The Samplisizer

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Lucia Hoerr

Spring, 2023

Technical Project Team Members

Quinn Ferguson

Cooper Grace

Frederick Scotti

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry C. Powell, Department of Electrical and Computer Engineering

Richard D. Jacques, Department of Engineering and Society

Statement of Work:

Cooper Grace
Cooper was primarily responsible for system integration and elements of the software development and design. He wrote the code to interface with the microphone allowing for samples to be recorded. He also designed the finite state machine for the system and integrated the lights and button into the code. Additionally, he collaborated with Quinn on the MIDI interface and playback of processed sounds. Cooper was also responsible for everything relating to the setting up of our microcontroller, code repository, and shared development environment.

Lucia Hoerr
Lucia was primarily responsible for the pitch shifting and audio algorithm in software. She wrote the code to implement a pitch synchronous overlap and add algorithm in Python which allowed for the input audio to be shifted up and down and mapped to the MIDI keyboard. Lucia was also responsible for the design, 3D-printing process, and assembly of the case for the Samplisizer. Lucia also assisted Frederick in finding and ordering the parts for the hardware portions of the project.

Quinn Ferguson
Quinn primarily worked on the MIDI interface and optimizing MIDI response time. He wrote code that would immediately respond to MIDI inputs and interpret them to quickly playback the correct sounds. He also wrote a program that would detect the first jump amplitude in a recorded sound and trim out all the dead space before it. This prevented any pauses in playback of sounds that did not start immediately, overall increasing responsiveness. He also made some modifications to raspberry pi files to make the program run on boot, and then debugged the main code so that it would run smoothly in that environment. He also worked on the basic GPIO functionality before it was integrated into the main program.

Frederick Scotti
Frederick was primarily responsible for the PCB. He designed the block diagram of the header board as well as the layout for manufacturing. This board was used to record audio using an ADC and control the volume of the output signal with a potentiometer. It also communicated with the Raspberry Pi through GPIO. He also ordered all the necessary parts and created all of the assembly documentation to send to WWW Electronics, Inc. (WWW). Frederick's secondary responsibility was assisting with integrating the GPIO drivers for the switches and LEDs.

Table of Contents

Table of Figures

Abstract

The Samplisizer is a Musical Instrument Digital Interface (MIDI) controlled electronic musical instrument that records and plays back sampled audio at different tones [1]. Each tone corresponds to a certain MIDI signal allowing the user to play music using different pitches of any sound recorded. This instrument takes the form of a box that has a microphone for recording audio, a circuit board that controls the volume with a potentiometer as well as handles button inputs and LEDs, and a Raspberry Pi to process the audio in digital and control what sounds are being output. These separate parts form a system that can create a user-friendly and accessible piece of music production equipment.

Background

The invention of digital audio samplers in the mid-1970s drastically altered the music industry and helped shape it into what we know today. Most modern pop, hip-hop, and electronic artists rely heavily on samplers in their songs. In today's music world, a sampler is just as important as any other instrument. Audio samplers allow for the creation of music from any recorded sound, be it a user's voice or the clicking of computer keys, for people with any level of musical ability. Our group members' musical talent ranges significantly, yet we will all be able to play the finished sampler box to make fun, catchy tunes that everyone will enjoy.

A similar project was created by the company YellowNoiseAudio [2] and is posted online under open source and open hardware licenses. The SamplerBox [3], as they call their product, is similar to our idea in that the actual bulk of the project is contained in a central box that can be hooked up to a MIDI keyboard and speakers. Their sampler does not allow for input through a microphone, however, so you must upload audio files using a Secure Digital (SD) card to the Raspberry Pi [4]. Our audio sampler will take the technology further by including a microphone for recording user sounds and with amplification of the sample.

This project brings together much of the knowledge we have gained during the last three years of engineering including embedded systems, hardware design, and software development. From the Computer Engineering Fundamentals courses (ECE 2630, ECE 2660, ECE 3750), all of the group members have experience designing and assembling Printed-Circuit-Boards (PCBs) for various projects as well as signal processing skills. Frederick has more previous experience with hardware design and electrical engineering, so we relied more heavily on him for the hardware portions of the project. Lucia, Cooper, and Quinn have embedded systems experience from the Embedded Computing and Robotics I and II courses (ECE 3501 and ECE 3502) which centered around an MSP432 microcontroller [5] and writing code for it in the C programming language [6]. Frederick took Introduction to Embedded Computing Systems (ECE 3430) which also lent to this. Finally, both Quinn and Frederick possess musical abilities and knowledge of pitches and scales that were useful for understanding sampling and shifting the audio. These skills and

experiences together helped the group successfully navigate the hardware and software system requirements for the Samplisizer.

Physical Constraints

Design Constraints

Our design was relatively straightforward to manufacture as the PCB and casing were the only custom components involved in the design. Most of the design consisted of off-the-shelf components which minimized complexity. Additionally, the use of standardized interfaces for the MIDI input and auxiliary (aux) output allowed for customizability on the user's end while also reducing the complexity of constructing the device. The PCB manufacturing process, carried out by the contract manufacturing firm WWW Electronics [7], was a significant time constraint: one week to print the board plus several days to solder on the components. Our team had to go through this process twice because the first PCB had issues with the audio jack layout and the components were too small to be hand-soldered. The general PCB components were available in large quantities from multiple vendors and did not contribute as constraints other than longer-than-expected shipping times. The only parts that were not interchangeable were the analog to digital converter (ADC) and the Raspberry Pi. The ADC we used (TI PCM1808) was available in large supply from multiple sellers (Digikey, TI, Mouser) [8]. The Raspberry Pi was in short supply because it is not currently in production due to supply chain issues. This forced us to purchase a Pi from a 3rd party vendor via Amazon.com for far above the manufacturer's suggested retail price. Because of this, obtaining the Raspberry Pi was one of the largest obstacles to manufacturing the device.

Cost Constraints

The biggest economic concern was acquiring our main microcontroller, a Raspberry Pi Model 4 B. Due to shortages and supply-chain issues, Raspberry Pi's were in extremely short supply, causing price gouging on the few boards in stock. The boards were only available from third-party sellers for around $160 while the manufacturer's retail price on an industry website such as Digikey is $45 [9]. The fabrication and assembly costs for the custom printed circuit board (PCB) were also one of the main expenses, running at approximately $55 per board. The first PCB had component soldering issues, so the board was redesigned and reprinted, doubling the cost. Lucia had access to a 3D printer and was able to print the Samplisizer case for free. For demo and testing purposes a microphone was purchased for $10, and a piano and speaker were sourced for free. The microphone, piano keyboard, and speakers would not be included with the Samplisizer if it were to be sold on the market, so they are not included in market cost consideration. All board components were either significantly less expensive or freely sourced from past projects and we were able to stay well within the $500 constraint. Overall, the total costs came to just under $350. If the Raspberry Pi could have been purchased at the regular retail price the expenses would be reduced to $235.

Tools Employed

Several different software and hardware tools were utilized throughout the semester to design and test the project. The tools are detailed below.

Hardware:

To create the custom header board schematics and PCB layout, our team used KiCad software [10]. This was extremely useful because it allowed for the creation of schematics, layout, and Gerber files [11] all in one program. In order to simulate the volume circuit, Multisim was used [12]. Finally, the Advanced Circuits FreeDFM tool was used to check that the board was possible to manufacture [13].

Software:

The code for this project was written in the Python programming language [14] due to the previous knowledge and comfort level of all group members with the language as well as the abundance of pre-built libraries available. The code was written, edited, and debugged using the Visual Studio Code integrated development environment (IDE) [15] with the Pylint [16] extension to assist with writing clean code. To collaborate on a shared codebase we used Git [17] for version control and GitHub [18] to host our repository. Additionally, to ensure high-quality code standards, automated code quality scans were implemented for all pull requests in our repository using Sonarcloud [19].

We used Raspberry Pi OS (also known as Raspbian) [20] as the operating system (OS) on the Raspberry Pi. This OS was designed specifically for use on Raspberry Pi and as such comes with the appropriate drivers while also being lightweight enough to run on a Raspberry Pi. We utilized the PSOLA Python library [21] as well as the Python Pitch Shifter repository [22] and the Python Pianoputer repository [23] to implement the audio processing in software. The user-recorded Waveform Audio File Format (WAV) file will be pitch-shifted by various steps to correspond to each of the keys on the keyboard. These will then be individually exported to WAV files for each key without changing the speed or length of the recording [24].

In order to ensure that the audio samples were shifted by the correct pitch, the digital audio editor and recording application software Audacity was used to measure the frequencies [25].

3D Modeling:

To design the case for our device we used 3D modeling in Fusion 360 [26] because our group had previous experience using the software. We printed the case using an Ultimaker S3 printer with Ultimaker-brand polylactic acid (PLA) filament which Lucia has free access to in the Introduction to Engineering laboratory [27].

Societal Impact Constraints

Environmental Impact

Our team's personal impact on the environment includes the amount of electricity we used to power our personal computing devices as well as the project itself during research and testing. We must also acknowledge all the negative environmental impacts accrued from before the products reached our hands. The pre-recycling design lifecycle of all the electronic components in our project includes these steps: raw material collection, manufacturing, transportation, and finally usage. Each of these steps involves some new materials to be harvested, some energy expenditure, and some waste creation [28]. In order for companies to make the electronic components for our project, including the printed circuit board and Raspberry Pi, many raw materials must be mined or otherwise collected. These materials include but are not limited to plastics, copper, tin, silver, palladium, lithium, and gold [28]. Mining is detrimental to the environment and nearby ecosystems, and the mining process requires an inordinate amount of freshwater [29]. All of this also requires electricity and the drilling of fossil fuels which releases greenhouse gases into the atmosphere and often pollutes water sources [30]. After the production of these materials into components is complete, there is also the transportation of the goods to retailers and to us that further negatively impacts the environment due to fossil fuel usage.

