet (under 13x13 inches)	Footprint dimension comparison

Table 1: Design Specifications

II. DESIGN

A. Overview

We solved our problem by creating a sequencer that can use chosen sounds and effects imported into our device. We also created the ability to have more than four steps in a beat with a much more straightforward user interface. This strongly increases the user customizability of this product which is the problem that was seen almost every other on the market sequencer we researched. One of our design alternatives was to not have a pallet button grid and just map sounds to the measure matrix buttons. However, we realized this would make it more complicated to operate and less user friendly. We also considered not using a screen and having a visual interface through a computer application. However, this requires another connected device for the user and is harder to use/travel with. We expected that our product would solve the problem we had because some of our group members have had experience with using similar products. They found other alternatives limited and hard to pick up for a new user. We knew by simplifying the way to operate our device while adding customizability, the main issues we saw in other products would be solved.

When creating our specifications, there were some tradeoffs to be made. We wanted to apply filters to each individual sound file so you could adjust how each sound worked independently. We realized when trying to import sounds and layer them from different storages it would be significantly more work for a small feature so we decided to apply the filter over the whole system opposed to individual sound files.

Hardware Block Diagram:

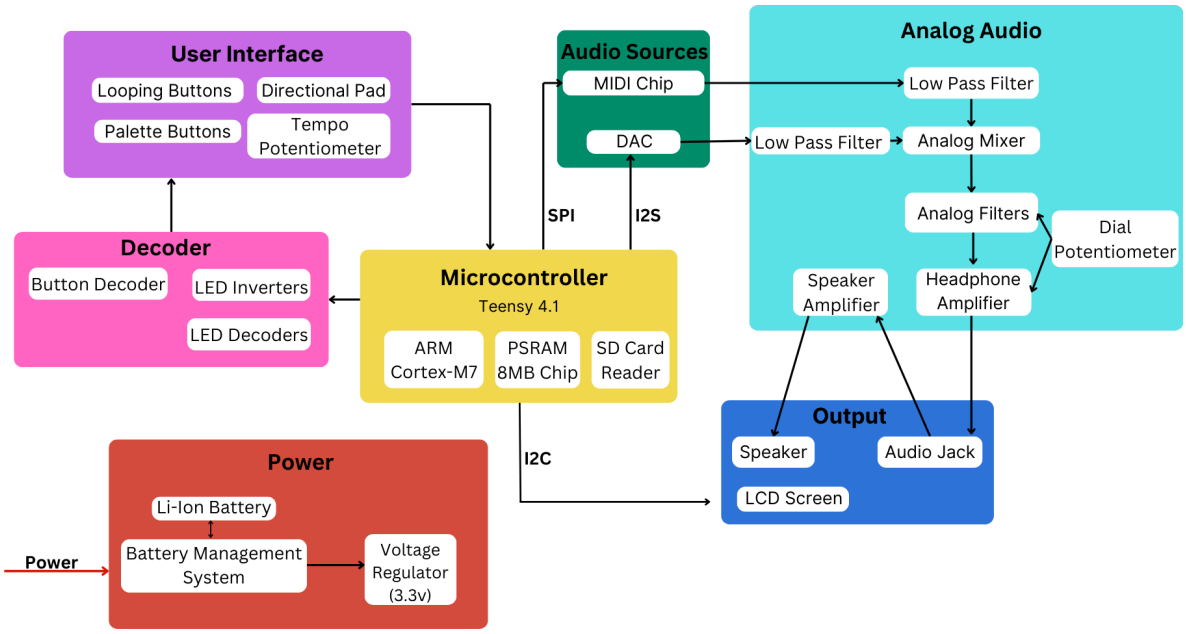

Figure 1: Hardware Block Diagram

In figure 1, our hardware diagram is set up with inputs on the left and outputs on the right.

1. Power: The red battery power block shows how we get wall power to charge our onboard batteries for the device as well as distribute power to our system. There is a Li-Ion battery which sends power through the battery management system(BMS) out to our 3.3v voltage regulator. If power is plugged into the BMS it will charge the Li-Ion battery while sending power to the voltage regulators for our system.

2. User Interface: The purple user interface box is user inputs that interact with the Teensy 4.1. These are either buttons to control the measure matrix or palette and the directional-pad to control the LCD screen. These are all inputs that are sent to the microcontroller which decides what is activated next. There is also a tempo potentiometer which provides an adjustable voltage which the Teensy 4.1 reads to adjust the tempo

3. Teensy 4.1: The yellow box is the Teensy 4.1 where all the processing is done. The Teensy 4.1 has an ARM Cortex-M7 processor with a 600MHz clock that enables us to create accurate timing. If any buttons related to sounds/effects are hit, the microcontroller sends a SPI or I^2S up to the audio sources to activate a sound. If the directional-pad is used, the microcontroller sends an I^2C signal right to the LCD in the output. There are also signals sent out through the GPIO pins of the Teensy 4.1 to interact with the decoders for the buttons/LEDS. This is to scan the buttons for inputs and activate the LEDs.

4. Audio sources: The green box is the audio source that outputs an analog signal. The MIDI chip gets sent a string of numbers linked to a specific sound over SPI from the Teensy which outputs as an analog signal. It is sent into a low-pass filter in the analog audio section to reduce noise. The DAC receives an I^2S signal from the Teensy which contains the signal for a specific sound file that was put on the SD card. The signal is converted to analog and then output into a low pass filter in the analog audio to reduce noise.

5. Analog Audio: The teal box is the analog audio where all of the analog signal manipulation happens. Any sound files are sent through a DAC to convert the signal to analog. Then the MIDI chip signal and DAC signal are both sent through their own low pass to reduce noise. They are both sent through the analog mixer to combine them into a single signal. From here we apply our analog filter to adjust frequency, lowpass, and highpass as desired by the user. Potentiometers are used for each filter to adjust how much the filter affects the audio signal. Then, this complete signal is sent through an LM386 amplifier to allow for more user volume control. There is another potentiometer on the amplifier used to adjust the volume. This completed signal is then sent to an output device.

6. Output: The blue box is the output of our system. The signal leaves the analog audio and can either be output through an audio jack if headphones are plugged in, or through the onboard speaker if the audio jack is empty. There is a disconnect in the headphone jack that allows the signal to pass through to the speakers only if no headphones are plugged in. If there are no headphones plugged in, the audio signal goes back into the analog audio section and into a TPA2016D2 amplifier. This is sent to the onboard speakers which allows us to send a larger signal through our onboard speakers but not as much through the headphone jack to achieve the volume output that would balance well. This is how our device outputs the measure and its audio signal. If the directional pad is used to operate the LCD screen, the screen will output accordingly displaying the information requested by the user.

Software Block Diagram:

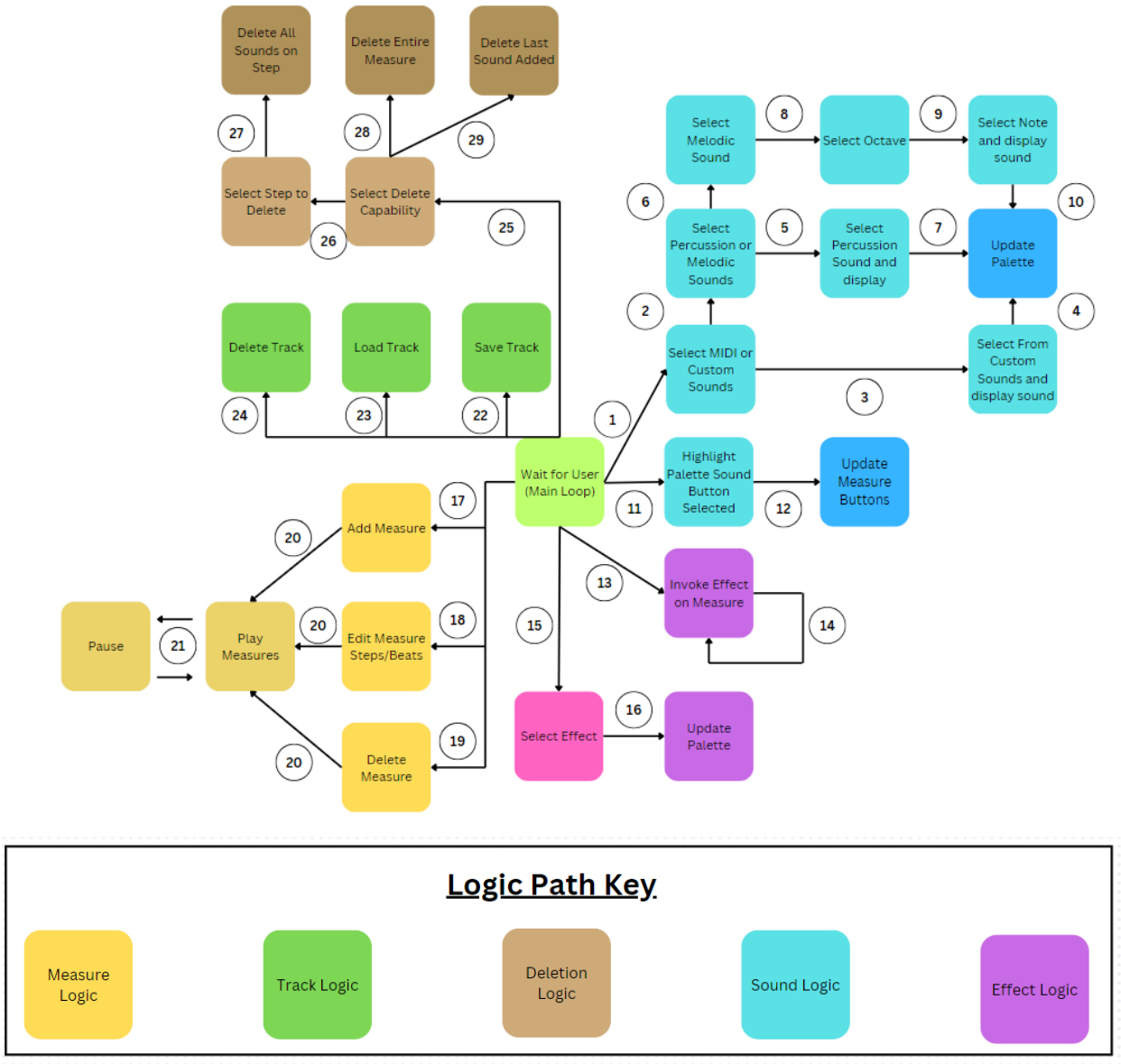

Figure 2: Software Block Diagram

1. Wait for user (main loop): When the user turns on ModuLoop, the software logic will start at this block and visit it at every clock cycle. There are five possible logic paths that can begin downstream from this block. These five sections include the sounds section (blue), effects section (pink and purple), measure section (yellow), track section (green), and deletion section. If the user selects sounds through the display, the logic will follow path (1) in the sound logic section. If the user selects a palette button containing a sound first, then the logic will follow path (11). If the user selects a palette button containing an effect the logic will follow path (13) in the effect logic section. If the user selects effects through the display first, then the logic will follow the path (15). If the user selects to add a measure through the display, then the logic will follow the path (17). If the user selects to

edit the steps and beats in a measure through the display, then the logic will follow the path (18). If the user selects to delete a measure through the display, then the logic will follow the path (19). If the user selects to save a track through the display, then the logic will follow the path (22). If the user selects to load a track through the display, then the logic will follow the path (23). If the user selects to delete a track through the display, then the logic will follow the path (24). If the user selects the delete capabilities through the display, then the logic will follow the path (25).

2. Highlight palette sound button selected: The button that the user selected on the palette will be highlighted. Once the user then selects a measure matrix button, the LED associated with the palette button will turn off and the logic will follow path (12).

3. Update Measure Buttons: The sound that was associated with the palette button selected is copied and assigned to the measure matrix button selected. The logic then returns back to the beginning of the sounds logic section

4. Select MIDI or Custom Sounds: If the user selects MIDI sounds through the display the logic will follow path (2). If the user selects custom sounds, the logic will follow path (3).

5. Select From Custom Sounds and Display Sound: Once the user selects a custom sound from the display, the LEDs on the measure matrix will turn off (with the sounds still playing) and the user will be prompted to select the palette button they wish to assign the selected sound to. Once they do that, the logic will follow path (4).

