Portable In-line Audio Equalizer
Final Report

By

Ankit Jayant (ajayant2)

Avinash Subramaniam (avinash6)

Ji Yeon In (jiyeoni2)

Final Report for ECE 445, Senior Design, Spring 2021

TA: Prashant Shankar

05 May 2021

Project No. 8

Abstract

The Portable In-line Audio Equalizer (PIAE) is an eight frequency band audio equalizer that is intended to be portable for everyday use. It can boost or attenuate each frequency band up to a magnitude of 10 dB, and allows users to change those sound signatures in real time. Any media capable of listening (e.g. speakers, headphones) and producing sound (e.g. mobile phones, computers) can interface with the PIAE as long as they support 3.5 mm audio jacks. This report describes the design and functionality of the product, both in terms of its hardware and software.

Contents

1 Introduction

1.1 Objective

There are varying preferences to equalization (EQ) in audio, whether it is through personal preference or a need, such as helping people with a hearing impairment. Some media players do not have a built-in equalizer nor do they allow for downloading EQ mobile apps. Therefore, users are unable to adjust the sound signature of what they are listening to. Additionally, many pre-existing EQ devices are too large or heavy to be portable.

The solution is the Portable In-Line Audio Equalizer. Using the data from a desired media player, the PIAE uses signal processing algorithms to output audio data with a boost or attenuation at certain frequency ranges. This device allows for equalization, and has the advantages of convenient everyday use.

1.2 Background

Hearing loss can come in different ranges. One form of hearing loss to consider is a "notch" hearing loss, which is hearing loss at a certain frequency range [1]. In order to help with this type of issue, any desired frequency range can be boosted by an audio equalizer when using a media player. There are also people with personal preferences with sound signatures who use equalizers.

Some devices have built-in equalizers, like in computers and MP3 players, but that is dependent on the specific version and brand. Equalizer mobile apps can also be downloaded, but that is not possible for older devices, such as CD players.

Although portable audio equalizers exist, they typically have fewer operating ranges. Larger operating ranges allow for more options for the user, as well as a greater ability for the user to fine-tune the emphasis on the desired frequencies. This is especially important for users suffering from hearing loss. Commercial equalizers can have eight band filters, but those devices are not usable in a casual setting [2]. Portable devices are more convenient, but sacrifice performance by using fewer operating ranges [3]. The goal for the PIAE is to maintain the performance provided by commercial equalizers while also providing usability for everyday people.

The performance of an audio equalizer is not only restricted to operating frequencies, but also to latency. If the latency introduced by the PIAE is too large, then users may prefer comparable products with lower latencies [4]. This may drive down customer satisfaction and demand. Therefore, the PIAE is limited to having a latency below 100 ms.

1.3 High-Level Requirements

- The PIAE must have a latency of less than 100 milliseconds.
- The PIAE should use eight frequency bands when constructing its filters, instead of the typical three frequency bands. The frequency bands will be centered at the following frequencies measured in Hz : [100, 250, 500, 1000, 2000, 4000, 8000, 16000].
- The PIAE must have a size of less than 16 x 12 x 6 cm for the device to be sufficiently portable.

1.4 Summary

This report begins by detailing the design of the PIAE. The design is broken up into subsystems, which are further broken up into components. Each subsystem description includes a schematic as well as specific component descriptions, which covers required specifications, mathematical equations associated with the component, and software associated with the component. Secondly, the report details the design verifications that were conducted for each component. These verifications ensure that the components and subsystems work as expected. Then, the report enumerates the costs and parts necessary for this project. Finally, an overview is provided concerning the accomplishments, uncertainties, ethical considerations, and future work with respect to the project.