Sustainability

The long-term costs of using this device will be low as the Raspberry Pi only consumes 6 Watts of electricity while utilizing all four cores under load [31]. Additionally, the device will only be plugged in while in use and therefore it won't be consuming electricity most of the time. As described in the Environmental Impact section above, there are still cases of before and after our product usage that must be considered. Once our instrument becomes obsolete in the future or if a part breaks and the product must be disposed of. If a consumer is extremely sustainability-conscious, they could go through and open the entire box and dispose of the parts individually. Parts of the metal and plastic in each of the electronic parts could be recycled, depending on the local waste management company offerings, which still uses a small amount of electricity. More likely though, the entirety of the plastic, electronics, and wood will be thrown away and contribute to the waste in landfills, potentially leaking harmful chemicals into the ground or releasing harmful toxins into the air if burned [32].

Health and Safety

If we are to market our product to consumers, then all of the aspects of our instrument will need to be evaluated according to the Consumer Product Safety Act [33] and any addendums as outlined by the United States Consumer Product Safety Commission [34]. These include safety standards required of all consumer products. As our product will only be for consumers aged 13 and older, we are not required to conform to the additional safety standards for children's products [35]. The products and parts we are purchasing have already passed all of the consumer product safety tests, and the board we are creating will be manufactured to meet these standards

as well. All of the parts we are creating and combining will be housed in a central plastic box that will be permanently sealed at production. The user will only be able to interact with the standard connectors to plug in the consumer products (USB-C power, microphone, keyboard, and speaker) to the box. The user will also be able to view the LEDs on the box top and interact with the knobs but will not be able to see or mess with any of the electrical connections or components, therefore eliminating the majority of the potential risk for the user.

Ethical, Social, and Economic Concerns

While creating a useable device, accessibility has to play a large role. We hope our device can be used by individuals with a wide range of abilities, and therefore we are using a standard input interface (MIDI) which will allow a user to plug in any input device of their choice preventing our device from being restricted to individuals with the dexterity to play a keyboard. Additionally, we are creating a device that can create loud and repetitive noises and therefore could be uncomfortable for people especially children with Autism [36]. Ideally, the user of our device would be cognizant of those around them so as not to make others uncomfortable. The device will also allow for the volume of the output to be adjusted and it will be able to be outputted to headphones so only those who want to hear can.

External Considerations

External Standards

We wanted our project to be as accessible and versatile as possible, which means we needed to interface with many standardized protocols to ensure compatibility. Namely, our project needed to work within the specifications of MIDI, aux connectors, and USB-C standards. For MIDI compatibility we used a standard USB-A jack and communicated at the given 31.25 (+/- 1%) Kbaud rate [37] over a USB cord. A library was also installed to interpret MIDI signals on the software side. For aux output, our hardware was made to handle input impedance of 10 or higher in accordance with the standards of using a 3.5mm audio jack. Furthermore, most aux inputs expect a nominal voltage input of -10 dBV so we tuned our hardware to output voltage around that range [38]. Since the microcontroller we are using for our project takes power input via a USB-C connection, we will simply need to use a standard USB-C power adapter to provide power to the system. We also used a customized PCB, so to ensure good manufacturability and functionality, all circuit board standards for all phases of the PCB design process were followed. Most notably we followed the standards laid out in the IPC-2581 [39] and IPC-2221 [40] codes. IPC-2581 outlines the standard way of organizing the PCB design information for communication with manufacturers. We followed these standards to ensure that our design could be understood and manufactured consistently. IPC-2221 gives standards for the basic PCB acceptance. Our board followed these standards to ensure optimal manufacturability of our design.

Furthermore, our project deals with potentially hazardous voltages so all electronics are contained in a Class 1 National Electrical Manufacturers Association (NEMA) enclosure [41]. The container will be sturdy and tight with openings only for necessary user interfaces. This enclosure will protect the electronics from potential water exposure and general physical wear to help prevent malfunction, but in the event of an unexpected short or voltage surge, the insulating container will also protect the user from dangerous shocks.

Intellectual Property Issues

A very similar patent is from Intel and describes a portable hand-held music synthesizer and networking method and apparatus [42]. This encompasses material extremely similar to our project and would likely cause intellectual property issues if we attempted to patent The Samplisizer. This pocket music synthesizer resembles ours in that it also utilizes a MIDI connection and the WAV file audio format.

There is also a patent filed by Yamaha Guitar Group detailing a stringed instrument (guitar) for connection to a computer to implement DSP modeling [43]. The patent describes using this guitar plugged into a laptop as a synthesizer for mimicking other instruments and noises and manipulating the audio. This is similar to our Samplisizer except that instead of having the hardware in a separate case with standard MIDI instrument ports to allow for interchangeability, the hardware is built into the electric guitar and must be controlled through a computer. If we chose to demonstrate our capstone with a guitar instead of a piano this would be slightly more similar.

Finally, there is a patent for the pitch synchronous overlap and add algorithm (PSOLA) that allows for an audio file to be shifted up or down in pitch without changing the length of the sound [44]. This algorithm is used in our Python code for pitch shifting.

Detailed Technical Description of Project

The project is a fun and interactive instrument that allows a user to play a recording of their own making as notes with a MIDI-compatible instrument of their choice. The device is powered off of a Universal Serial Bus Type-C (USB-C) [45] cable and controlled by a Raspberry Pi 4 Model B. Audio is recorded via a microphone when a button labeled “Record” is pressed and passed into the Raspberry Pi via a USB compatible microphone. The Raspberry Pi runs the 32-bit Raspberry Pi OS which is a Debian-based [46] version of Linux [47] designed for the Raspberry Pi.

In software, the recording is pitch-shifted up and down in increments of a half step for 43 notes and then stored in an SD card via the built in SD card reader in the Raspberry Pi. This is done using the Pitch Synchronous Overlap and Add algorithm (PSOLA) [21]. The PSOLA algorithm was chosen because can modulate an audio sample without changing the playback speed. The

user can then play these notes via any MIDI over USB compatible instrument (a keyboard will be used for our testing). The MIDI messages trigger the Raspberry Pi to send the correct stored audio sample to a digital-to-analog converter (DAC) which then goes into a knob-controlled analog volume control [48]. Finally, the analog signal is output to a powered speaker via an aux cable.

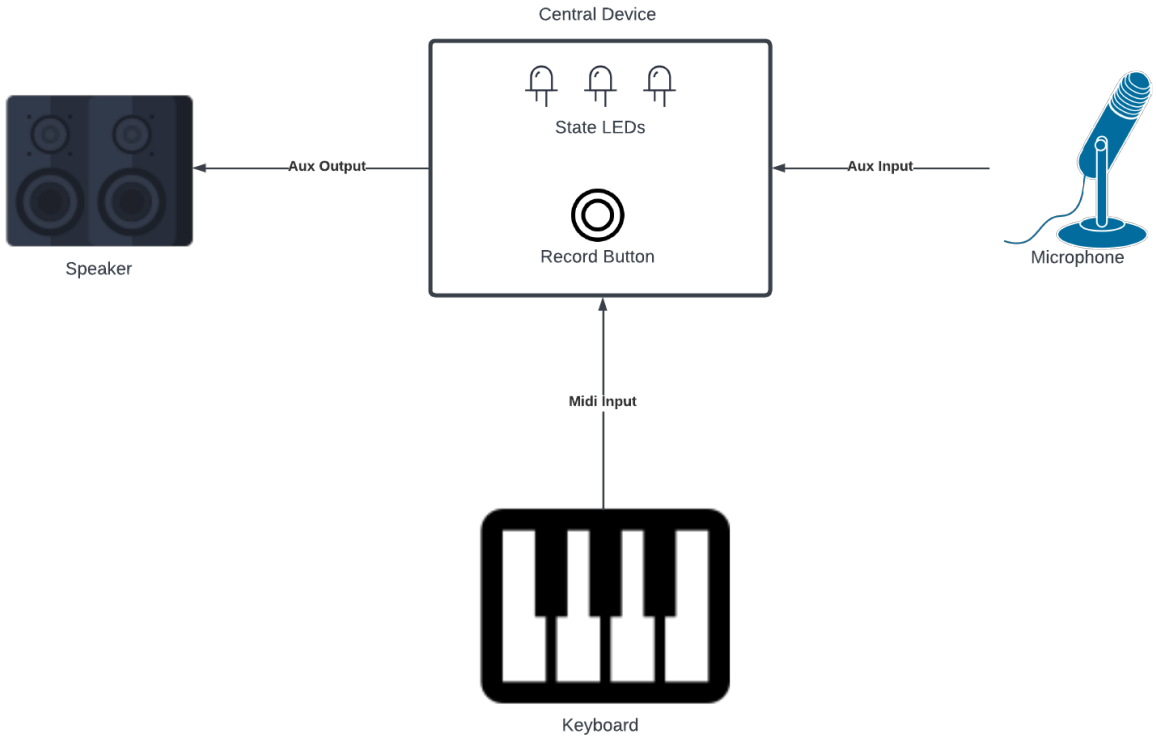

Figure 1 Basic Block Diagram

For usability, there are three Light Emitting Diodes (LEDs) placed centrally on the case of the device. The first LED is red and labeled “Recording”. which blinks three times after the record button is pressed and then illuminates for three seconds while the device is actively recording. The second LED is yellow and labeled “Processing”. It is illuminated after the user finishes recording, while the audio is being processed. The last LED is green and labeled “Ready” and is illuminated when the device is ready to be played. When the record button is pressed again it overwrites the old sample and the new sampled audio can be played after it has finished processing.