6. Select Percussion or Melodic Sounds: If the user selects the melodic category, the logic will follow path (6). If the user selects the percussion category, the logic will follow path (5).

7. Select Percussion Sound and Display: Once the user selects a percussion sound from the display, the LEDs on the measure matrix will turn off (with the sounds still playing) and the user will be prompted to select the palette button they wish to assign the selected sound to. Once they do that, the logic will follow path (7).

8. Select Melodic Sound: Once the user selects which melodic sound they want, the logic will follow path (8).

9. Select Octave: Once the user selects which Octave they want to play their sound in, the logic follows path (9).

10. Select Note and Display Sound: Once the user selects a note from the display, the LEDs on the measure matrix will turn off (with the sounds still playing) and the user will be prompted to select the palette button they wish to assign the selected sound to. Once they do that, the logic will follow path (10).

11. Update Palette (sounds logic section): The sound that was previously selected through the display is assigned to the palette button that the user selected then the logic returns back to the beginning of the sounds logic section.

12. Invoke Effect on Measure: The effect associated with the selected palette button will be invoked on the measure matrix until the user releases that button. Once they do, the logic will return back to the beginning of the effects logic section.

13. Select Effect: Once the user selects the desired effect from the display, the LEDs on the measure matrix will turn off (with the sounds still playing) and the user will be prompted to select the palette button they wish to assign the selected effect to. Once they do that, the logic will follow path (16).

14. Update Palette (effects logic section): The effect that was previously selected through the display is assigned to the palette button that the user selected then the logic returns back to the beginning of the effects logic section.

15. Add Measure: Once the user selects the add measure option through the display, a blank measure will be added to the end of the measure chain and the logic will follow path (20).

16. Edit Measure Steps/Beats: Once the user selects how many steps per beat and beats per step they want for the current measure being edited, the logic will follow path (20).

17. Delete Measure: Once the user selects the delete measure through the display, the measure currently playing will be deleted and the logic will follow path (20).

18. Play measures: The sounds associated with each measure matrix button will play sequentially at the desired tempo the user sets the system to in BPM. Measures will be played in the order specified with the correct number of beats per step and steps per beat in each measure. Once the pause button is pressed, the logic will follow path (21).

19. Pause: The logic will remain in the pause block until the play button is pressed. The sound produced by the measure matrix is stopped while in this logic block. Once the play button is pressed, the logic will follow the path labeled (21) back to the Play Measures logic block.

20. Select Delete Capability: If the user selects to delete the last sound from the display the logic will follow path (29). If the user selects to delete the whole measure from the display it will follow path (28). If the user selects to delete all sounds associated with a step from the display, the logic will follow path (26).

21. Select Step To Delete: The user will be prompted to select a button on the measure matrix which they want all sounds to be deleted from. Once the user selects this button, the logic will follow path (27).

22. Delete all Sounds on Step: All sounds associated with the selected step will be removed from the measure and the logic will return back to the beginning of the deletion logic.

23. Delete Entire Measure: All sounds associated with every step in the measure will be removed and the logic will return back to the beginning of the deletion logic.

24. Delete Last Sound Added: The last sound that was added to the measure will be removed and the logic will return back to the beginning of the deletion logic.

B. SD Sounds

Custom sounds are imported into the system via the Teensy 4.1's built-in micro-SD socket. This built-in SD socket uses fast 4-bit native SDIO to access the card to populate the navigation system with available sound files. The SD card is formatted to FAT32 for greater compatibility with our system and user devices for storing custom sound files.

Uncompressed RAW sound files of type 16 bits/sample, 44100 samples/sec, signed integer samples must be used for the system. RAW sound files are uncompressed sound files without header information, this is canonical with a WAV file stripped of its 44-byte header. The system is functional if given WAV files of 16 bits/sample, 44100 samples/sec, signed integer samples, but it treats the header data as sample data, thus producing unwanted noise when a WAV file is played by playing the illegible header data.

Due to hardware and software limitations of SD card media and the Arduino SD library, the onboard micro-SD card is not capable of simultaneous SD playback. This is because many class 10 SD cards are only fast for sequential access. When 3 or 4 files are read, one block at a time, the controller inside the card can add terrible latency, which stalls the audio library. This limitation prevents us from achieving our four-sound polyphony for custom sounds solely by interfacing directly with the SD card as playback latency and polyphony are critical features.

Our solution to solve this problem and achieve four custom sound polyphony within latency constraints was by soldering an 8-pin QSPI 8MB external pseudo-static random-access memory module known as PSRAM directly to the Teensy 4.1 to cache actively assigned custom sounds. PSRAM offers fast access times like SRAM while also exhibiting high memory density and low-cost DRAM while operating at a low power consumption. The selected PSRAM module can be accessed via QSPI at a clock frequency of 133MHz, and each data transfer operation involves transferring 32 bytes of data in a wrapped burst mode. Once a custom sound file is selected to be used in the system, it is cached to the external PSRAM and accessed by the PT8211 DAC over I^2S. The Teensy Audio Library assigns the I^2S controller DMA to transmit blocks of sample data to the DAC whilst allowing the main program to execute without halting.

To import custom sounds into the system using an SD card, the uncompressed RAW 16 bits/sample, 44100 samples/sec, signed integer sample files must be placed within a root directory named "sounds" on the FAT32 formatted SD card. Upon system start all custom audio

files stored in this directory will be parsed and made available from the GUI and accessible to the user for palette assignment.

C. MIDI Sounds

The MIDI chip selected was the VS1053b from VLSI Solution. This chip has a full implementation of General MIDI 1 melodic and percussive sounds, as well as a partial implementation of General MIDI 2 percussive sounds. In the General MIDI standard, each instrument is given a dedicated number that can be used to recall that sound later on. In this chip, these sounds were implemented in two separate banks of memory: Melodic (default) & percussive.

The VS1053b is interfaced over SPI at a 31,250 baud rate with only the TX from the Teensy. There are hexadecimal values for each of the commands supported by the chip. The ones used here are message send, program, note on, note off, sound bank select, and volume select, all with corresponding values that can be found in the datasheet for the chip. The VS1053b can run sound over 16 channels, addressed as a one-bit hex value. We decided to use dedicated channels depending on what type of sound was being played. Channel zero is reserved for sounds in the percussive bank and channel one is reserved for sounds in the melodic bank. The chip identifies what channel a message is being sent to by the OR of that value plus the hexadecimal value for the message command. Based on this, to select a message bank, a series of three serial messages are sent. First the message command OR'd to the desired channel number, then the bank select character, then finally the memory address of that bank. Similar processes exist for selecting notes, enabling notes, and changing the output volume of the chip. For this implementation, the max volume (127) was used so that the control could be handled externally. Once the sound bank is set, a user can send repeated commands to enable specific instruments within this bank without having to reselect the bank.

To access all of the sounds on the VS1053b, there is a library of all potentially selected sounds that is integrated with the user display, so as users select sounds defined there, they can be applied with the appropriate commands to the chip to play the specific sound(s). This command sequence is implemented within the timing system to time each of the commands for the selected step.

D. Analog Mixing

ModuLoop has two audio sources that output analog audio signals. The VS1053b natively outputs audio from its internal DAC, requiring no external components. The SD sounds we used are fed from the Teensy 4.1 to the Princeton Technologies PT8211 DAC via I^2S, where we convert from digital to analog audio.

At the output of each audio source, there are identical second-order Sallen-Key active low-pass filters with Butterworth response, whose active stage is built around the Texas Instruments NE5532 op-amp. This selection is detailed in Appendix G. These filters have a cutoff frequency f_c = 22kHz. These filters are being used to filter high-frequency noise that can be generated from these sources/DACs. The design of these filters is shown in Figure 3.

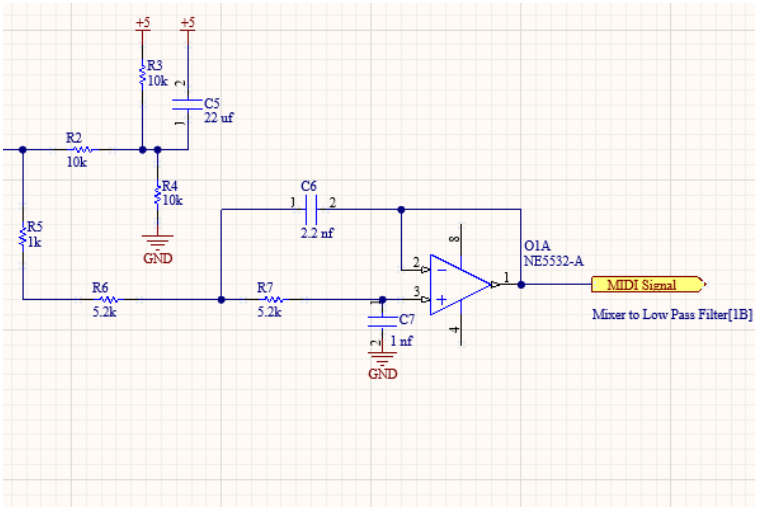

Figure 3: Audio source output filter design.

Each of these filters is implemented close to the output of the audio sources to remove the noise as early as possible in the signal path. Based on the physical layout of components on our PCB, each filter has its own NE5532. Because each of these ICs has two op-amps in them, the unused amp is tied off according to [8].

After each of the low-pass filters, the two audio signals are mixed. We decided upon mixing over resistors to reduce the complexity of the system and to reduce the noise that is introduced through each active stage. In this case, there are 100Ω resistors added in series with the signal that are connected on one end, connecting the signal paths before moving further through the circuit.

E. Analog Filters

We implemented hardware high-pass and low-pass filters in the analog signal path to allow a user to apply them when and how they see fit. In designing these filters, we utilized Analog Devices' Filter Design Tool [6], which allows you to design filters based on their cutoff frequencies and simulate different characteristics like the frequency response and phase shift over the frequency range of the filter. The first in the signal path is the low-pass filter. This filter, shown in Figure 4, shows our final design. The actual filter is implemented on op-amp O3A. This is a second-order active Sallen-Key low-pass filter with Butterworth response and a cutoff frequency f_c = 300Hz. The active stage of this filter is the Texas Instruments NE5532. Through this filter, there are two signal paths. The first, which passes through O3A and O4A is the filtered signal path. First, it passes through the filter, then passes through a buffer stage to boost the signal strength and isolate the filter from the rest of the design. The second signal path, which passes along the bottom through O3B, is the unfiltered signal. Both of these signals are passed into opposite ends of a 10k Potentiometer, which is acting as a blend pot to mix the two signals together. This can be thought of as a wet-dry mix. This is how the user can apply the filter. The cutoff frequency is not changing, only the blend. In practice, this design gives the perception of an adjustable cutoff frequency audibly. From here, the signal passes through another buffer, O4B, to again boost the signal and separate the low-pass filter from the high-pass filter.

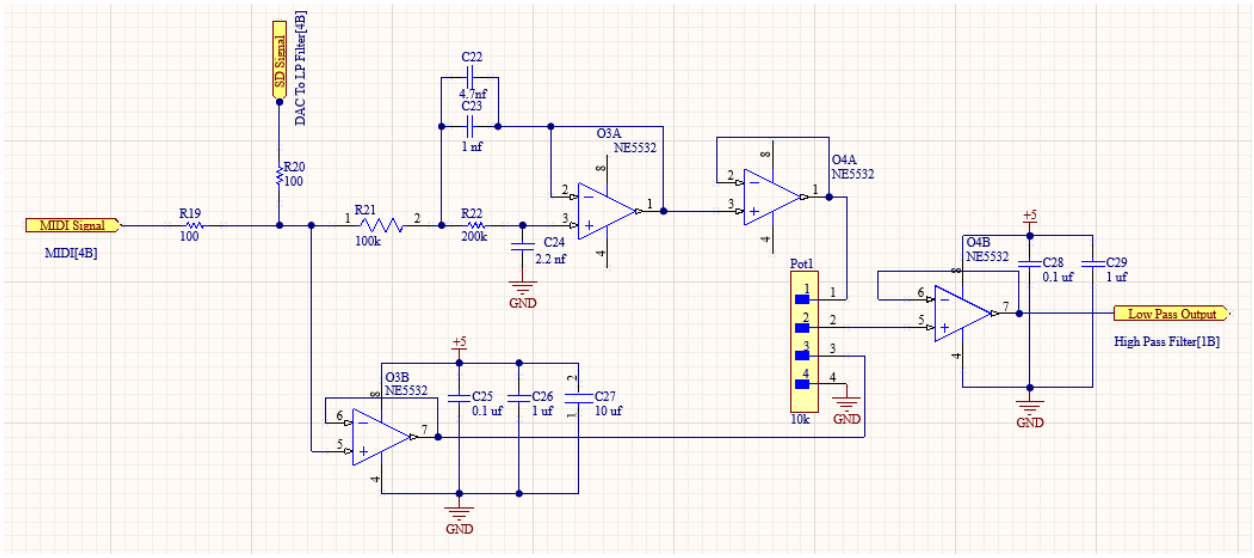

Figure 4: Low-pass filter design.