2 Design

2.1 Block Diagram

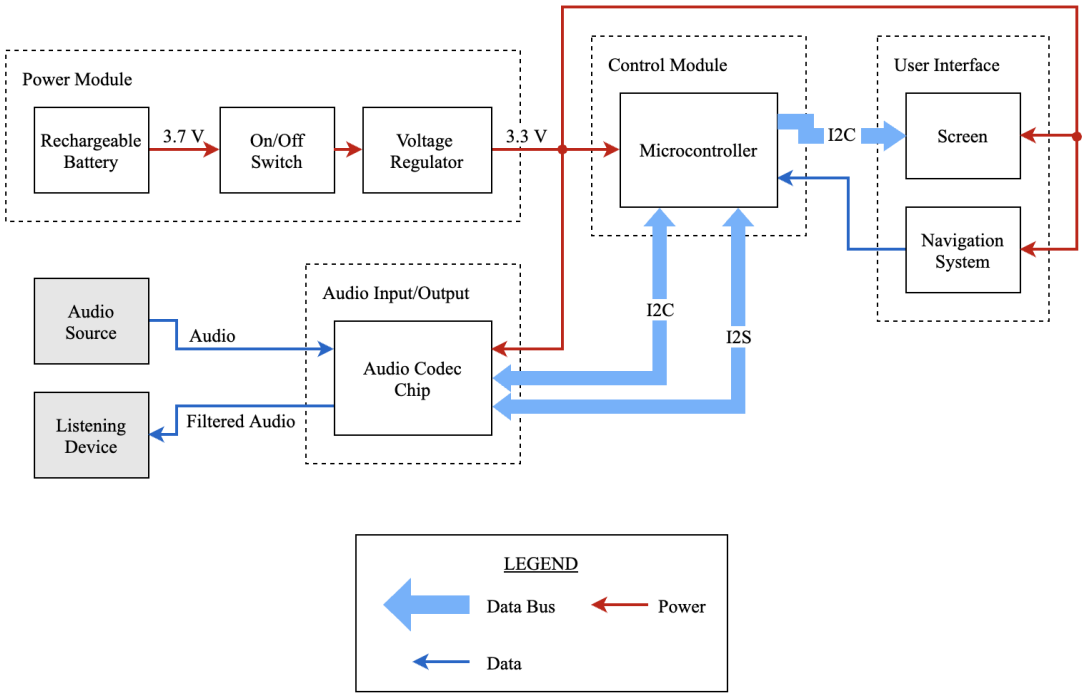


Figure 1: A Block Diagram of the PIAE

The PIAE design has power, control, user interface, and audio input/output as the primary units. The power module generates an adequate amount of voltage for the other modules to use. The audio input/output module formats audio data accordingly, allowing other components to understand the data. Using the data and desired filters that a user selects through the user interface, the control module generates filtered data. This filtered data returns to the audio input/output module which is then outputted.

2.2 Power Subsystem

2.2.1 Rechargeable Battery

The lithium-ion battery provides power for the device. Although it interfaces exclusively with the on/off switch, power is routed through the switch to the low-dropout (LDO) regulator and subsequently to the rest of the circuit. This specific battery model was chosen because it supplies a voltage of 3.7 V, which is relatively close to the circuit's operating voltage of 3.3 V. This allows the LDO regulator to be far more

efficient, according to Equation (1) [5].

$$\text{Efficiency} = \frac{I_{OUT}}{I_{OUT} + I_{GND}} \cdot \frac{V_{OUT}}{V_{IN}} \tag{1}$$

$$\text{Efficiency} \propto \frac{V_{OUT}}{V_{IN}}$$

$$\text{Efficiency} \propto \frac{3.3}{3.7}$$

$$\text{Efficiency} \propto 89.12\%$$

Therefore, the voltage of the battery ensures excellent efficiency in the LDO regulator. Furthermore, the battery has a capacity of 2500 mAh. When it is fully charged, the battery allows for a lifetime of at least three hours, which is shown in Equation (2).

$$\text{Battery Life} = \frac{\text{Battery Capacity (mAh)}}{\text{Load Current (mA)}} \tag{2}$$

$$\text{Battery Life} \approx \frac{2500 \text{ mAh}}{791.1 \text{ mA}}$$

$$\text{Battery Life} \approx 3 \text{ h}$$

The load current in Equation (2) is derived from Table 1 below. The lifetime is reasonable and procuring a battery with greater capacity may compromise the portability high level requirement due to a larger battery being needed. Additionally, improving efficiency when using the audio codec and the microcontroller unit (MCU) helps to increase the uptime of the PIAE.