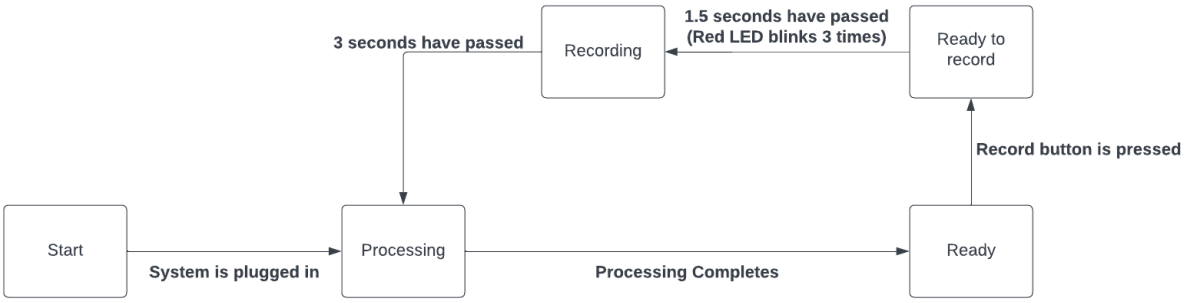

Figure 2 High Level System Design

Software Design

The business logic for the Samplisizer exists in one python program. This program's code is split into distinct sections to handle the different elements of the device's functionality. The sections are the pitch shifting logic, the audio recording logic, the general-purpose input/output (GPIO) logic, and the MIDI handling.

The pitch shifting works by first breaking up the audio wave file input into very small segments. When a higher pitch is desired, first the small segment is stretched while conserving the pitched and then sped up so that the sound is the same length as when recorded. Conversely, in order to lower the pitch the sound is overlapped and then slowed down so the sound file is also still the same duration.

The program is able to record audio through a USB connected microphone. This was implemented using the PyAudio [49] and Wave [50] Python libraries. The program first sets the parameters for the recording such as the resolution, the sampling rate, and the length of the recording (set to 3 seconds). Then it creates a PyAudio instantiation before it creates a PyAudio stream with the aforementioned parameters. Subsequently the recording starts. It records by looping through the audio stream and appending audio chunks to a frame array. Then once 3 seconds from the start of the recording pass the stream is stopped, and closed, and the PyAudio instantiation is terminated. Once the sample has been recorded, the program trims off any dead space at the start of the WAV file setting the start of the file to be the start of the audio chunk with an absolute value over 500.

The GPIO is handled using the RPi GPIO library [51]. The GPIO is responsible for turning the status LEDs on and off as well as registering button clicks for the record button. The GPIO functionality in the programs works first by initializing the pins via the library's setup function. The pins being used for the LEDs are set to outputs and the pin being used for the button is set to an input with a pullup resistor. The way the GPIO is used to indicate the status of the device via LEDs during operation is through the use of four functions called recording, processing, ready to record, and ready. These functions correspond to the states of the finite state machine shown in Figure 2 above. Each function is responsible for turning on the correct LED and turning off all other LEDs. The recording state has just the red LED on. The processing state has just the yellow

LED on. The ready state has just the green LED on. Finally, the ready to record state blinks the red LED three times with all of the other LEDs off. The blinks happen every quarter second and the delay after the last blink is 0.22 seconds to account for the execution time of the program as it transitions to the recording state.

To interface with the MIDI controller, the program made use of the python library rtmidi [52]. This library allowed the programmers to set a callback function that would execute immediately on MIDI input, which would have access to the MIDI message as well as any one user variable. The MIDI message would contain the number corresponding to the note played and whether it was a press event or a release event. The program passed in a list of sound objects to this function and used the note number to map the played note to the index of its corresponding sound object in the list. That sound would then either be played or stopped depending on whether the note was pressed or released. Whenever a new sound was recorded, the sound objects would be regenerated from the new WAV files and the list would be passed again to the callback function to keep it up to date.

Hardware Header Board

The PCB header, or Pi Hat, had multiple functions. These include generating an Inter IC-Sound (I2S) signal that encodes the analog microphone audio, controlling the volume output of the device with a potentiometer, and housing spots for the button and LEDs that were controlled by GPIO. The PCB header board interfaced with the Raspberry Pi through a 2x20 pin jack that plugs directly into the pi's GPIO. The entire board has 3.3 V electrostatic discharge (ESD) protection diodes on any node that may come into contact with a static shock.

The basic flow of the board can be seen in Figure 3. First, the analog microphone signal is input onto the board through a 3.5 mm headphone jack. This signal gets filtered through a basic low pass filter before entering into the ADC circuit. The schematic of the ADC can be seen in Figure 4. This particular ADC is a PCM1808 which is able to sample stereo audio at up to 100 kHz at 24-bit resolution [8]. For WAV file standards, we configured the ADC to sample at 44.1 kHz and set the PCM1808 to run in master mode. This can be changed after the board is printed with jumpers to change FMT, MD0, and MD1 from high to low. Because we are recording in mono, the left and right channels of the ADC receive the same analog signal. Master mode means that the PCM1808 only needs a clock signal input into it from SCKL and it will then generate the different I2S channels to be read by the RPi (DOUT, LRCK, BCK). The schematic used was based off of the TI documentation for the PCM1808 and also a datasheet for a TI development board that included the PCM1808 [53]. The I2S signals from the ADC go directly into the GPIO of the RPi to be read and converted into a WAV file. Unfortunately, this ADC circuit ended up being redundant in the final product because we switched from a 3.5 mm cable microphone to a USB microphone.