The next stage in the signal path is the high-pass filter. This filter is a second-order active Sallen-Key high-pass filter with Butterworth response with a cutoff frequency f_c = 3kHz. The active stage of this filter is the Texas Instruments NE5532. Through testing, the traditional layout of the Sallen-Key design was blocking the signal flowing through R24. To solve this issue, we flipped the positions of R23 and R24, moving the grounded resistor away from the input of the op-amp. The topography of this filter is the same as the low-pass filter, with a filtered path running through O5A and O6A, and an unfiltered path through O5B that meet at either end of a 10k Potentiometer. The signal path then passes through another buffer to boost the signal and isolate it from the amplifiers that come after.

Through these filters, there is significant signal degradation, reducing the output signal strength from the system. To combat some of this, we added an active gain stage with g = 2.72. This boosts the signal back up close to its initial strength pre-filtering. This allows us to maximize the functionality of our power amplifier on the output signal.

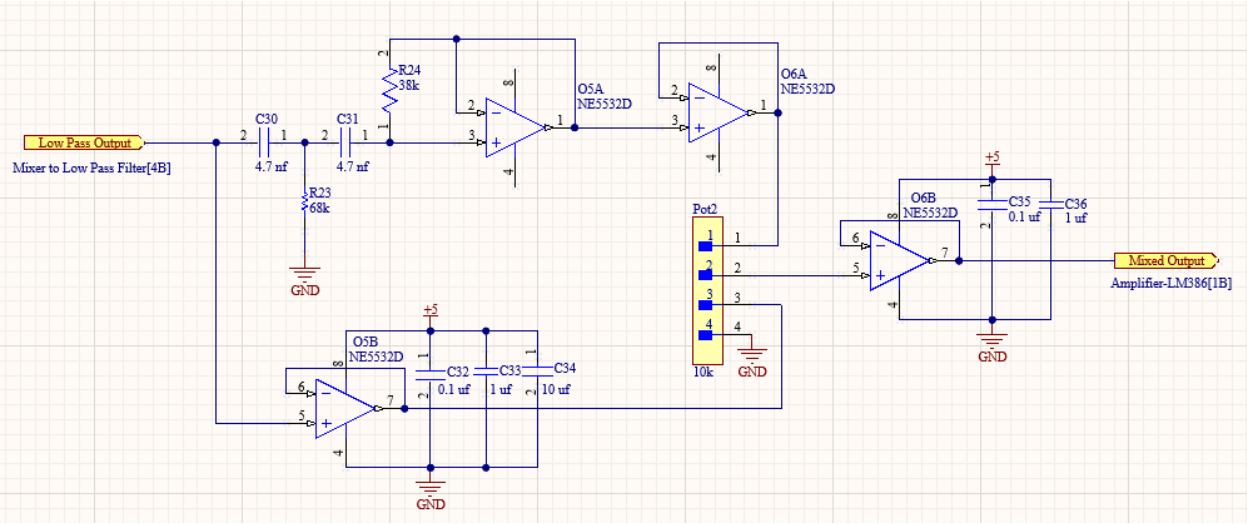

Figure 5: High-pass filter and gain stage.

F. Analog Amplifier

To bring the output signal up to a level suitable for headphones, we are using the Texas Instruments LM386. This is a versatile power amplifier that is suitable for use in low-power and small-form factor systems. This chip was selected because it can function on the 5V supply that our system is using, and it requires significantly fewer external components than other similar amplifiers, like the Texas Instruments LM384 or LM380. Figure 6 shows our implementation of the LM386. The components selected to supplement this filter are based on an implementation from its datasheet [13].

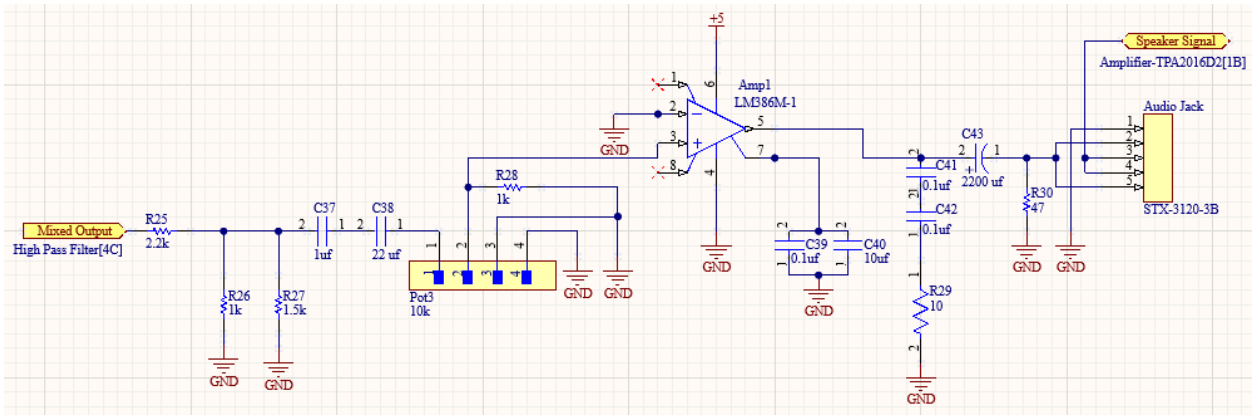

Figure 6: LM386 Amplifier

We are using the internal gain setting of g = 20. This chip has the capacity to increase the gain, but with that comes tradeoffs in increased noise, distortion, and potential oscillation/undesired behavior of the amplifier, so we chose to not increase it. We added 10uf and 0.11uf capacitors on pin 7 to help balance the chip. This is not required when using the default gain setting but is recommended for higher gain settings. For our application, we increased the size of the output capacitor C43 from 220µf to 2200µf, and we added a 47Ω

resistor on the output to simulate a load on the amplifier. In testing with the second amplifier, the LM386 behaves out of the ordinary, so these modifications were made to work around this IC's limitations.

On the input of the amplifier is a small high-pass filter to help with noise degradation before being fed into the volume control potentiometer. This is a 10k ohm potentiometer. The response of the human hearing is logarithmic, but the potentiometers we are using are linear. Because of this, we added a 1kΩ resistor that bridges the wiper to ground. Doing this converts the response of the potentiometer to respond in a pseudo-logarithmic or law-log way.

The output of this amplifier can take two paths. If something is plugged into the headphone jack, the signal will output there, and will not continue on. Otherwise, the signal continues into the second amplifier circuit. After the headphone jack, the signal passes through a differential converter that converts the single-ended signal that is used in the rest of the circuitry to a differential signal. This converter is implemented using the Texas Instruments NE5532 op-amp, shown as O7A and O7B. This is then fed into both channels of the Texas Instruments TPA2016D2. [16] More information about this amplifier can be found in Appendix H. The schematic for this amplifier was based off the Adafruit breakout board schematic, which can be found in [24]. The output of this amplifier is then fed into two 8Ω 5W speakers mounted to the front of the case.

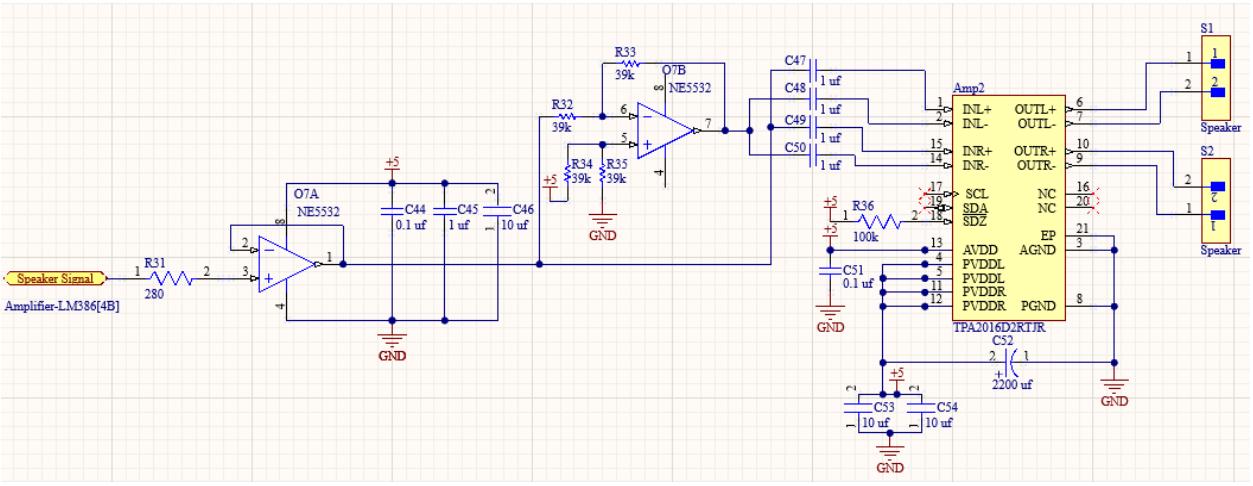

Figure 7: TPA2016D2 Amplifier

It was decided to use two amplifiers for a couple of reasons. For one, the LM386 does not boost a signal enough at a 5V supply to give the desired output volume of the system. Second, putting the headphone jack after the TPA2016D2 caused concern about the signal being too strong for headphones, potentially causing headphone damage and hearing damage. Therefore, we decided to use two amplifiers: one primarily for headphones and primarily for the speakers to give the system versatility in its output capability while maintaining non-destructive audio levels for a user that chooses to use headphones.

G. Graphical User Interface and Navigation

The system's graphical user interface is implemented using a four-row by twenty-column character liquid crystal display using I^2C. The display is navigated using a four-button directional pad.

The user interface is constructed following model view controller (MVC) design principles for predictable and reliable development and behavior. Figure 7 below outlines an example of MVC relational behavior, although not based on our system.

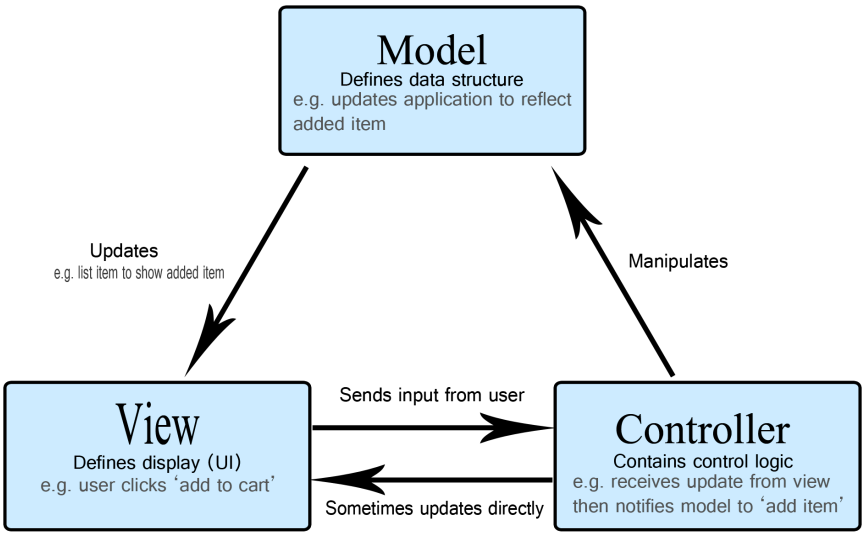

Figure 8: MVC Example [22]

In our system, directional pad inputs are read by the controller which signals the model to perform actions based on the current state of the navigation data structure. The model is constructed by a tree data structure of custom C structs. Each instance of the data structure contains an array of all display items, a sub-array of four active display items corresponding to the LCD's four rows, an index associated with the position in the main array of display items which is used to determine the sub-array, and lastly relational links between parent and child structs. The sub-array of a navigation struct is used as the view for the system and is dynamically populated by the main array. Additionally, depending on the navigation state, the bottom row of the sub-view array is dominated by a system statistic banner displaying relevant information regarding active measure tempo, and global step per beat settings.