Table 1: Max Current for Each Component

Name	Max Current (mA)
Audio Codec Chip	150
STM32 Microcontroller	600
LCD Screen	41.1
Total	791.1

2.2.2 On/Off Switch

The on/off switch is connected to the battery, which allows it to power the PIAE on and off. The output of the switch is connected to the voltage regulator.

2.2.3 Voltage Regulator

The voltage regulator ensures that the voltage supplied to the circuit is maintained at 3.3 V ±5% and has an output current of 800 mA ±5%. When the switch is on, the regulator takes the battery output of 3.7 V and converts it to a usable 3.3 V, which powers the rest of the device.

2.2.4 Design Procedure

The rechargeable battery is connected directly to the on/off switch to prevent the constant draining of the battery. This makes the PIAE more sustainable. The output of the switch is connected to the voltage regulator since the power must be adjusted for the other components to be able to use it.

2.2.5 Design Details

A Single Pole Double Throw (SPDT) switch is chosen, but used as a Single Pole Single Throw (SPST) switch by leaving one of the outputs unconnected and the other output connected to the input of the voltage regulator. Many available SPST switches are too small to comfortably be used by a user. As a result, the SPDT switch is used.

Considering the battery output of 3.7 V, the LDO regulator is necessary as it is effective with regulating voltages that are close to the desired 3.3 V output. As shown in Appendix B, the regulator has capacitors connected to the input and output in order fix the output voltage at 3.3 V.

2.3 Control Subsystem

2.3.1 Microcontroller

The microcontroller filters the I^2S audio data from the audio codec chip according to the currently selected EQ settings. For filtering, the microcontroller uses eight frequency bands in the digital signal processing of the audio data within a frequency range of 100 Hz to 20000 Hz. Additionally, the unit controls the screen display so that the currently selected EQ setting, as well as other possible EQ settings, are shown. Therefore, the microcontroller ensures that users can quickly and accurately change EQ settings to their preference.

The STM32 MCU provides many advantages as opposed to other microcontrollers. It has 1 MB of RAM and can compute complex fast Fourier transforms (FFT) quickly [6]. Also, it is highly accessible as it allows for users to program in C/C++, allowing for ease of use.

2.3.2 Design Procedure

At the broadest level, the control subsystem is designed to execute all relevant filtering in the frequency domain. Data arrives via I^2S from the audio codec chip's analog to digital converter (ADC) and is transformed into the frequency domain. Once it is filtered, the data then transforms back to the time domain and is sent through I^2S to the digital to analog converter (DAC) in the audio codec. This approach requires filters to be constructed in the frequency domain.

An alternative to this frequency-centric approach is to conduct all filtering in the time domain, but this would require convolutions between the audio data and time domain filters, which is often far slower than executing operations in the frequency domain [7].

Finally,the short-time Fourier transform (STFT) and its inverse (ISTFT) are used to transform the audio data, as well as the FFT to accomplish the former feat and to construct the filters [8], [9].