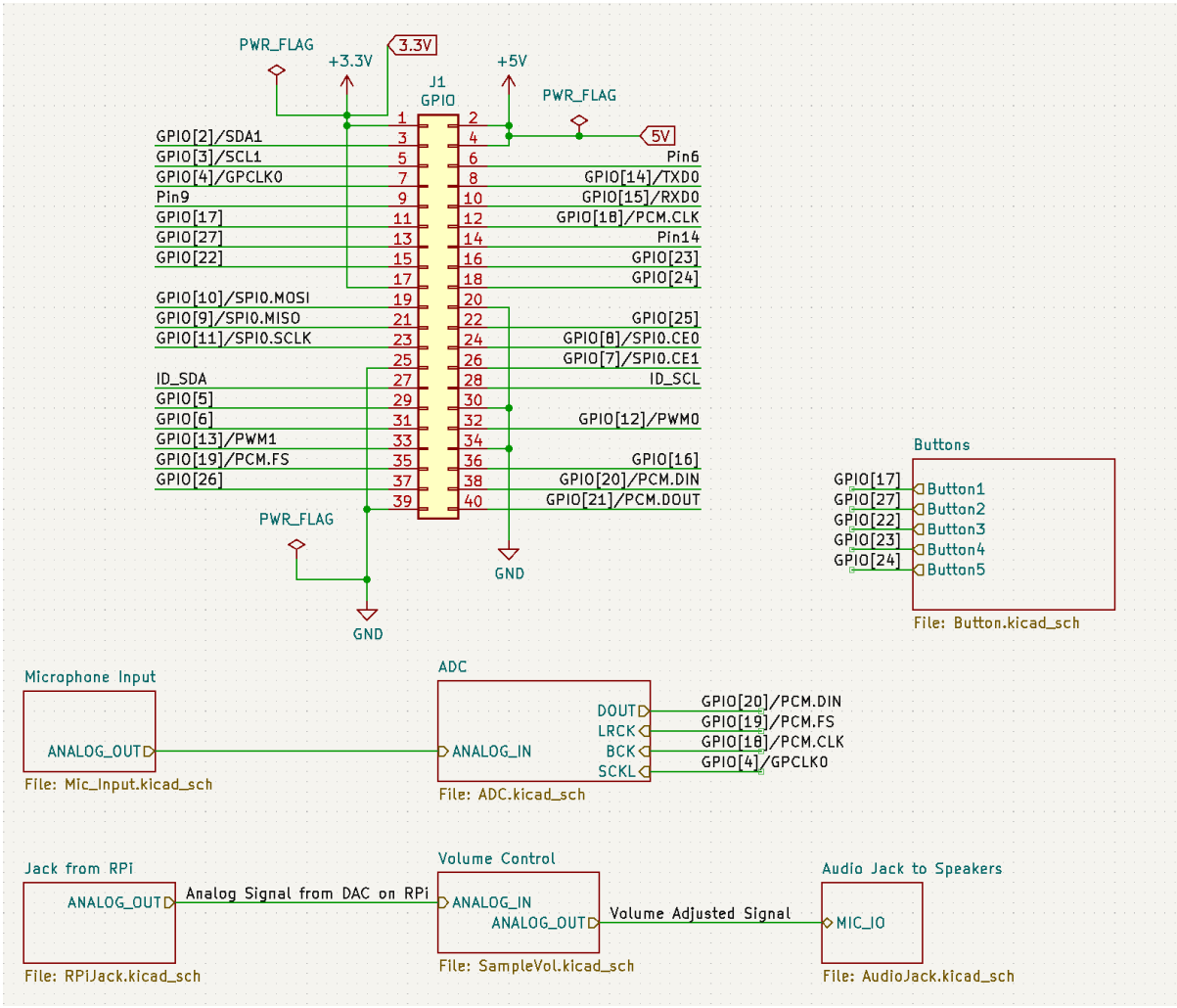

Figure 3 Header Board Block Diagram

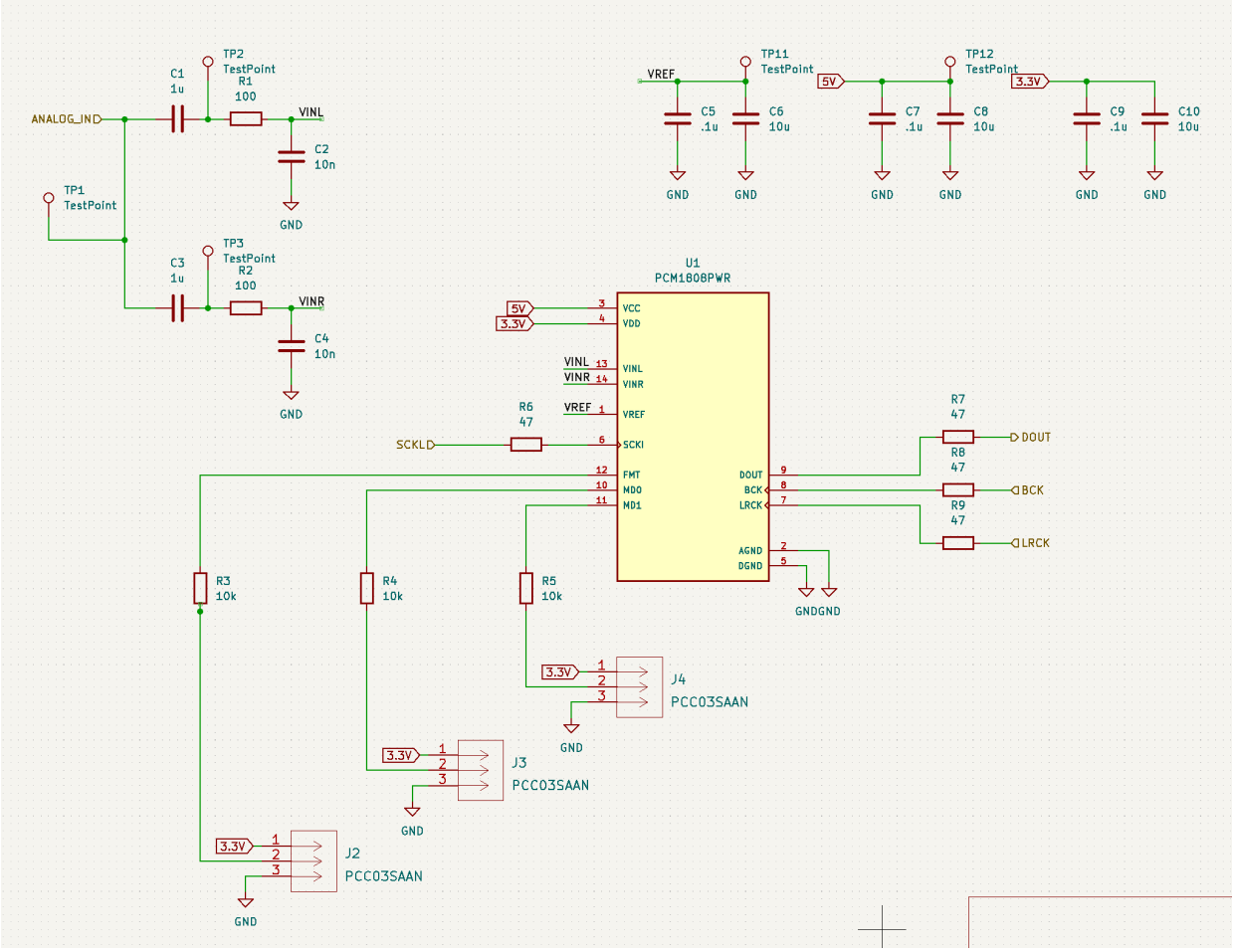

Figure 4 ADC Schematic

When a keyboard key is pressed after the RPi is finished processing it outputs an analog audio signal back into the board through another audio jack. This signal then passes into an analog volume control circuit based on the one seen in Figure 4. This circuit has a roughly logarithmic response versus the percentage of the potentiometer that is turned. One change that was made in our design was changing the value of R3 so that the maximum gain of the circuit is 0 dB so that we would not over volt the speakers. Because our board did not have access to negative voltage, our rails could only go from 0 to 5 V. This required that we biased the input up by 2.5 V DC and then referenced 2.5 V instead of ground. The reference voltage was created with an op amp voltage follower and the signal was biased with the capacitor and resistor network made of C18, R15, and R16. This also necessitates the need for high pass filtering after the volume circuit to return the DC bias to 0 V which can be seen in C19 and R17 in Figure 6. After the volume control circuit, the analog sound is output to a third audio jack that goes to the speakers.

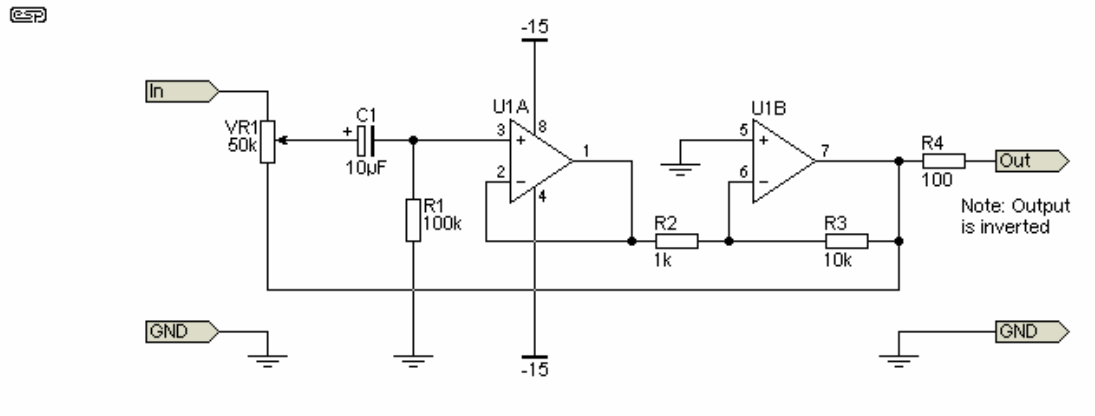

Figure 5 Logarithmic Volume Control Circuit

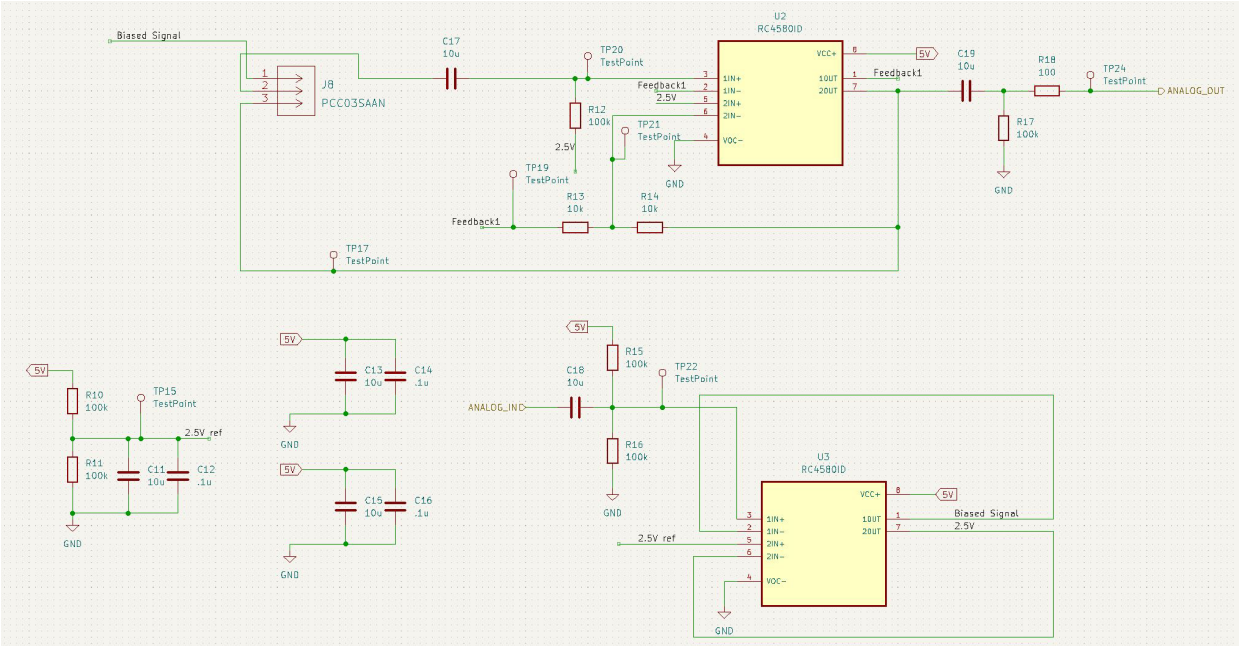

Figure 6 Modified Log Volume Control Circuit

The board also houses a pull up circuit to protect the button inputs. The schematic is shown in Figure 7. Five button handling circuits were put onto the board to accommodate the potential need for more inputs but in the end only one button was needed. When a button is pressed, the corresponding GPIO signal flips from high to low. This GPIO button signals the RPi to start recording a new sample.