The product of MVC integration with the directional pad and LCD allows for dynamic navigation through midi sounds, custom sounds, and digital effects for custom sounds for palette assignment.

H. Button Matrix

The buttons and LEDs for the measure matrix and palette are set up in a row-column address. This allows us to use less GPIO pins to connect our buttons than if we wired them all individually. We then have the button and LED columns hooked up to decoders. This lets us reduce the total GPIO pins we need even more. The buttons have rows that are set as input pullups and the columns are all set high. The decoder goes through and sets each column low

one at a time. If a row turns low during this, we are able to determine there was a button push based on which row went low and which column was low at the same time. For the LEDs, the rows are set high and the columns are set low by using the decoder connected to an inverter. The invert makes the column go high when selected. If an LED needs to be turned on the corresponding row is set low and the column is set high. This allows the LED to turn on. By using a row-column address as well as decoder we reduced the amount of GPIO pins needed from 26 to 18. Figure 8 shows a schematic of the buttons decoders and inverter. You can see decoder 2 has all of its output connected to inverters which allows for a high output when selected for the LEDs.

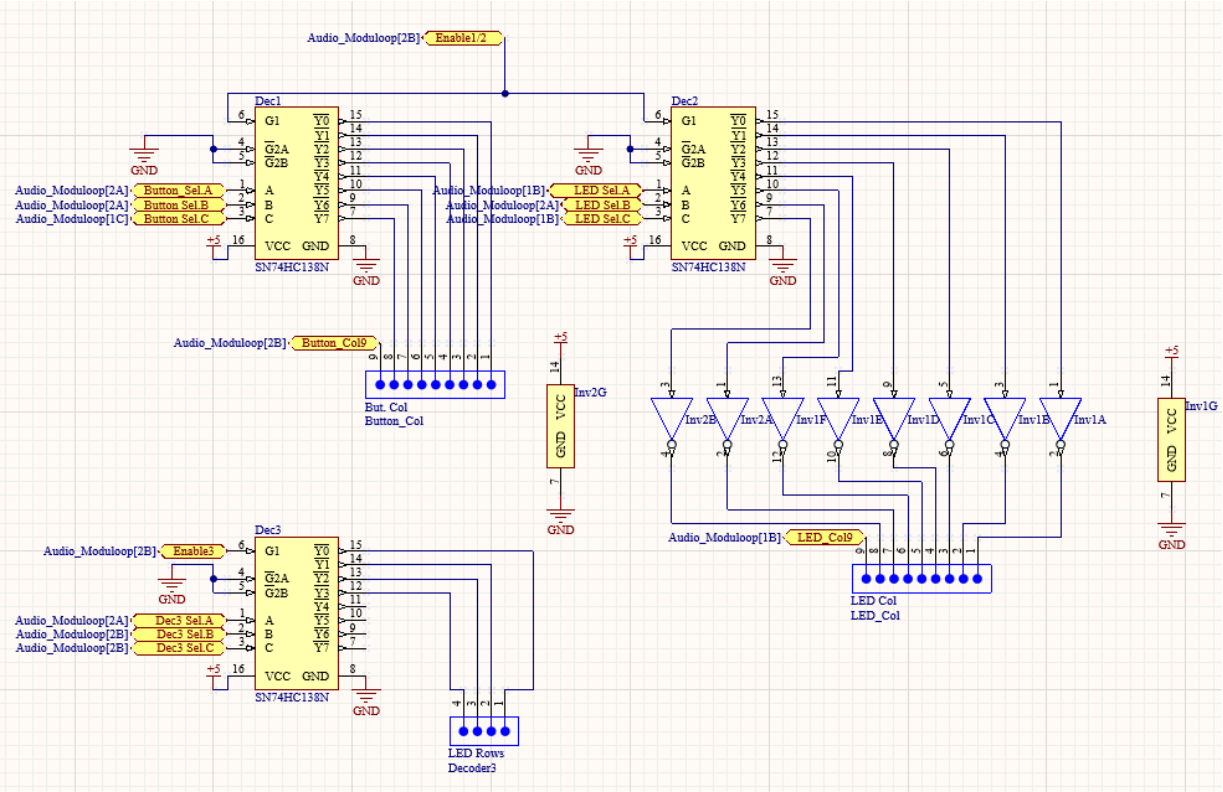

Figure 9: Decoders and inverter for measure matrix and palette

I. Timing

The timing of this system is extremely critical for any music sequencer. For our project, we want the timing of our measure matrix to align with a typical metronome clock. This is adjustable to the beats per minute of the matrix and dependent on steps per beat. To accomplish this goal of timing, we utilize the metro library to interface the onboard real time clock to set software traps indicating a desired timing interval has elapsed. We configure this trap's desired interval based on the active measure and beat played as each beat may be configured to have one to six steps. The number of steps per beat influences the interval of the trap because each beat is subdivided by the number of steps.

The Metro library's trap interval is set in milliseconds thus we must calculate the interval between steps by converting the configured BPM to BPMs. Then the BPMs must be inverted to attain the Ms/B and lastly divided by the active steps per beat to find the millisecond interval between steps.

$$BPM\ to\ BPMs = \frac{Beats}{Minutes} \cdot \frac{Minutes}{Seconds} \cdot \frac{Seconds}{Milliseconds} = \frac{Beats}{Milliseconds}$$

$$BPMs\ to\ Ms/B = \frac{\left(\frac{Beats}{Milliseconds}\right)^{-1}}{Steps_{Active}} = \frac{60000\ Ms}{BPM \cdot Steps_{Active}} = Metro\ Interval$$

We leveraged the Teensy 4.1's 600MHz ARM Cortex-M7 clock speed to minimize the delay between checking if the Metro timer's trap has triggered. Once the interval set for the Metro timer has elapsed, the per step subroutine is executed to stop previous step MIDI sounds and turn off the previous step's LED. Next, the following step's LED and sounds are activated and the subroutine exits, returning to the main loop to scan button inputs and wait for the next Metro trap.

J. Power Source and Battery

We met system portability and functionality constraints by utilizing the energy density of lithium-ion battery chemistry, accompanied by a fully featured battery charger for seamless passthrough, disconnect, and shut off functionality. With the goal of at least eight hours of active system operation while powered by the onboard battery we conducted a full system power analysis to direct the selection of battery capacity. The conservative estimate from the power analysis was four watts; therefore, to achieve at least eight hours of active operation, we needed a battery capacity of at least 32 Watts/hour.

We selected a 1S3P 18650 battery pack with a 3.7V nominal voltage at 10,050 mAh. This battery pack has a 37-watt/hour capacity, allowing for a predicted 9.25 hours of system operation, with a more realistic up time of 11.50 hours at 3.2W operation. Lithium-ion battery chemistry provides the greatest energy density in our price range but also carries significant safety hazards if used improperly. The battery pack we chose contains an onboard battery management system implementing overcharge protection at 4.25V, under-discharge protection at 2.5V, and current protection for 4-6A. Although the battery management system implements some safety features, it alone is not suitable to attach directly to our system.

To integrate the battery pack into the system, we selected the Adafruit PowerBoost 1000c Charger to act as an intermediary battery management system and battery charger. This additional power circuity completes the functionality of the power delivery system. The PowerBoost Charger enables six crucial features: load sharing, buck-boost, power switch, charging port, indicator LEDs, and high operational efficiency.

The onboard MCP73871 Lithium-Ion battery charge management controller adds load-sharing functionality allowing for seamless operation while connecting and disconnecting to an external power source. The onboard TPS61090 DC to DC buck-boosting converter bumps the 3.7V nominal battery voltage to a stable 5.2V used across the system powering the Teesy 4.1, LCD, and analog circuitry. Onboard the charger is an enable pin that is used to control the

charger output, implementing an on / off switch. An onboard micro-USB port allows 5V 1A charging to the battery pack or as a passthrough to the system if the battery is already fully charged. This passthrough increases the battery's longevity by supplying the charger output from the input without needlessly cycling the battery. Indicator LEDs inform the user of the charger's power state. Blue indicates a 5V output power state. Red indicates lower battery voltage below 3.2V. Yellow indicates the battery is being charged. Green indicates the battery is done charging. Lastly, the Powerboost Charger implements all of these features with a 90% operation efficiency.

K. Track Saving & Loading

We achieved track saving and loading functionality by writing files to the onboard SD card in the format of the nested data structures used for measure chaining. ArduinoJSON was utilized to perform buffered I/O on json files in the root directory on the SD card labeled "tracks". The track files are named in the format TRACK-<XXX>.json based on a three character selection by the user in the graphical user interface. On track save, all structs corresponding to nested measure, nested beats, nested steps, nested sounds, steps per beat, beats per measure, and palette state are stored in the nested JSON list and object format. When a saved track is selected in the graphical user interface to be loaded, the current track active has its memory freed except for shared cached sounds in the new loaded track. Then the new loaded track's custom sounds are cached and palette state and all chained measures are populated with the corresponding data structure information from the json file.

JSON as an object data structure format was chosen given its wide adoption in web applications. As an open source project, basing the track storage data structure on JSON opens the door for future feature expansion from the open source community.

L. Effects

This menu contains effects that modify how ModuLoop loops through each measure. They are used by selecting them in the menu, assigning them to the palette, and then holding down the assigned palette button to use them. Once the effect button is released, the measure goes back to looping how it was previously. Here is a list of effects the user can choose from:

a. **Repeat return**
   While holding, the device repeats the step the effect was held on, once released the measure starts back at the beginning of the measure.

b. **Repeat continue**
   While holding, the device repeats the step the effect was held on, when released the measure continues playing from where it left off.

c. **Reverse**

While holding, the measure starts playing backwards. This means the steps play from right to left, bottom to top. When released the measure returns to looping normally which is left to right, top to bottom.

d. **Echo**
While holding, the device repeats the step the effect was held on while the volume of the sounds decreases to silent. When released the measure continues where it left off.

e. **Mute**
While holding the measure LEDs still cycle normally but the sounds are all muted. When released the sounds continue to play again.

f. **Scratch**
While holding, the device repeats between the step the effect was held on and the step before it. Once released the measure continues playing from where it left off.

g. **Double step**
While holding, each step is played twice in the same interval as one step. Once released the measure goes back to playing normally.

III. THE PROTOTYPE

A. Prototype Overview

In terms of software, our system has specified all features we set out to complete. . Following our software block diagram, the entire user experience can be followed. By following the display navigation and software block diagram logic, the user can access all features of the system. Dynamic population of both custom and MIDI sounds are implemented into the display navigation, and interaction between the display and palette is functional. The complete four-row by nine-column button matrix is functional and implemented, allowing for navigation to palette sound assignment, palette to measure matrix step assignment, and palette effect toggling alongside complemented LED integration are all active. Four-sound polyphony is functional between any combination of MIDI and custom sounds per step whilst achieving a +/ 2% timing accuracy when played at the maximum tempo of 200 BPM. Effects can be assigned to any palette button and are functional and correctly integrated to the active measure while evoking corresponding LED's per step played. The prototype has the ability to chain measures together to create larger pieces of music, along with being able to delete measures from this measure chain. Within these measures, we have the ability to edit beats per step along with the number of steps per each beat. It also has the ability to save tracks, delete tracks, and load tracks. This means that we can save measure chains with custom palette configurations, sounds on each measure, and steps per beat and bets per step associated with each measure in the measure chain. When editing the sounds on the measure matrix, we can identify where sounds from the

palette are on the measure matrix by holding down that palette button. In response to this, only the LEDs associated with the step that specific sound is on will light up when cycling through the measure, giving the user more insight into what is being played. If you press a palette button, the display will tell you which sound or effect is on that button. When you press a measure matrix button outside of edit mode, the display will list all sounds on that step, or tell you the step has no sounds and is empty.