2.3.3 Design Details

In order to filter, we must first construct eight frequency band filters. These eight filters are drawn from three types - a bell filter, a high shelf filter, and a low shelf filter. The high shelf filter is only used for the highest frequency band, and similarly, the low shelf filter for the lowest frequency band. Bell filters are used for every other frequency band, and cannot be used for the highest or lowest frequency because their bell shape prevents them from properly boosting or attenuating frequencies at the very edge of the frequency range. The equation for each filter can be found in Reference [10] - these equations are omitted here because they are extensive and do not provide much in addition to the original source. Numerator and denominator coefficients are constructed as follows in Equation (3) for each filter.

$$\mathbf{A}, \mathbf{B} \in \mathbb{R}^N \text{ , where } N \text{ refers to the size of the fast fourier transform (FFT)} \tag{3}$$
$$\mathbf{A} = \begin{bmatrix} a0 & a1 & a2 & 0 & 0 & ... & 0 \end{bmatrix}$$
$$\mathbf{B} = \begin{bmatrix} b0 & b1 & b2 & 0 & 0 & ... & 0 \end{bmatrix}$$

$a0$, $a1$, $a2$ and $b0$, $b1$, $b2$ are all coefficients described in Reference [10]. In order to produce each filter, we take the Fourier transform of these vectors ($\mathbf{A}$ and $\mathbf{B}$) and divide FFT($\mathbf{B}$) by FFT($\mathbf{A}$) ($\frac{FFT(\mathbf{B})}{FFT(\mathbf{A})}$). Once the MCU powers up, we can immediately construct all 8 filters and store them in a data structure, such as an array, for later use.

To build the filters, we use S factors and bandwidths of [1.0, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 1.0]. Each number in this list corresponds to the frequency of the same index in the list of frequencies in the high-level requirements, e.g. 1.0 to 100 Hz in the first index. S factors are used for the high shelf and low shelf filters, whereas bandwidths are used for the bell filters. Bandwidths were intended to represent the number of octaves between midpoint gain frequencies, but experimentally, were shown to be inaccurate for this purpose. Instead, bandwidths provide a vague sense of how wide each bell filter would be. Similarly, S factors determine how 'steep' the shelf filters are. We did not meticulously choose bandwidth and S factor values because minor adjustments to them produced no real effect on equalization. Therefore, we chose them to balance between preventing 'valleys' between center frequency gains and attenuations and reducing inaccurate gains and attenuations produced as a result of the filters interfering with each other.

Filtering the data itself is a more involved process. In order to execute the STFT, we must split the data into overlapping frames, separately window the resulting segments, and take the FFT of these windowed segments [8]. Similarly, to execute the ISTFT, we reverse the process. We must take the inverse FFT of each segment, undo the windowing, and then overlap each segment appropriately and add them. Given the STFT of the audio data, we merely have to multiply each frame with the product of the constructed filterbank across all frequency bands. The entire algorithm, including the pipelining of audio data, is depicted below

in Algorithm 1.

Algorithm 1: Psuedo-algorithm of the Filtering Process

N = fft_size;
$hop = \frac{N}{2}$ (STFT frame overlap);
$A = n \cdot N$ (audio buffer length);
$n\_frames = \lfloor \frac{(A-N)}{hop} \rfloor + 1$;
F = initialize_filterbank($gains$, $center\_frequencies$, $bandwidths$, num_bands=8);
$slice\_len$ = A-N;
win_a = Hann_window(N);
$input\_signal = \mathbf{0}^{1xA}$;
while *True* **do**
  $input\_signal[N \rightarrow A]$ = Receive_from_I^{2}S($slice\_len$);
  for n = *0* $\rightarrow$ n_frames-*1* **do**
    $X[n]$ = FFT($input\_signal[n \cdot hop \rightarrow n \cdot hop + N] \cdot win\_a$);
  end
  $X = X \cdot F$;
  $filtered\_signal = \mathbf{0}^{1xA}$;
  $norm = \mathbf{0}^{1xA}$;
  for n = *0* $\rightarrow$ n_frames-*1* **do**
    $filtered\_signal[n \cdot hop \rightarrow n \cdot hop + N]$ += IFFT($X[n]$)$\cdot win\_a$;
    $norm[n \cdot hop \rightarrow n \cdot hop + N]$ += win_a ^{2}
  end
  $filtered\_signal$ /= $norm$;
  Send_to_I^{2}S($filtered\_signal[hop \rightarrow hop{+}slice\_len]$);
  $input\_signal[0 \rightarrow N] = input\_signal[slice\_len \rightarrow A]$;
end

There are a couple of things to note. Firstly, the Hann window can be calculated as follows in Equation (4), which is derived from Reference [11].

$$w(n) = 0.5(1 - cos(2\pi \frac{n}{N-1})) \; 0 \leq n < N \tag{4}$$

Secondly, variable n, which is used to calculate A, can be any integer $n \geq 2$.