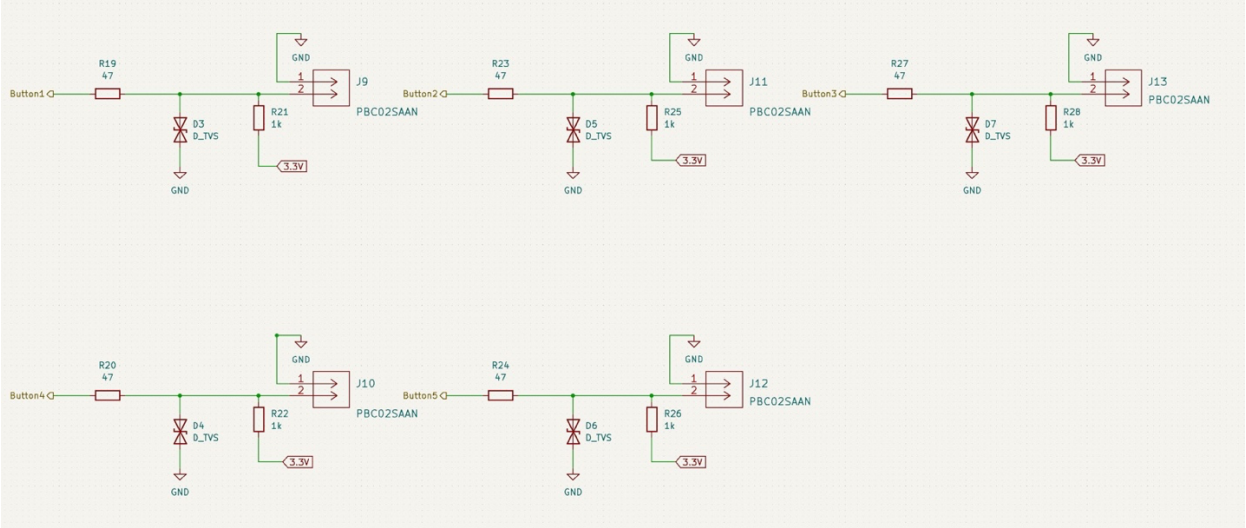

Figure 7 Schematic of Button Handling Circuit

After the schematics were designed, the circuits were laid out on a PCB template from Adafruit [54] for making Pi hat PCBs. An overview of the layout with top and bottom traces can be seen in Figure 8. It is important to note that there is a top and bottom copper ground plane. This is to reduce noise on the ADC by having a ground trace between the channels to minimize signal coupling. There are also many ground vias dispersed across the board to equalize voltage between the planes and eliminate islanding that occurs from the traces. All of the resistors, capacitors, and diodes use 0603 packaging [55] in order to save space while still being capable of hand assembly.

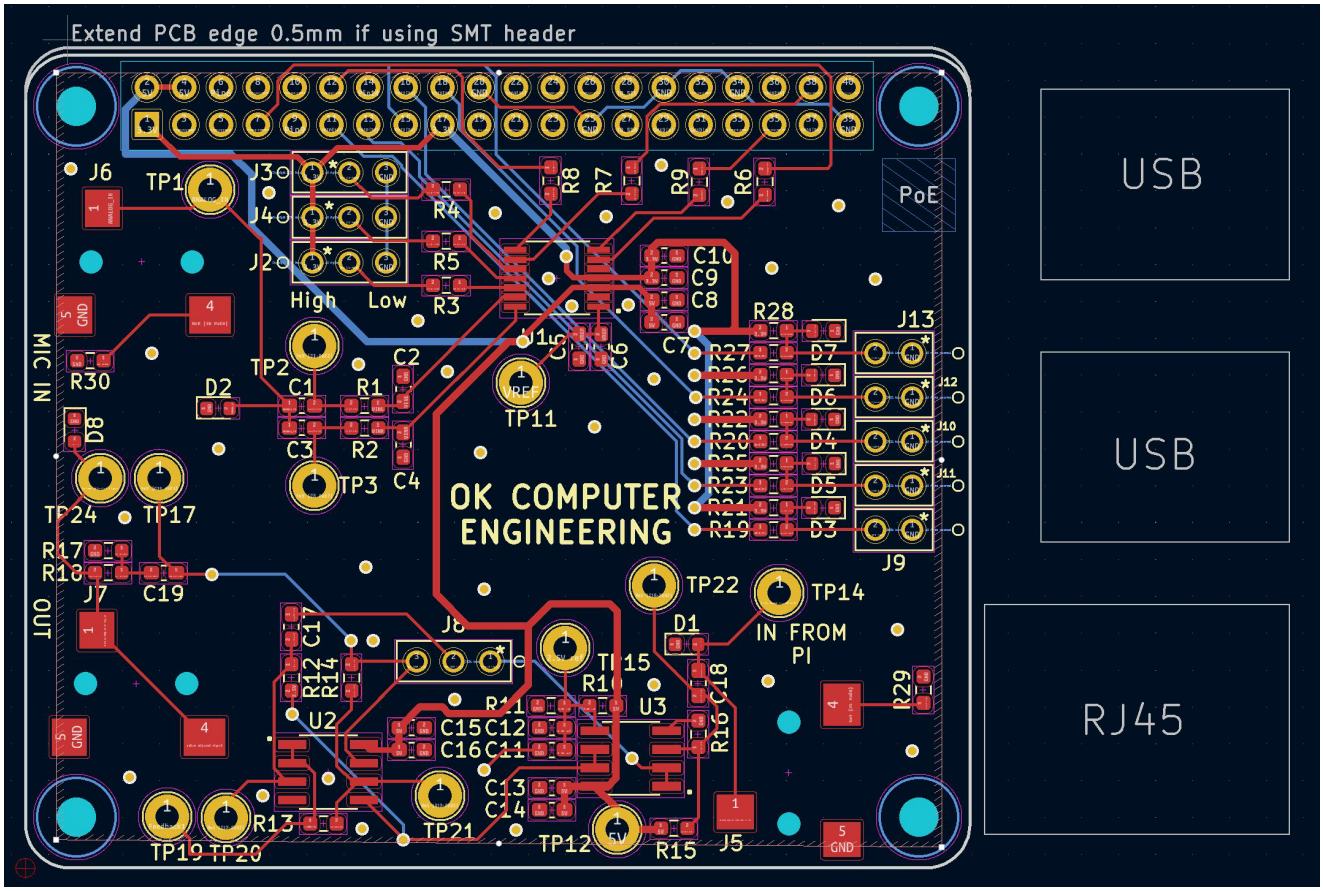

Figure 8 PCB Layout

The above figure is the second version of the board that we had printed. The original board had a couple of issues. The first was that one of the 3.5 mm headphone jacks was facing towards a USB port meaning that it was impossible to plug a cable in while the board was seated in the GPIO port. The second issue was that the ESD chips used had 0201 packaging which WWW was not able to hand solder. They also were unipolar meaning they could only prevent discharges with either positive or negative voltages but not both. These were swapped for bipolar diodes that used 0603 packaging. The last issue was islanding in the bottom ground plane under the ADC chip. This not only caused there to be drifting between the two different ground planes but also nullified the noise reducing effect between the audio input channels.

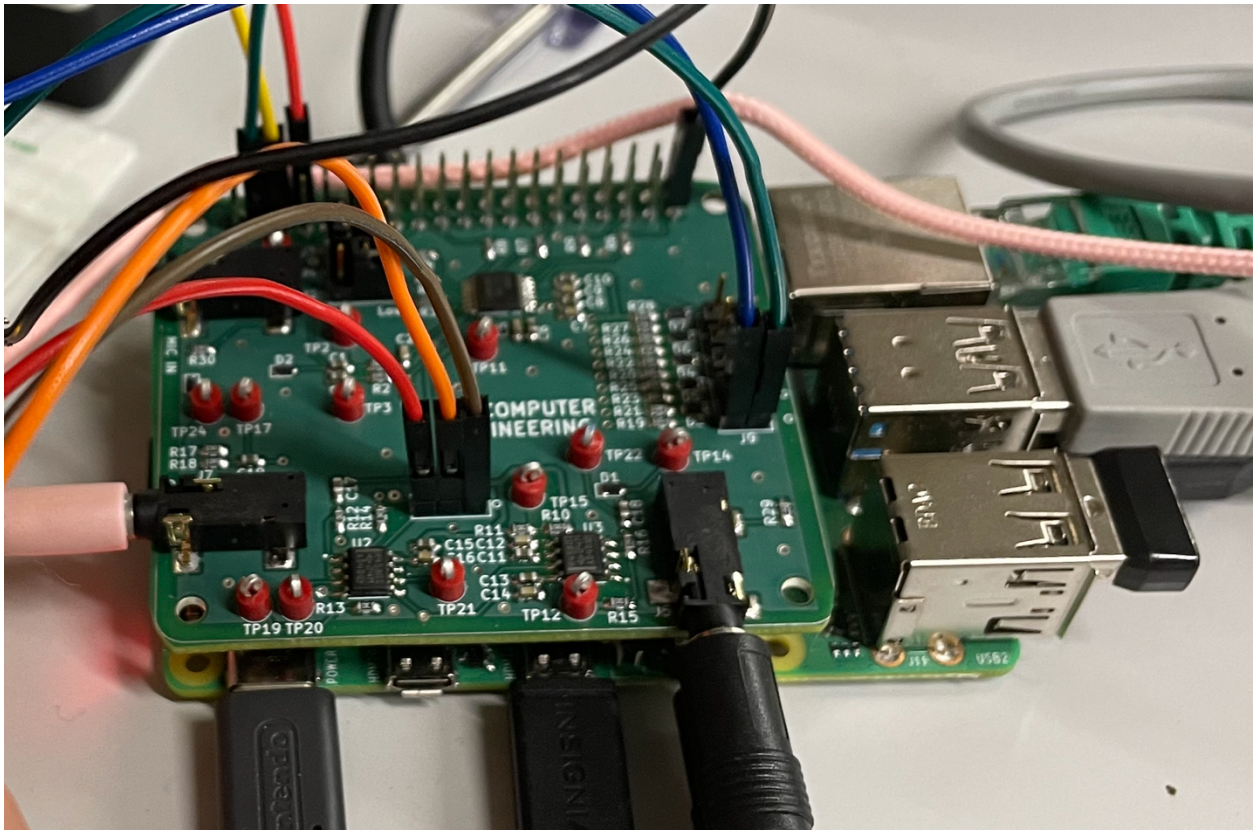

Figure 9 Finished PCB on the Raspberry Pi

Casing

In order to encase all the hardware elements into one product, a 3D-printed box was designed using Fusion 360 and printed using an Ultimaker S3 printer. The box consists of two halves where the top half can slide completely over the bottom half for a clean, simple aesthetic. The top half was printed in black plastic and the bottom half in white plastic to match the black and white keys on a piano. The two halves also allowed the case to be opened and the components inside to be inspected while testing. The USB and audio out ports were left exposed in the case so that the user can plug in their own microphone, MIDI instrument, and speakers or headphones. The remainder of the ports were covered and inaccessible to the user except for the ethernet port which was left exposed for group testing and debugging. If the Samplisizer were marketed, the ethernet port would be covered. The power cord goes through the back of the box so that the user can access the wall plug but cannot disconnect the power from the board. The case had holes for the volume control potentiometer, the record button, and the three LEDs to show recording, processing, and ready status. Each of these components were labeled for the user's understanding. The name of the product was also displayed on the top of the case.