In terms of hardware, we have designed subsystems for our device that have been implemented together. Our button system uses a row-column address that scans through each button using decoders. The decoders allow us to connect more buttons using less GPIO pins on the Teensy 4.1. The LEDs use decoders as well as inverters to activate an individual LED with a row-column address. This Interfaces with the Teensy 4.1 to detect button presses for assigning sounds to our measure and the LEDs dynamically change based on the buttons pushed.

Our Analog Audio system includes the low pass filter for signal noise, a variable low-pass and high-pass filter, and two amplifiers. The first amplifier is before the headphone jack that is used as a way to adjust the volume of the signal. The second amplifier is after the headphone jack which only amplifiers the onboard speakers. All of the filters are an active design and the variable filters use a potentiometer to adjust how much of the audio signal is filtered. This system is fed sounds through the MIDI chip or the DAC which are mixed together and filtered before outputting after the amplifier. Our output is sent to a headphone jack which has a disconnect to send the audio signal to our speaker only when there are no headphones plugged in.

Our power management is done with a battery management system and a LiPo battery. The BMS will regulate and send power to our system from the battery and when the BMS is plugged into the wall it will charge the battery and pass through power to our system at the same time. We also have an LCD that communicates over I^2C with our Teensy 4.1. This is to allow for relevant information to be displayed to the user. Figure 9 shows the built out system for our FPR prototype.

We have created a custom enclosure for ModuLoop using a 3D printer and wood. The faceplate was designed using a Fusion360. For the button matrix and palette cutouts we referenced an adafruit design that created a 3D box with the button pads we used. We measured and designed cutouts for the rest of the components as well as adding the necessary screw hole. The faceplate was designed as two separate pieces that were glued together above the measure matrix which can be seen in figure 9 where the faceplate starts being angled.The faceplate was then painted with multiple layers of a black acrylic paint and the potentiometer labels were printed out of a screen printing vinyl. We then sprayed a clear coat over the whole faceplate to protect it and achieve the texture we wanted.

For the wooden part of the casing we got half inch poplar boards for the sides and quarter inch plywood for the bottom and sized them based on the faceplate we designed. We cut a quarter inch groove along the top of the case to allow the faceplate to sit flush in the case. We also created notches around where the headphone jack, micro-USB, and Micro SD card slots are to ensure they are accessible to the user. There are metal standoffs as seen in Figure 10 that are screwed to the bottom of the case to secure the audio PCB as well as the three Button PCBs. Once the sides of the case were glued together we put two coats of a wood stain on to give it a finished, professional look.

Figure 10: Prototype for FPR build

Figure 11: Casing with PCB standoffs

B. List of Hardware and Software

List of Software:

- C++ Standard Template Library
- Metro Timing Library
- Teensyduino IDE plugin for Teensy microcontroller
- Arduino LiquidCrystal library modified for Teensy
- Arduino SD library modified for Teensy
- TeensyVariablePlayback by Nic Newdigate
- TeensyAudioFlashLoader by Nic Newdigate
- LiquidCrystal I2C by Frank de Brabander
- StreamUtils by Benoit Blanchon

Audio PCB Hardware:

- Audio PCB x 1
- Teensy 4.1 x 1
- PSRAM 8MB x 1
- Adafruit Music Maker Featherwing x 1
- Adafruit Powerboost 1000c BMS x 1
- PT8211 DAC x 1
- 10k ohm Potentiometer x 4
- LM386m-1 x 1
- NE5532 x 7
- TPA2016D2 x 1
- 74HC138 Decoder x 3
- 74HC104 Inverter x 2
- BA33JC5 3.3V Voltage Regulator x 1
- 0805 SMD Resistors x 40
- 0805 Capacitors x 32

- Through-Hole Capacitors x 31
- 3.5mm Headphone Connector with Disconnect Pin
- Assorted Header Connectors

Button PCB Hardware:

- Button PCB x 3
- 3mm Red Through-Hole LED x 36
- 1N5818 Diode x 36
- 1206 SMD Resistors x 18
- Assorted Header Connectors
- Adafruit Silicone Button Pads x 3

Other Hardware:

- Adafruit 10050mAh 1S3P Lithium-Ion Battery Pack x 1
- 20x4 LCD with I2C Controller x 1
- 12mm Push Buttons x 4
- Non-Momentary Power Switch x 1
- 8Ω 5W Speakers x 2

C. Custom Hardware

For the PCB design, we referenced the design from a 2023 senior design project as well as the PCB files given for the button pads we chose. The button we chose uses silicone button pads with a custom contact. The PCB had to include this custom contact pad as long as resistors and diodes. We used the same row column layout as the reference designs, but we are using smaller buttons so there was less space to route everything. We have connected three button PCBs that are a 3x4 matrix to create one big 9x4 matrix. Figure 12 shows the three PCBs which are connected through their rows to create the 9x4 matrix. The back of one 3x4 button PCB is shown in Figure 13. You can see traces connecting the rows horizontally as well as the diode footprints to protect from false button pushes.

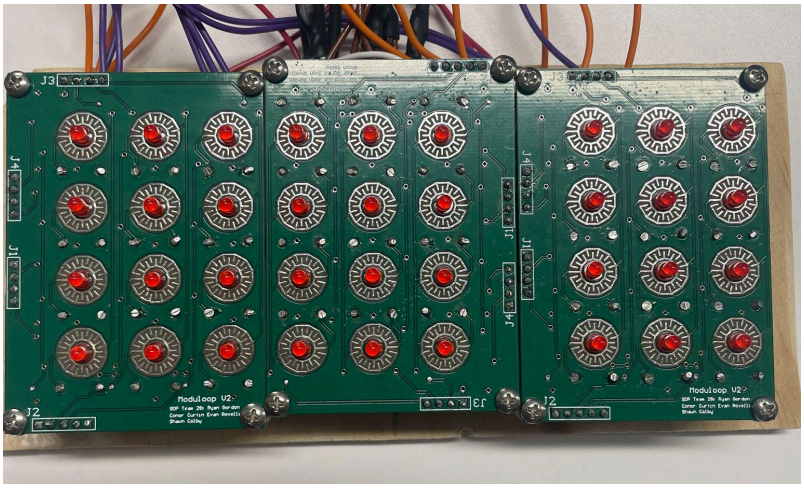

Figure 12: Full 9x4 Button matrix with LEDs

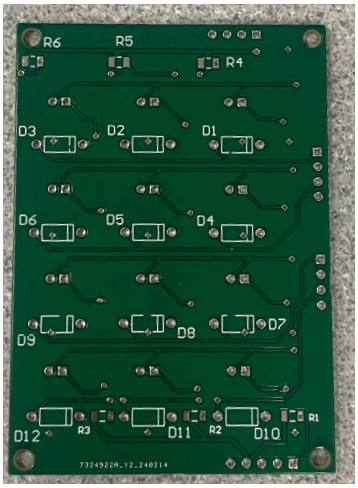

Figure 13: Back of one 3x4 Button matrix

We have also designed a second PCB to hold the rest of our components. This includes three decoders and two inverters to control our buttons and LEDs. We also have our audio sources. There is our MIDI chip as well as a DAC to output our sounds as an analog signal. Our low pass filters for noise, variable low-pass/high-pass filter, and two amplifiers implemented on our board. One amplifier is for the headphone jack and volume control and the second is just to increase the output volume from our onboard speakers Our Teensy 4.1 and our battery management system are also in the audio board design. When designing this board we made sure to physically separate the analog and digital components. This is a way to reduce the noise of digital signals with the analog signal. Figure 14 shows the audio board design with a line dividing the analog and digital components. Along with physical separation, we implemented decoupling capacitors and proper grounding techniques to reduce as much noise as possible.

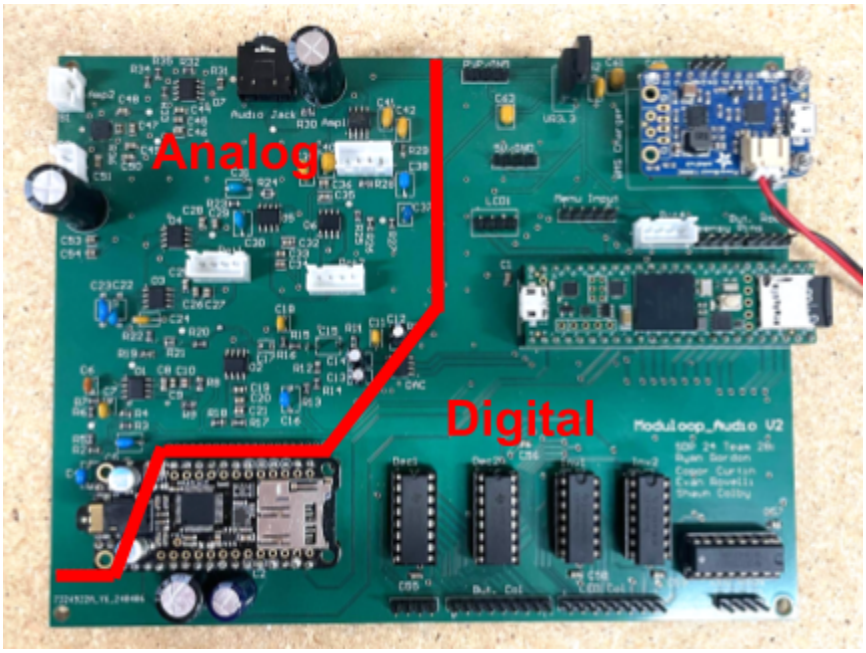

Figure 14: Audio board showing the separation of digital and analog components

D. Prototype Functionality

Our system has all the functionality we intended to have for the final FPR build. It can play a continuous looping measure. On each of the steps of this measure, it can play sounds from both SD sounds and MIDI sounds, with a max polyphony of four. Each step being played is identified by an LED that denotes the position currently active in the matrix. The LCD with input from the directional pad allows the user to navigate through the available custom sounds stored on the SD expansion card and the instrumentation library on the MIDI chip. The menu can also set the number of steps in each beat which can be applied to all the beats in a measure or just a specific beat. The menu also allows the user to select how many beats are in each measure. The default is using all 6 steps and all 4 beats, but the user can select 1-6 steps to play for every beat and 1-4 beats for each measure. We also have a tempo potentiometer that can dynamically update the tempo which is displayed at the bottom of the LCD.

We designed a variable low pass and high pass filter that uses potentiometers to adjust how much of the filtered sound is being output. Both of the filters can adjust how much of the low pass or high pass sound is sent to the output. These filters are balancing the filtered and unfiltered signals when turning the potentiometers in order to make sure there isn't any volume loss when turning up the filters.

We can also assign sounds from our menu into our palette and then on to our measure matrix. The sound is selected with the direction pad on the LCD which prompts the user to select a palette button. Once a palette button is selected the sound is assigned to the specific button. Pressing that button again will display what sound was selected on the LCD and then

measure matrix buttons can be selected. Once a measure matrix button is selected the sound is placed into the measure and it will start playing in the loop. You can remove a sound from the measure by selecting the palette button the sound is assigned to and then selecting a measure button with the sound already assigned to it.

Another feature we have is effects. They are assigned to the palette the same way as sounds. The effect menu is selected in the LCD which then allows the user to assign it to a palette button. Effects are activated when holding down the button. We have implemented 7 total effects into our system. The names and descriptions of all the effects are detailed in block L of the Design Section. We also added more deletion options. There is a new menu within the LCD which allows the user to delete the last sound added, all the sounds on a selected step, or all the sounds in a full measure.

ModuLoop has the ability to save recall tracks. A track includes all the measures that have been chained together as well as the current configuration of the palette. The user can save a track with a three character name that can be recalled. The user can also recall the palette of a saved track into the current track being edited. ModuLoop has the ability to chain multiple measures together in the same track and the user can recall a specific measure in a saved track to add onto the current track being edited.

Our system has a battery management system that allows ModuLoop to run off of a Lithium Ion battery or micro-USb power. Our system uses 0.44A at 3.7V which is a 1.6W power draw. With our 37 W/hr battery we are able to estimate a 23 hour runtime for our system on battery power. We have also implemented other features to improve the user experience that we didn't initially define as specifications. The user is able to hold down a palette button with a sound assigned to it and the LEDs in the measure matrix will only light up that specific sound on that step. This is a way for the user to easily see where specific sounds are within a measure. We also added a play/pause option within the LCD menu. This allows the user to stop the measure from cycling to apply sounds easier or just take a break.

E. Prototype Performance

Our prototype achieves all of our design specifications, and more. The functionality of ModuLoop includes:

- Playing SD sounds uploaded to the device
- Playing both SD sounds and MIDI sounds on the same step
- Navigate through SD sounds in the user display
- Play audio through 3.5mm audio connector to external headphones or speakers
- Play audio through built-in speakers
- Having accurate metronome timing while the full system is running and users are interacting with the device
- Mix 4 sounds on a step
- Have fully-functioning HPF and LPF that can be applied to the output
- Navigate MIDI sounds in the user display
- Have a user-adjustable tempo
- Power the system through built-in batteries or external wall power
- Save/load user-created tracks

- Measure chaining
- Measure reordering
- Variable beat/step selection
- Update sound deletion logic
- Sound pause functionality
- System is under 13in x 13in
- System is contained in one enclosure

The overall performance of ModuLoop is also quite positive. Details of the testing can be found in Appendix C. The quantitative results are outlined below:

- Total power consumption: 1.6W
- Estimated battery life: 23h
- Timing accuracy between steps is accurate to 2% of the projected timing

IV. CONCLUSION

This project has been a very valuable learning experience for all members of this team. Our final prototype exceeded the standards we set, and we were able to achieve substantially more functionality than we had initially anticipated. By combining each member's expertise and research, we were able to achieve on a very high level, and all members of our team are beyond proud of what we have created. ModuLoop is able to take a user's musical ideas and turn them into a reality. The modularity of features provided by ModuLoop creates an intuitive platform where a musician can create and develop their own ideas on a fully functional device.

ACKNOWLEDGMENT

We would like to thank Professor Anderson and Professor Lorenzo for their influential feedback at various stages of our design. We would like to thank the course TA Qingchuan Wu and Jonah Yolles-Murphy for their assistance in designing and troubleshooting our system. We also would like to thank SDP23 Team 11 for their support early on in our first PCB revision. Last, but certainly not least, we would like to thank our advisor Professor Baird Soules for his amazing guidance and support throughout this past year.

REFERENCES

[1] Jenkins, Jake. "The History of Sequencers." *inSync*, 18 Apr. 2022, www.sweetwater.com/insync/the-history-of-sequencers/.

[2] P. Horowitz and W. Hill, The art of electronics, 2nd ed. Cambridge [England] ; New York: Cambridge University Press, 1989.

[3] "Audio Component Grounding and Interconnection | diyAudio." Accessed: Apr. 03, 2024. [Online]. Available: https://www.diyaudio.com/community/threads/audio-component-grounding-andinterconnection.163575/

[4] D. Self, Audio power amplifier design handbook, 4th ed. Oxford Boston: Newnes, 2006.

[5] “ESP - Op Amp Bypassing.” Accessed: Apr. 03, 2024. [Online]. Available: https://sound-au.com/project00.htm
[6] “Filter Design Tool | Filter Wizard | Analog Devices.” Accessed: Apr. 03, 2024. [Online]. Available:https://tools.analog.com/en/filterwizard/
[7] G. M. Ballou, Ed., Handbook for sound engineers, 4. ed. Burlington, Mass.: Elsevier, Focal Press, 2008.
[8] E. Mullins, I. Williams, G. L. Voi, and R. Puzio, “How to Properly Configure Unused Operational Amplifiers”. Texas Instruments. Accessed: Apr. 03, 2024. [Online]. Available:https://www.ti.com/lit/ab/sboa204a/sboa204a.pdf?ts=1712351983256&ref_url=https%253A%252F%252Fwww.google.com%252F
[9] “Op Amp Applications Handbook, 2005 | Analog Devices.” Accessed: Apr. 03, 2024. [Online]. Available: https://www.analog.com/en/resources/technical-books/op-amp-applications-handbook.html
[10] D. Self, Small signal audio design. Oxford Burlington, MA: Focal Press, 2010.
[11] R. A. Pease, Troubleshooting analog circuits: with electronics workbench circuits, Paperback reprint. inThe EDN series for design engineers. Boston, Mass.: Newnes, 1993.
[12] Texas Instruments, “Dual Low-Noise Operational Amplifier”, NE5532 datasheet, November, 1979 [Revised Jan. 2015].
[13] Texas Instruments, “Low Voltage Audio Power Amplifier”, LM386 datasheet, May, 2004 [Revised Aug. 2023].
[14] Texas Instruments, “3-Line To 8-Line Decoders/Demultiplexers”, SN74HC138 datasheet, December, 1982 [Revised Oct. 2021].
[15] Texas Instruments, “Hex Inverters”, SN74HC04 datasheet, December, 1982 [Revised Apr. 2021].
[16] Texas Instruments, “2.8W/Ch Stereo Class-D Audio Amplifier With Dynamic Range Compression and Automatic Gain Control”, TPA2016D2 datasheet, June, 2008 [Revised May. 2016].
[17] A. Industries, “Adafruit Music Maker FeatherWing - MP3 OGG WAV MIDI Synth Player.” Accessed:Apr. 03, 2024. [Online]. Available: https://www.adafruit.com/product/3357
[18] VLSI Solution, “Ogg Vorbis/MP3/AAC/WMA/FLAC/MIDI AUDIO CODEC CIRCUIT”, VS1053b datasheet, August, 2014 [Revised Feb. 2024].
[19] Princeton Technology Corporation, “16-Bit Digital to Analog Converter”, PT8211 datasheet, February, 2012.
[20] A. Industries, “Lithium Ion Battery - 3.7V 10050mAh (10 Ah).” Accessed: Apr. 03, 2024. [Online].Available: https://www.adafruit.com/product/5035
[21] ROHM Semiconductor, “1.5A Variable/Fixed Output LDO Regulators”, BA33JC5 datasheet, February, 2012 [Revised Jun. 2012].
[22] “MVC - MDN Web Docs Glossary: Definitions of Web-related terms | MDN.” Accessed: Apr. 06, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC
[23] Texas Instruments, “Low-Noise FET-Input Operational Amplifiers”, TL072 datasheet, September, 1978 [Revised Apr. 2023].
[24] A. Industries, “Stereo 2.8W Class D Audio Amplifier - I2C Control AGC - TPA2016.” Accessed: May 14, 2024. [Online]. Available: https://www.adafruit.com/product/1712

[25] “Project 160.” Accessed: May 15, 2024. [Online]. Available: https://sound-au.com/project160.htm
[26] Texas Instruments, “2.5W Audio Power Amplifier”, LM380 datasheet, December, 1994 [Revised Apr. 2013].
[27] Texas Instruments, “5W Audio Power Amplifier”, LM384 datasheet, February, 1995 [Revised Apr. 2013].
[28] Texas Instruments, “1.4W MONO Filter-Free Class-D Audio Power Amplifier”, TPA2005D1 datasheet, July 2002 [Revised Oct. 2015].
[29] “SparkFun Mono Audio Amp Breakout - TPA2005D1 - BOB-11044 - SparkFun Electronics.” Accessed:May 15, 2024. [Online]. Available: https://www.sparkfun.com/products/11044
[30] Texas Instruments, “Low-Noise FET-Input Operational Amplifiers”, TL072 datasheet, September, 1978 [Revised Apr. 2023]

APPENDIX

A. Design Alternatives

Over the course of this project, we made a few key design decisions that shaped our final project. The first was our decision to use C/C++ and the Arduino IDE. Our other option was using Rust. After selecting our microcontroller, we discovered the plethora of supporting libraries for it that are written in C++. If we were to use Rust, we would have to port all of these libraries so that we could use them. Noting the amount of time that would take, as well as the time needed for our team to learn Rust and the limited resources available to us for Rust, we decided on C/C++.

Another key software decision was to use a pseudo-interrupt system based on the hardware clock built into the Teensy 4.1. While timing was a major priority in our system’s design, we decided not to do a traditional hardware or software interrupt because our use case requiring a hard deadline could alternatively be handled without being built upon a real time operating system. Through careful organization of our mainloop and leveraging the 600MHz ARM Cortex-M7 processor on the Teensy 4.1 we minimized the instruction ratio between idle computing between measure steps and our Metro library based timer trap allowing for near imperceptible looping accuracy.

We chose to interface the buttons and LEDs on our system using a row-column addressing system. This was done primarily to limit the number of GPIO pins we would need access to to control this part of our system. With this came the decision to use three separate button PCBs instead of one large PCB. In doing this, we were able to save some cost because the minimum order quantity from the vendors we used (OshPark & JLC) was 3-5 boards. This made mounting more complicated, as there were more points that needed to be attached to support the three PCBs.

Our final analog filter design blends filtered and unfiltered signal paths over a potentiometer. This allows a user to adjust the mix of the signals from a know on the front of the device. Our original intent was to have adjustable cutoff frequencies for the filters. This is easily attainable for a low-pass filter by swapping out the resistors for a potentiometer, but not for a high-pass filter. Through testing and research, we were unable to figure out a method that would allow both for an adjustable cutoff frequency and for the signal to pass through the filter. We

then settled on the design used for our final build, and configured the low-pass filter the same way for uniformity.

We made the decision to mix our signals in analog. Our original idea was to do digital mixing and have some DSP present, but after selecting the VS1053b, we decided to go analog. The VS1053b outputs directly to analog, so to do digital mixing we would have to convert back to digital via an ADC, then convert back to analog through a DAC. Mixing in analog simplified some of the circuitry needed for this conversion, but also added more complexity in the circuit and PCB designs. The signals from our two audio sources are mixed over 100Ω resistors. This is the simplest way to mix analog signals, and we chose to do this because it again reduced the complexity of our analog audio circuit.

Our final casing design utilized a wooden structure with a 3-D printed faceplate. Our original idea was to make the casing out of metal to give ModuLoop a robust structure. Using a metal case also would have allowed us to ground the whole system to the case. However, our team lacks the experience with metalworking that would have been required had we gone down this route. As well, doing a sheet metal case would have substantially increased the weight of our system. Part of our faceplate sits at a 22.5° angle. This was decided upon because of the 90° viewing angle. This was enough that a user can clearly read the display either while sitting down or standing up over the device. In laying out our system, we decided to offset the audio board in the case and set it 0.25in into the side of the case to give a user access to the ports on the board. We could have used pass-through ports, however we needed access to the micro-SD card slot on the Teensy 4.1, so we would have needed to do this anyway. We also chose to use separate PCBs to handle our buttons and Audio components. Originally, we were planning to put everything on one PCB, but we split them up to give us more flexibility in mounting; that which we needed when it came to the case design.

B. Technical Standards

We are using various serial connections to interface with the hardware we selected. The VS1053b is interfaced using SPI through the software serial library for Arduino. The PT8211 DAC is interfaced through I^2S to transmit audio data. The PSRAM IC mounted on the bottom of our microcontroller board is interfaced through QSPI (quad-SPI).

These communication protocols are industry-standard practices and, while they don't have specific IEEE standard numbers, they conform to IEEE's overarching guidelines for electronic communication. Specifically, they adhere to the principles of synchronous serial communication, ensuring seamless compatibility across a variety of devices.

C. Testing Methods

To measure the accuracy of the timing of our system, we tested our system at three tempo settings: 50 BPM, 100 BPM, and 200 BPM. At 6 steps per beat, we are expecting 200ms, 100ms, and 50ms delay between each note for 50, 100, and 200 BPM respectively. We then applied staccato notes (musical notes with a distinct attack and little resonance) to the first two steps in each beat of the measure. Using an oscilloscope, we were able to visualize these notes, and used the cursor feature to measure the delay between the two notes, and compared them to the calculated thresholds. Figures 15-17 in the timing section show the plot outputs.