Figure 1010 Assembled Case

Figure 1111 Case Ports

Project Time Line

Included below are a previous version and a current version of the team's Gantt Chart.

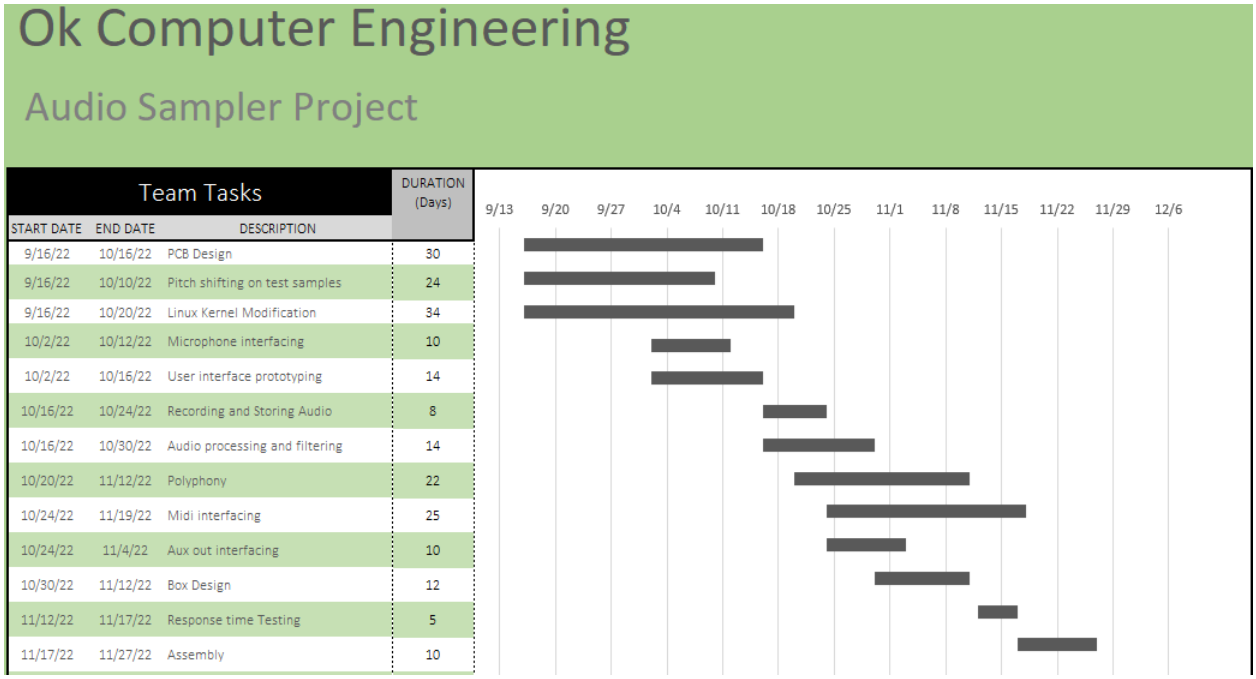

Figure 1212 Proposal Gantt Chart

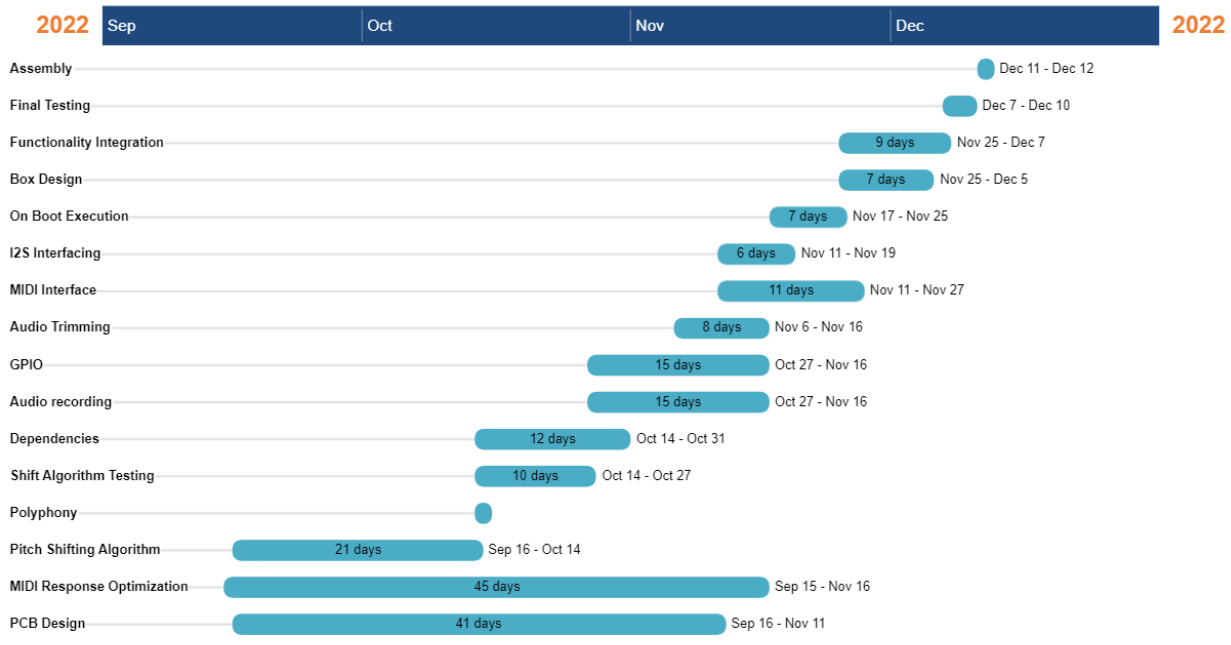

Figure 1313 Updated Gantt Chart

Serial tasks included the initial brainstorming of the idea for the Capstone project. Once the group knew we were going to create a sampling instrument, most work was completed in parallel. Frederick worked on designing the PCB while Lucia, Cooper, and Quinn focused on software-related tasks. Cooper set up the code management platforms and researched the I2S software requirements. Lucia researched pitch shifting algorithms and wrote the initial Python code to change the pitch of a hard-coded audio wave file up and down as well as mapping the transposed notes to a digital keyboard. Cooper assisted with integrating this code with a physical MIDI piano keyboard. Quinn focused on the real time MIDI library and helped with this integration as well. Frederick designed the PCB and sent it to WWW for manufacturing and finished assembling it returned from soldering. All of this was done in parallel throughout the semester.

Towards the end, the group came back together and worked in serial on testing the board and code together once integrated fully. Lucia designed and 3D printed the case for the hardware of the instrument. Cooper and Quinn configured the code to run on the Raspberry Pi on boot so that the system could be headless. Quinn added audio trimming functionality to cut out dead space during this time as well. All group members helped set up for the demo and present the final capstone project at the final event.

Our schedule changed greatly from our initial projections. Some tasks ended up being slightly longer than we expected. For instance, we had some issues with our initial PCB design and had to manufacture it again. Other tasks were far easier than anticipated. Polyphony, for example, turned out to be a trivial implementation with the right python library. Some tasks we simply did not anticipate needing. We hadn't considered the need to trim the audio until we tested a sample audio that had a pause at the beginning. Even with these unexpected changes, we were able to stay on task and finish the project satisfactorily.

Test Plan

Our test plan is divided into three main parts: software, hardware, and integration. This is due to the software and hardware being developed independently until late into production when the systems are integrated together.

Software

Each section of functionality developed in software was tested individually before being integrated into the larger system. The first step of our test plan was to ensure that we could read MIDI inputs to the USB port on the RPi. This was done as soon as we had the RPi and the keyboard. The RPi was programmed to print out the key press in python. In parallel, we also tested the PSOLA algorithm's ability to take an existing WAV file and pitch shift it relative to the 12-tone chromatic scale while maintaining the original length of the audio. A WAV file of google translate pronouncing the word goat was used as the place holder sample for the PSOLA

algorithm. This was verified in Audacity using a spectrum analysis and finding the peak of the fundamental tone. To test the recording functionality sounds were recorded on the RPi and played back both in audacity and using our software to verify that our software was up to par with a more traditional recording interface. The MIDI interface was tested by printing the MIDI messages to a terminal window on the RPi then verifying that the correct information was outputted for a given keypress.

Hardware

When the hardware board was returned from being manufactured, it was immediately beep checked. This entailed using an Ohm meter to measure the resistance between different nodes and ensure they aren't electrically shorted. It also allowed us to verify that there was no islanding in the ground plane. After beep checking, the board would be ready for assembly at WWW.

Once the board was assembled, 4 main board systems had to be tested. The first of these was the power supply. The board was plugged into the RPi and using a voltmeter all of the test points were checked to ensure that all the chips were receiving the correct voltage and that there were no unseen shorts. Next was the analog to digital converter, which had to be configured in master mode using jumpers and fed a clock signal. Using an AD2 as an oscilloscope, the output of BCK, LRCK, and DOUT was verified. Then, the volume control circuit was tested. The 2.5 V reference and the DC biasing network were measured to check their voltages were at the correct values. A network analysis from the input to the output was run from 0 Hz to 20 kHz on the volume circuit for different rotations of the potentiometer. This ensures there is no filtering, and the circuit can logarithmically change the volume of all frequencies equally (the plot should be a perfectly horizontal line at or below 0 dB). Finally, the off board GPIO circuits were tested. This involved running a 3.3 V source across each color of LED and ensuring their relative intensities were the same. If they were different, the corresponding resistor would be changed to allow the desired amount of current to flow through the diode. Also, the button ports were tested to ensure the GPIO is pulled up to 3.3 V when the button is off, and changes to 0 V when the button is pressed.