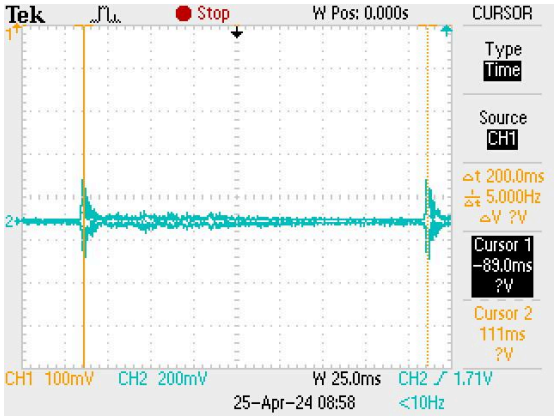

Figure 15: Timing Measurement at 50 BPM

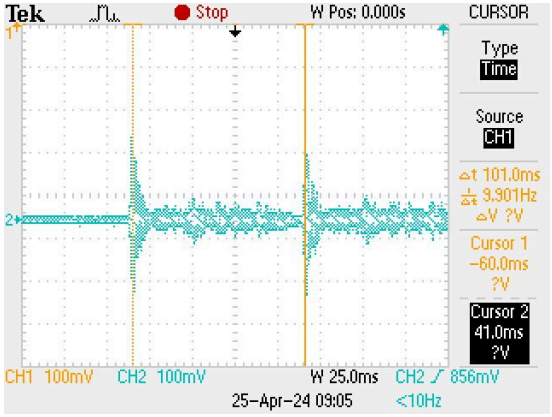

Figure 16: Timing Measurement at 100 BPM

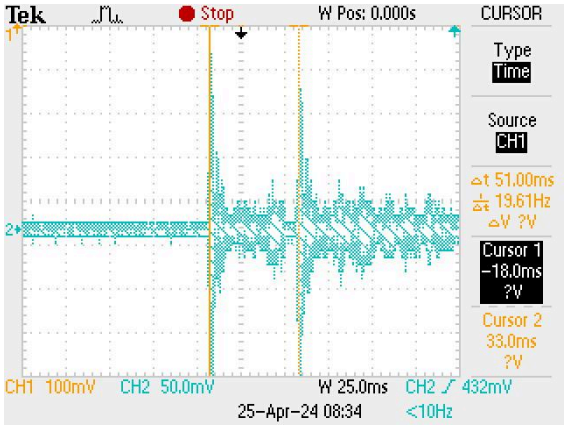

Figure 17: Timing Measurement at 200 BPM

To measure the analog filters, we used the FFT feature on an oscilloscope. We took a baseline measurement showing the full spectrum of audio being tested, which is shown in Figure 18. We then measured with the low and high-pass filters active, which are shown in Figures 19 and 20, verifying their functionality.

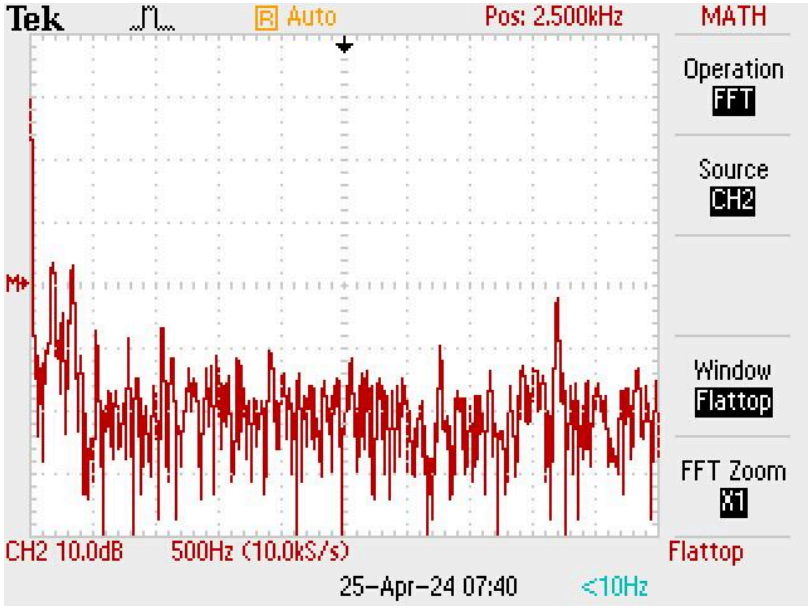

Figure 18: Unfiltered FFT Spectrum

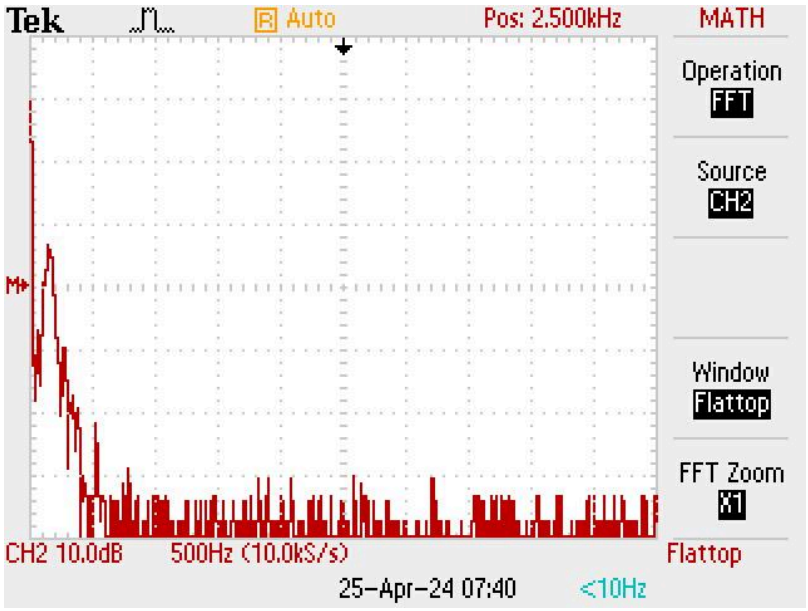

Figure 19: f_c = 300 Hz Low-Pass Filter Applied

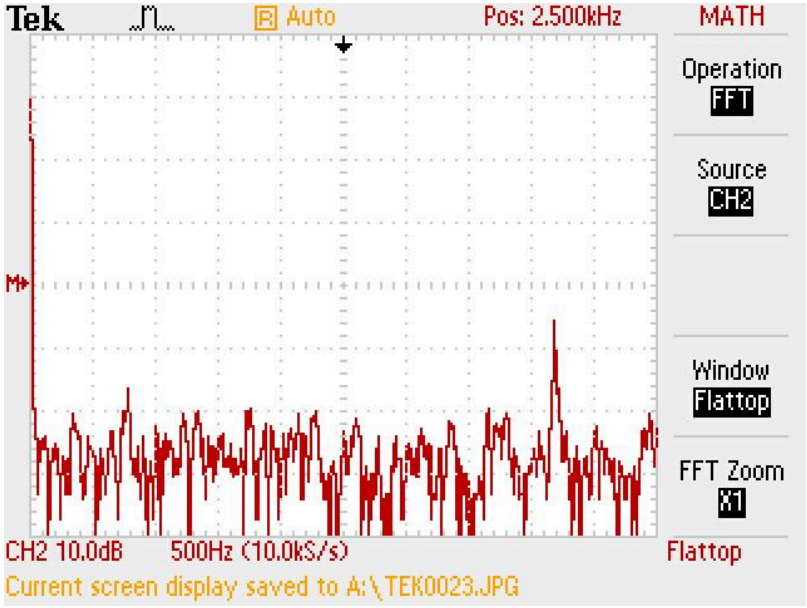

Figure 20: f_c = 3kHz High-Pass Filter Applied

D. Project Expenditures

The overall BoM of ModuLoop came in at $224.10. Table 2 shows the full component list of our system, not including the casing materials.

Component	Quantity	Price	Component	Quantity	Price	Component	Quantity	Price
Teensy 4.1	1	$31.50	3mm Red LED	36	$6.20	SN74HC04N	2	$1.40
PSRAM 8MB	1	$1.80	Diode	36	$9.58	BA33JC5	1	$2.57
PT8211	1	$2.65	Silicone Button Pad	3	$23.16	PCB	1	$7.96
LM386m-1	1	$1.49	1206 Resistor	9	$0.90	20x4 LCD Display	1	$13.30
TPA2016D2	1	$2.23	Powerboost 1000c	1	$19.95	Speakers	2	$10.99
NE5532	7	$3.78	0805 Resistors	40	$0.68	Potentiometer Caps	4	$8.79
10k Potentiometer	4	$6.08	0805 Capacitors	32	$2.40	Power Switch	1	$0.67
Music Maker Featherwing	1	$19.95	Through-Hole Capacitors	31	$4.37	Li-Ion Battery	1	$29.95
Button PCB	3	$6.54	SN74HC138N	3	$3.72	12mm Grey Tactile Button	4	$1.49
						Total Cost		$224.10

Table 2: Final System Expenditure

Our total expenditure for this project came out to $477.77. Table 3 shows the full list of components we ordered.

Component	Quantity	Price	Component	Quantity	Price	Component	Quantity	Price
LM386m-1	4	$5.96	PT8211 DAC	4	$10.60	0.1uf Through-Hole Ceramic	20	$3.12
Music Maker Featherwing	1	$24.95	PSRAM 8MB	4	$7.20	1uf Through-Hole Ceramic	3	$1.44
Silicone Button Pads	3	$14.85	10k Potentiometer	10	$15.20	10uf Through-Hole Ceramic	10	$5.61
20x4 LCD	1	$13.30	Single Row Connector Pack	1	$1.95	470uf Through-Hole Aluminum	3	$1.59
3mm Red LED	50	$8.60	8ohm 2W Speaker	2	$5.20	Audio PCB V1	5	$39.80
1N5818 Diode	50	$13.30	TPA2016D2	5	$11.15	Audio Stencil V1	1	$7.00
Powerboost 1000c	1	$19.95	0.1uf 0805 Ceramic	50	$2.09	Audio PCB V2	5	$40.00
Li-Ion Battery	1	$29.95	1uf 0805 Ceramic	50	$2.29	Audio Stencil V2	1	$7.00
Button PCB V1	3	$30.70	10uf 0805 Ceramic	20	$1.40	Tax, Shipping, Tarrif	N/A	$148.97
Button PCB V2	5	$3.10	1nf Through-Hole Ceramic	10	$1.50	**Total**		$477.77
						Remaining Budget		$22.23

Table 3: Total Project Expenditure

E. Project Management

Our team divided our work into sections based on our individual strengths, Ryan was the PCB lead, Conor was the Logistics and budget Lead, Evan was the software lead, and Shawn was the timing lead. We each focused on these sections and worked together to integrate each portion into one cohesive design.

Ryan led the design for all the PCB designs for our system. He designed the Button PCBs and created two iterations to refine the design. He also wrote the code for the button and LED logic. He then worked with Shawn to integrate that logic with the overall timing of the system. Ryan also designed the audio board and completed two iterations of that PCB as well. He worked with Conor to help design the analog low-pass and high-pass filters and then designed the PCB with the system we tested with on breadboards. Finally, Ryan worked to design and assemble the case. He modeled the faceplate with cutouts for all the components and 3D printed two iterations to solidify the design. Ryan also did the painting and finishing of the faceplate as well as working with Conor to mount and assemble all of the components into our case.

Shawn managed the timing of the software logic to accurately match the tempo and desired BPM of the system to ensure a seamless music experience for the user. He did this by extensively researching the topic and prototyping different approaches early on in the design process. Once landing on the solution for this using the Metro library, it was still critical to ensure

that the software does not get “stuck” somewhere so the system always reached the hard deadline for playing a sound within the desired timing constrictions. After getting the backbone of the timing logic in place, Shawn wrote the software for how both MIDI sounds and SD sounds were stored and called back during their desired play times within the measure. After figuring out how to store and call back sounds within the system, Shawn wrote the logic to move these sounds from the palette to the measure matrix. In addition to this, Shawn wrote the software for all the effects that can be imposed upon a measure when holding down a palette button.

Conor was the manager for the team. He was responsible for coordinating team meetings, advisor meetings, and the four evaluations. He also handled the budget for the team, and did the majority of the component procurement. Conor was primarily responsible for the analog audio circuitry design. He spent vast amounts of time researching and building test filter circuits to work towards the filter functionality desired. Once the filters were built and tested audibly, Conor handled testing and validation of the filters via an oscilloscope FFT. He also handled the amplifier R&D, and did the testing necessary to make the amplifiers work together in series. Conor did the early development for the VS1053b needed to work with the IC in its MIDI mode. He researched and validated the functionality of the codebase utilized. Conor was responsible for most of the soldering and SMT stencil work required for ModuLoop’s PCBs. After the PCBs were populated, he worked alongside the rest of the team to validate and troubleshoot the PCBs. Conor also built the wooden portion of the case, and worked alongside Ryan to fit the faceplate into the casing and do the final layout of the components.

Evan was responsible for the software architecture and hardware integration. He achieved this by modularizing the program structure for simple linking of hardware modules and supporting data structures. He worked with Conor to integrate MIDI sounds, Ryan to integrate matrix button and LED control, and Shawn to integrate looping, effects, and palette logic. Evan designed data structures to implement model view controller based graphical user interface allowing the navigation of sounds, effects, tracks, and other measure related settings. He integrated persistent track storage and accessing alongside custom SD sound caching, and playback. Additionally integrated LCD, SD, and DAC hardware interfacing. Coalesced the version control system branches on github to unify program state across the team for collective development. Held open dialogue with team members during critical design decisions to address concerns and elevate ideas for example during major system restructuring and code cleanup from CDR to FPR.

F. Beyond the Classroom

Ryan - Before this project I had never designed a PCB before. I used my introductory knowledge of Altium and used a lot of their documentation to learn how to properly design a PCB. We had two completely different board designs as well as a lot of analog components in our design. I learned a lot about audio circuit design as well as mixed signal board design since we had analog and digital components on the same PCB. I learned a lot about noise decoupling and the importance of grounding within a mixed signal PCB. I also improved my soldering skill significantly. We had over 100 components on our audio board alone along with our button PCBs and other components that needed to be soldered. This gave me the opportunity to get experience soldering various kinds of components as well as using an SMT stencil for our audio PCB. I learned the importance of revising designs during this project. I made multiple designs for both PCBs and the faceplate which was modeled in Fusion 360. Being able to design each

of these components multiple times allowed us to greatly improve and refine our designs for each revision to achieve the desired product. I also researched different filter designs for our adjustable low-pass and high-pass filters. It took a lot of time and design iterations to create filters that had the desired response. We used a lot of different components to test with and being able to use components from the UMass M5 Makerspace was crucial in being able to test a large amount of designs quickly. I also learned how important project management is. We created Gantt charts to manage our time and kept track of how much of the budget we were spending. Our team also used Github and Google Drive to share our work and keep the whole team up to date. As we designed ModuLoop, I wrote a lot of lists of things to do, problems we had, or questions that came up. This was very helpful for visually seeing everything we needed to get done and not forgetting any features or problems that needed to be addressed. This project has been useful for my life as a professional. Audio and instrument design is a field I am interested in pursuing and being able to this project designing a music sequencer has helped me build the skills that would be used in an audio/instrument design engineering field.

Shawn - In the past, I have worked on numerous projects that have taken weeks or even months to complete, but I have never tackled a project of this magnitude. A component that surprised me the most about this process was the extensive planning that needs to be done before the project can be started. PDR taught me how to properly plan out what needs to be done in a long term project like this one. Once PDR was over, it was then critical for us to split the project into pieces that each of us specialize in. For my part, I needed to learn a lot more about music in general and how timing in software generally works, which is something I have never explored before. I needed to expand upon my knowledge of hardware which was minimal before this project since I have not elected to take many advanced hardware classes. After finishing individual parts of the project, bringing things back together posed to be sometimes difficult and changes needed to be made to everyones' parts. This taught me to write my code in a way that can be easily integrated into other people's work. It also taught me to write more comments in my code to make it more easily readable for others so they can build off of it as well. I also had to familiarize myself with C++ which wasn't a language I have ever really worked with in the past. I have worked with C in some school projects and a few times in my internships, but I wouldn't consider low level languages to be my strong suit. This project was good practice for using these types of languages instead of the typical high level languages like python that I usually use. For our organization of software, we utilized Github which is a very commonly used application. This was good practice for me and helped me better understand how to use it. When I got stuck on a certain problem in this process, I had multiple resources to turn to. The first resource was either our advisor or TAs available for the course. In many other instances, I also found the TennsyDuino forum to be extremely helpful for many of the issues I was having with my software. Overall, I believe this experience was very helpful for my career in the future. I have already experienced many of the same issues and solutions between my internships and this project. Working within a team to accomplish a bigger goal is something I will be most likely doing for the rest of my professional career. Having this project as practice right before I start in the workforce is a great way to prepare to make the transition from the academic to the professional setting.

Conor - Much of what I was responsible for was something that I had never done before. The analog circuitry was way outside my wheelhouse, requiring copious amounts of research to determine where to start. Even simple things like determining the difference between resistor and capacitor materials, and the difference between operational amplifiers, instrumentation amplifiers and power amplifiers (and the classes of power amplifiers) were things I had to learn. Through research, I discovered a slew of textbooks ([2], [4]-[5], [7], [9]-[11]) that contained oodles of resources to support much of what I needed to get going. As well, coming across tools like Analog Devices' Filter Design Tool ([6]) were invaluable in quickly evaluating the response of certain filters. Along the same lines, I spent quite a bit of time researching and evaluating different amplifier solutions to maximize the versatility of our signal outputs. Through sites like Sound.au [25] as well as the Texas Instruments datasheets for a few amplifiers detailed in Appendix H, I was able to decide on the optimal choice for our system. Our final circuit design contained over a hundred passive components, making our final audio PCB quite large. Due to space constraints, we utilized a large number of surface mount components specifically in the analog audio circuit. Thankfully, our advisor told us about SMT stencils to make the assembly much easier. Because I was not familiar with stenciling or what solder paste was, I did quite a bit of research on how solder pasting worked and how to get the paste to not squish out from under the stencil. After a few attempts, I was able to get an even application on the board without it smearing. I found it really cool to watch the paste flow and shrink to the pads on the PCB. I also did some research and testing into different hand soldering techniques like drag soldering, which I used to mount the TPA2016D2. There was also a learning curve in the software portion of our project. I was not directly involved in writing most of the code, but much of the hardware debugging that I needed to do required me to be familiar with the codebase and to learn C++ pointers and objects. I had previously only used C and that pointer structure, so learning and playing with the C++ stuff was a good learning experience. While building the case, I had to learn how to use a router to notch the sides to make the faceplate sit flush on the case. I'm not super experienced in woodworking, but I really enjoyed the process of building the case and planning how to lay out the casing. I previously knew what a router was, but I had never actually used one. Now I want to buy one. This project in general taught me how to work on a team. Mainly, I had to learn how to handle the logistics of our team, and how to properly schedule things like meetings. I've learned so much over the course of the past year, and all of it will be useful for my professional career.

Evan - In the past I have worked on various embedded system and C applications but have never worked on a project of such hardware interoperability and feature richness. The project as a whole from conception to completion has presented me with a great deal of learning opportunities stemming from a background of little music experience and interaction with new technologies and protocols. My past embedded systems experience is predominantly using the ATmega328p arduino 8 bit microcontroller which was greatly beneficial when tackling the Teensy 4.1 as the Teensyduino development environment is integrated directly into the Arduino IDE. Difficulties arose however when learning although Teensyduino supports all standard Arduino libraries, they are in fact modified and not exhaustively canonical in feature set and reliability with the original libraries. This became a hindrance when encountering a major roadblock preventing custom sound polyphony. The pjrc.com wiki and forum was instrumental

as a resource when finding compatible hardware or troubleshooting unexpected problems. The optimization of pin allocation for hardware interfacing was a meticulous, creative, and fun process to achieve desired functionality while staying without resource constraints. Additionally, while I have programmed embedded applications in the past, I have never done so in C++ making this first time experience both challenging and exciting. It's very intimidating taking on a large project without knowing the exact best practices for a particular language, but this this experience at its heart is the core of engineering – devising solutions without having all the answers readily available, embracing the challenge, and ultimately expanding your knowledge and skills in the process.I intend to carry the lessons and resilience gained from this experience with me far into the future, as I continue my journey as a software engineer.

G. Op-Amp Selection

Initially, we planned to use the Texas Instruments TL072CP [30]. This chip was a popular choice for audio projects early on in our research for this project, and was readily available to us through M5. We built some early mixer and filter circuits using this IC, but quickly noticed undesired noise output from it. We also noticed that the tone of the output was quite harsh, and was lacking depth in the sounds. The TL072 is a FET-input op-amp, meaning it has a mosfet transistor being used to modulate the current across its input. Through further research, it was discovered that this chip does not behave well at low supply voltages. Based on its datasheet, we were running it at half of its recommended supply voltage. While still within its rated range, lower supply voltages reduce the input signal's operating voltage range. When this voltage goes past a threshold, it can make the FET-input latchup, sending the output voltage close to the voltage of one of the supply rails, and cause clicking on the output.

After some more research, we settled on the Texas Instruments NE5532 [12] as our op-amp of choice. This IC was also readily available through M5. The NE5532 is a BJT-input (Bipolar Junction Transistor) device, is rated for lower noise than the TL072, and is also rated to run on a 5V supply with no issue. This chip is also rated for unity-gain operation, allowing us to use it as a buffer without having to alter its gain configuration. After testing this IC in our circuit, we were pleased with its performance and the significantly better tone of its output.

H. TPA2016D2 Selection

We spent quite some time figuring out a solution to have both built-in speakers and a headphone jack. We first explored the Texas Instruments LM384 and LM380. These were selected for further research because they are readily available to us through the UMass M5 Makerspace, and they are available in a DIP format, making them easy to test with. After reviewing their datasheets, [26]-[27] we decided against both of these chips firstly because they are not rated to run at the 5V supply (LM380 is rated at 10-22V and the LM384 at 12-26V) that we have access to from our BMS. As well, these chips can get quite hot when running and based on [25], require an external heat sink if the output is going to drive greater than 1W to prevent the IC from dying.

From our research, the only way to drive more than 1W output without needing more than a 5V supply is to use a Class D amplifier. Class D amplifiers are differential, meaning that the amplitude of the signal ground is adjusted inversely to that of the signal itself, functionally doubling the strength of the signal. We explored both the Texas Instruments TPA2005D1 [28] and TPA2016D2 [16]. These ICs were again selected because they are readily available through the UMass M5 Makerspace. Both of these chips are surface-mount only, but are

available on breakout boards; the TPA2005D1 through Sparkfun [29] and the TPA2016D2 through Adafruit [24]. In testing, both of these amplifiers are good options for our application because they emit small amounts of heat and consume small amounts of power while also being able to drive 8Ω speakers. The TPA2005D1 has a single channel, whereas the TPA2016D2 is two-channel. Our intent was to use two speakers, so having two channels was ideal for our application. As well, putting two speakers in parallel cuts their resistance in half. Both of these ICs are not capable of driving 4Ω at the 5.2V we are feeding to the chips, so if we were to do this a 3.3V supply would need to be routed specifically for this chip. Because of this, we selected the TPA2016D2. We are using both channels of this chip, allowing us to drive both speakers independently. This IC has an I^2C connection that is used for a few things, most notably to set the gain on the amp. It also has active gain control (AGC), which progressively increases the gain on the amp until the signal begins to clip. This allows the amp to 'learn' the envelope in which it can operate, maximizing the drive strength of the amplifier. It is worth noting that AGC can only be disabled through the I^2C connection. The default gain setting is also only adjustable through the I^2C bus. For our application, we decided not to connect the I^2C bus and run just with the AGC. This IC also has a shutdown pin, allowing a user to turn the amp off when it is not in use. While this would save power when the device is outputting through the headphone jack, we did not utilize this feature solely based on lack of time to test it. On our PCB, we are using the bare IC, not on the Adafruit breakout board due to space constraints and because the IC itself is a quarter of the price. We also noticed in testing that this IC has very high spikes in power consumption that can cause the supply rail to dip and make the signal clip. To combat this, we added a 2200µf capacitor across its power supply pins to account for this.