Integration

After the board and software were tested individually, it was time to test them together. The first test was to verify that a button press could be read by the RPi and then trigger an LED to turn on. Then, the Pi's ability to run the analog to digital converter was tested. This requires the Pi to produce a high frequency clock from GPIO to run the ADC and then detect the ADC as a valid I2S recording device to read the signal as a WAV file. In our testing, the ADC produced the correct signals, but the Pi couldn't identify the ADC as a valid device. Because of time constraints, we opted to use a USB microphone instead of the ADC I2S. Finally, the total system was tested from recording to playing. This includes pushing keys at unexpected times, holding the record button, and playing while the Pi is processing audio.

Final Results

Our final product successfully achieved our goal of creating a fun, versatile, accessible, instrument. We were able to fulfill almost completely all the requirements we set out for ourselves at the beginning of the semester. At the final demonstration, the Samplisizer interfaced with a USB microphone, a piano midi controller over USB, and a speaker over a 3.5mm headphone jack. The casing for the project included an exposed button labeled “record,” that when pressed, initiated the recording process. A red LED would blink three times then go solid for three seconds while the program began recording audio data. A yellow LED would then illuminate while the audio was shifted up and down the 12-tone chromatic scale. On average the processing time fluctuates around the required 1 minute to complete, ranging from 50 to 70 seconds. The resulting audio files were clear and consistent, fulfilling our requirements of minimal audio distortion with no speed variation. With a good recording, the user could then play the sound responsively on the midi controller with little delay. If a recording was made in a noisy environment, the trimming program would mistake background noise as the start of the program causing any pause in the beginning of the recording to hurt responsiveness. Otherwise, the response latency was barely noticeable with no harm to the instrument's playability.

Figure 1414 Final Setup

Costs

Our project ended up costing $341.88 overall, well below the maximum budget of $500. The highest cost was the Raspberry Pi 4 Model B bought from Amazon at $160, which was approximately $115 above regular price due to supply chain shortages and price gouging. The next highest costs were the manufacturing and assembly costs for the original custom PCB as well as the fixed reprint, coming in at around $60 each. Following those main expenses, the microphone was $12.99 from BestBuy. The piano, speaker, and 3D printed case were borrowed or acquired for free. All of the software and other tools used were freely available. All together the remaining hardware components, such as resistors and capacitors, cost $48.99. Certain components were sourced for free from past projects, including the buttons and LEDs. The full breakdown of costs can be found in the budget spreadsheet in Appendix A.

If 10,000 units were manufactured, there would be significant discounts on most of the components. The total cost per product would be closer to $110, approximately 1/3 of the price when making a singular Samplisizer. More specific information can be found in the manufacturing budget spreadsheet in Appendix B.

Future Work

The Samplisizer provides functionality that can easily be built upon in addition to the ability to use the device as a component in a larger modular system. The device could be extended in both hardware and software to add a variety of new features and functionality. For example, arpeggiation could be added to the software and due to our custom PCB having the capacity to support 5 buttons a hardware interface could be added as well. Another intuitive next step would be the ability to store multiple samples. We used approximately 10 Gb of the 32 Gb SD card used to store the operating system and the samples. Thus, allowing ample storage space for multiple samples to be stored. Again, more buttons could be added to provide users with the ability to switch between stored samples.

The Samplisizer uses standardized connectors and protocols for its inputs and outputs meaning additional devices could be created that could connect with the Samplisizer to enhance and extend its functionality. The Samplisizer uses the MIDI protocol to interface with the device so any MIDI-compatible device could be used as a controller. Therefore, any custom or unique MIDI instruments developed in the future could be used to control the Samplisizer. Additionally, the Samplisizer outputs audio via an AUX port allowing for to be passed through one or more effects pedals such as a wah-wah pedal.

References

[1] J. Gibson, "The MIDI Standard," *Introduction to MIDI and Computer Music*. https://cecm.indiana.edu/361/midi.html (accessed Sep. 25, 2022).

[2] "YellowNoiseAudio." https://www.yellownoiseaudio.com/about (accessed Sep. 25, 2022).

[3] "SamplerBox." https://www.samplerbox.org/ (accessed Sep. 25, 2022).

[4] "Raspberry Pi," *Raspberry Pi*. https://www.raspberrypi.com/ (accessed Sep. 25, 2022).

[5] "MSP430 microcontrollers | TI.com." https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/msp430-microcontrollers/overview.html (accessed Sep. 25, 2022).

[6] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Pearson Education, 1991.

[7] "WWW Electronics Incorporated." https://3welec.com/ (accessed Dec. 09, 2022).

[8] "PCM1808 data sheet, product information and support | TI.com." https://www.ti.com/product/PCM1808 (accessed Dec. 13, 2022).

[9] "RASPBERRY PI 4B/2GB," *Digi-Key Electronics*. https://www.digikey.com/en/products/detail/raspberry-pi/RASPBERRY-PI-4B-2GB/10258782 (accessed Dec. 09, 2022).

[10] "KiCad EDA." https://www.kicad.org/ (accessed Dec. 09, 2022).

[11] "Official Gerber Format Website," *Ucamco*. https://www.ucamco.com/en/gerber (accessed Dec. 13, 2022).

[12] "Multisim Live Online Circuit Simulator," *NI Multisim Live*. https://www.multisim.com/ (accessed Dec. 13, 2022).

[13] "FreeDFM - A Service of Advanced Circuits." https://www.my4pcb.com/net35/FreeDFMNet/FreeDFMHome.aspx (accessed Dec. 13, 2022).

[14] "Python." [Online]. Available: https://www.python.org/doc/

[15] "Visual Studio Code." [Online]. Available: https://code.visualstudio.com/

[16] "Pylint." [Online]. Available: https://pylint.pycqa.org/en/latest/

[17] "Git." [Online]. Available: https://git-scm.com/

[18] "Github." [Online]. Available: https://github.com/

[19] "Sonarcloud." [Online]. Available: https://www.sonarsource.com/products/sonarcloud/

[20] G. Hollingworth, "Raspberry Pi OS (64-bit)," *Raspberry Pi*, Feb. 02, 2022. https://www.raspberrypi.com/news/raspberry-pi-os-64-bit/ (accessed Sep. 25, 2022).

[21] M. Morrison, "psola: Time-domain pitch-synchronous overlap-add." Accessed: Sep. 25, 2022. [Online]. Available: https://github.com/maxrmorrison/psola

[22] C. Woodall, "Python Pitch Shifter." Sep. 08, 2022. Accessed: Sep. 26, 2022. [Online]. Available: https://github.com/cwoodall/pitch-shifter-py

[23] "Python, Pitch shifting, and the Pianoputer - __del__(self)." http://zulko.github.io/blog/2014/03/29/soundstretching-and-pitch-shifting-in-python/ (accessed Sep. 26, 2022).

[24] K. Iqbal, "WAV - Waveform Audio File Format," Dec. 13, 2019. https://docs.fileformat.com/audio/wav/ (accessed Sep. 26, 2022).

[25] "Home," *Audacity* ®. https://www.audacityteam.org (accessed Dec. 13, 2022).

[26] "Fusion 360 | 3D CAD, CAM, CAE, & PCB Cloud-Based Software | Autodesk." https://www.autodesk.com/products/fusion-360/overview (accessed Sep. 26, 2022).

[27] "Ultimaker S3: Easy-to-use 3D printing starts here," *https://ultimaker.com*. https://ultimaker.com/3d-printers/ultimaker-s3 (accessed Dec. 09, 2022).
[28] "Raspberry Pi," *Design Life-Cycle*. http://www.designlife-cycle.com/raspberry-pi (accessed Sep. 26, 2022).
[29] "What Commodities Are the Main Inputs for the Electronics Sector?," *Investopedia*. https://www.investopedia.com/ask/answers/042015/what-commodities-are-main-inputs-electronics-sector.asp (accessed Sep. 26, 2022).
[30] June 01 and 2022 Melissa Denchak, "Fossil Fuels: The Dirty Facts," *NRDC*. https://www.nrdc.org/stories/fossil-fuels-dirty-facts (accessed Sep. 26, 2022).
[31] "Power Consumption Benchmarks | Raspberry Pi Dramble." https://www.pidramble.com/wiki/benchmarks/power-consumption (accessed Sep. 26, 2022).
[32] "Soaring e-waste affects the health of millions of children, WHO warns." https://www.who.int/news/item/15-06-2021-soaring-e-waste-affects-the-health-of-millions-of-children-who-warns (accessed Sep. 26, 2022).
[33] L. M. Benson and K. Reczek, "A Guide to United States Electrical and Electronic Equipment Compliance Requirements," National Institute of Standards and Technology, Jun. 2021. doi: 10.6028/NIST.IR.8118r2.
[34] "CPSC.gov," *U.S. Consumer Product Safety Commission*. https://www.cpsc.gov/ (accessed Sep. 26, 2022).
[35] "Children's Products," *U.S. Consumer Product Safety Commission*. https://www.cpsc.gov/Business--Manufacturing/Business-Education/childrens-products (accessed Sep. 26, 2022).
[36] "Sound Sensitivity and Autism." https://ei.northwestern.edu/sound-sensitivity-autism#:~:text=Hyperacusis%20(say%20it%20with%20me,for%20your%20child%20to%20hear.
[37] "MIDI 1.0 Detailed Specification," *The MIDI Association*. https://midi.org/specifications/midi1-specifications/m1-v4-2-1-midi-1-0-detailed-specification-96-1-4 (accessed Sep. 27, 2022).
[38] Arthur, "Are AUX (Auxiliary) Connectors & Headphone Jacks The Same?," *My New Microphone*. https://mynewmicrophone.com/are-aux-auxiliary-connectors-headphone-jacks-the-same/ (accessed Sep. 27, 2022).
[39] "Home," *IPC- 2581 Consortium*. http://www.ipc2581.com/ (accessed Dec. 13, 2022).
[40] R. PCB, "What is IPC-2221 Standard?," *Printed Circuit Board Manufacturing & PCB Assembly - RayMing*, Aug. 19, 2022. https://www.raypcb.com/ipc-2221-standard/ (accessed Dec. 13, 2022).
[41] "NEMA/IEC Enclosure Ratings from Cole-Parmer." https://www.coleparmer.com/tech-article/nema-iec-enclosure-ratings (accessed Sep. 27, 2022).
[42] A. T. Wilson, "Portable hand-held music synthesizer and networking method and apparatus," US8288641B2, Oct. 16, 2012 Accessed: Dec. 09, 2022. [Online]. Available: https://patents.google.com/patent/US8288641B2/en?q=music+synthesizer&oq=music+synthesizer
[43] M. Ryle, M. A. Doidic, P. J. Celi, and A. Zak, "Stringed instrument for connection to a computer to implement DSP modeling," US7799986B2, Sep. 21, 2010 Accessed: Dec. 09, 2022. [Online]. Available: https://patents.google.com/patent/US7799986B2/en?q=audio+sampler+box+microphone+piano+instrument&oq=audio+sampler+box+microphone+piano+instrument

[44] O. Rosec and D. Cadic, “Procédé et dispositif de modification d’un signal audio,” EP1970894A1, Sep. 17, 2008 Accessed: Dec. 09, 2022. [Online]. Available: https://patents.google.com/patent/EP1970894A1/en?q=psola&oq=psola
[45] “USB Type C Compliance Document.” USB Implementers Forum, Inc. Accessed: Sep. 12, 2022. [Online]. Available: www.usb.org
[46] “Debian -- The Universal Operating System.” https://www.debian.org/ (accessed Sep. 25, 2022).
[47] “Linux.org,” *Linux.org*. https://www.linux.org/ (accessed Sep. 25, 2022).
[48] “David’s MIDI Spec.” https://www.cs.cmu.edu/~music/cmsip/readings/davids-midi-spec.htm (accessed Sep. 26, 2022).
[49] “PyAudio.” [Online]. Available: https://pypi.org/project/PyAudio/
[50] “wave.” [Online]. Available: https://docs.python.org/3/library/wave.html
[51] B. Croston, “raspberry-gpio-python.” [Online]. Available: https://sourceforge.net/projects/raspberry-gpio-python/
[52] C. Arndt, “python-rtmidi: A Python binding for the RtMidi C++ library implemented using Cython.” Accessed: Dec. 13, 2022. [MacOS :: MacOS X, Microsoft :: Windows, POSIX]. Available: https://github.com/SpotlightKid/python-rtmidi

Appendix

Appendix A: Project Budget

OK Computer Engineering Budget

ITEM	Description	Manufacturers Part Number	SOURCE	UNITS	$/UNITS	TOTAL SPENT
Board V1	Custom PCB printing and assembly	-	3W	1	$60.00	$60.00
Board V2	Custom PCB printing and assembly	-	3W	1	$60.00	$60.00
Raspberry Pi	Raspberry Pi 4 Model B	RPI4-MODBP-4GB	Amazon	1	$160.00	$160.00
Microphone	Singing Machine - Unidirectional Dynamic Microphone	SMM205	BestBuy	1	$12.99	$12.99
ADC	PCM1808 Single-Ended, Analog-Input 24-Bit, 96-kHz Stereo ADC	PCM1808PWR	Digikey	2	$2.52	$5.04
Audio OpAmp	RC4580 Dual Audio Operational Amplifier	RC4580ID	Digikey	3	$0.99	$2.97
Jack	3.50mm (0.141", 1/8", Mini Plug) - Headphone Phone Jack Stereo (3 Conductor, TRS) Connector Solder	35RASMT2BHNTRX	Digikey	4	$1.56	$6.24
Test Point	Red PC Test Point, Compact Phosphor Bronze Tin Plating Through Hole Mounting	5270	Digikey	15	$0.32	$4.80
Potentiometer	50k Ohm 1 Gang Logarithmic Panel Mount Potentiometer 1.0 Kierros Carbon 0.1W, 1/10W PC Pins	PDB181-K420K-503A2	Digikey	1	$1.47	$1.47
TVS Diode	8.9V Clamp 2.5A (8/20µs) Ipp Tvs Diode Surface Mount 2-X1SON (1x.60)	TPD1E04U04DPYT	Digikey	7	$0.94	$6.58
TVS Diode	8.9V Clamp 2.5A (8/20µs) Ipp Tvs Diode Surface Mount 2-X2SON (0.6x0.3)	TPD1E04U04DPLR	Digikey	15	$0.34	$5.16
40-Pin Header	Stacking Header 40 Pin Raspberry Pi - Connector	2223	Digikey	1	$2.50	$2.50
3-Pin Header	Connector Header Through Hole 3 position 0.100" (2.54 mm)	PCC03SAAN	Digikey	5	$0.46	$2.30
2-Pin Header	Connector Header Through Hole 2 position 0.100" (2.54 mm)	PBC02SAAN	Digikey	6	$0.22	$1.32
10 µF Capacitor	10 µF ±20% 6.3V Ceramic Capacitor X5R 0603 (1608 Metric)	JMK107ABJ106MA-T	Digikey	50	$0.04	$2.17
1 µF Capacitor	1 µF ±20% 25V Ceramic Capacitor X7R 0603 (1608 Metric)	TMK107B7105MA-T	Digikey	10	$0.08	$0.76
0.1 µF Capacitor	0.1 µF ±20% 50V Ceramic Capacitor X7R 0603 (1608 Metric)	CL10B104MB8NNNC	Digikey	100	$0.01	$1.17
0.01 µF Capacitor	10000 pF ±10% 50V Ceramic Capacitor X7R 0603 (1608 Metric)	06035C103KAT2A	Digikey	50	$0.02	$0.83
100 kOhm Resistor	100 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-07100KL	Digikey	1000	$0.004	$3.90
20 kOhm Resistor	20 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0720KL	Digikey	20	$0.02	$0.42
10 kOhm Resistor	10 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0710KL	Digikey	20	$0.02	$0.42
100 Ohm Resistor	100 Ohms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-07100RL	Digikey	20	$0.02	$0.42
47 Ohm Resistor	47 Ohms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0747RL	Digikey	20	$0.02	$0.42
SUBTOTAL						**$341.88**

Appendix B: Hypothetical Manufacturing Budget

OK Computer Engineering Budget - Mass Manufacturing

ITEM	Description	Manufacturers Part Number	SOURCE	UNITS	$/UNITS	TOTAL SPENT
Board	Custom PCB printing and assembly	-	3W	1	$60.000	$60.00
Raspberry Pi	Raspberry Pi 4 Model B	RPI4-MODBP-4GB	Amazon	1	$45.000	$45.00
ADC	PCM1808 Single-Ended, Analog-Input 24-Bit, 96-kHz Stereo ADC	PCM1808PWR	Digikey	1	$0.911	$0.91
Audio OpAmp	RC4580 Dual Audio Operational Amplifier	RC4580ID	Digikey	3	$0.423	$1.27
Jack	3.50mm (0.141", 1/8", Mini Plug) - Headphone Phone Jack Stereo (3 Conductor, TRS) Connector Solder	35RASMT2BHNTRX	Digikey	4	$0.900	$3.60
Potentiometer	50k Ohm 1 Gang Logarithmic Panel Mount Potentiometer 1.0 Kierros Carbon 0.1W, 1/10W PC Pins	PDB181-K420K-503A2	Digikey	1	$0.656	$0.66
TVS Diode	8.9V Clamp 2.5A (8/20µs) Ipp Tvs Diode Surface Mount 2-X2SON (0.6x0.3)	TPD1E04U04DPLR	Digikey	5	$0.113	$0.57
40-Pin Header	Stacking Header 40 Pin Raspberry Pi - Connector	2223	Digikey	1	$2.500	$2.50
3-Pin Header	Connector Header Through Hole 3 position 0.100" (2.54mm)	PCC03SAAN	Digikey	5	$0.208	$1.04
2-Pin Header	Connector Header Through Hole 2 position 0.100" (2.54mm)	PBC02SAAN	Digikey	6	$0.120	$0.72
10 µF Capacitor	10 µF ±20% 6.3V Ceramic Capacitor X5R 0603 (1608 Metric)	JMK107ABJ106MA-T	Digikey	5	$0.021	$0.10
1 µF Capacitor	1 µF ±20% 25V Ceramic Capacitor X7R 0603 (1608 Metric)	TMK107B7105MA-T	Digikey	5	$0.019	$0.09
0.1 µF Capacitor	0.1 µF ±20% 50V Ceramic Capacitor X7R 0603 (1608 Metric)	CL10B104MB8NNNC	Digikey	5	$0.007	$0.03
0.01 µF Capacitor	10000 pF ±10% 50V Ceramic Capacitor X7R 0603 (1608 Metric)	06035C103KAT2A	Digikey	5	$0.008	$0.04
100 kOhm Resistor	100 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-07100KL	Digikey	5	$0.002	$0.01
20 kOhm Resistor	20 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0720KL	Digikey	5	$0.002	$0.01
10 kOhm Resistor	10 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0710KL	Digikey	5	$0.002	$0.01
100 Ohm Resistor	100 Ohms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-07100RL	Digikey	5	$0.002	$0.01
47 Ohm Resistor	47 Ohms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Moisture Resistant Thick Film	RC0603FR-0747RL	Digikey	5	$0.002	$0.01
SUBTOTAL						**$116.59**