For the demo, our Fourier transform length was 512, overlap between frames for the STFT was 256, sampling rate was 44000 Hz, and total buffer size for the audio input was 1024 (n=2). The total buffer size is the minimum size needed for the STFT (Fourier transform length $\cdot$ 2).

2.4 User Interface Subsystem

2.4.1 Design Procedure

The user interface subsystem allows users to select their desired EQ settings. The PIAE gives users the options of adjusting the eight frequency bands. After selecting the frequency band option, the system gives the option to set the decibel level according to the following values : [-3, -2, -1 , 0 , +1 , +2, +3]. These levels are selected for convenience, and it is a simple change to allow up to a magnitude of 10 dB as described in the Abstract. These options provide users a wide range of options to customize their audio according to their preferences. An example is users who like their bass boosted. They could boost the 250 to 500 Hz frequency range by 10 dB and that would increase the bass in their audio.

In order to fit the portability requirement, we use a 20 cm x 4 cm LCD screen, as well as push buttons for the selection of a setting and for toggling between options.

2.4.2 Design Details

The LCD screen relies on I^2C protocols for communication. In order to initialize the device, a buffer of 64 bytes is allocated on the microcontroller's memory module at an address specified by the programmer. Then, a signal is sent via inbuilt MCU functions to the screen for it to enter write mode. The write mode indicates to the screen that it should scan the memory location specified by the programmer and write the buffer to screen display. To write to the screen, we create a character buffer with our desired message that we want to display and place it at the memory address that was specified earlier. In particular, we must be careful about overwriting the allocated 64 bytes on the stack as it could interfere with the microcontroller's inbuilt functions. To prevent this issue, we place a memory fence around the allocated memory and cause the program to throw an error if the provided character buffer went out of bounds of our allocated memory.

The navigation system allows for users to cycle between options and select their desired frequency and decibel levels. The user interacts with the MCU through two push buttons, one for cycling between the EQ options and another to select the option. When pushed, the push buttons send 3.3 V to a specified pin in the microcontroller. The pin must be programmed to be interpreted as an external interrupt, which allows it to read the 3.3 V as a signal from the user. In software, an interrupt handler is written to deal with the different button presses. For each of the buttons, a global flag is set according to the button pressed, and we can take action based on the flag raised in software.

To put these two parts together, we used the following workflow for the PIAE. First, we enter an initialization sequence and allocate the memory on the heap. If no error occurs, we move on to the user settings selections. The user first selects the frequency option. We enter an infinite loop that is only broken if the global flag for the selection option has been raised. We then display the available frequencies band options, with the default option at 100 Hz. Within this loop, the options can be cycled through until the user lands on their desired option. Once the user hits the select button, the loop breaks and their frequency setting is saved. Next, the user selects the decibel level. We enter a similar loop and can cycle between the decibel options, with the default option being 0 dB. When the user hits the select button, we once again break out of the loop and save the selected decibel value. Then, the user is given the option of re-selecting their settings or continuing on with their current selection. If the user chooses to re-select, we enter the sequence after the initialization again. However, if the user continues on, we reconstruct the filters with the new gains and begin to synthesize audio according to the user preferences.

2.5 Audio Input/Output Subsystem

2.5.1 Design Procedure

The audio input/output subsystem is responsible for external connections, routing input audio from the audio source to the MCU, and routing filtered audio from the MCU to the listening device. As such, the subsystem comprises of the CS4245 audio codec chip, 3.5 mm audio jack connectors, and all associated passive components needed for the audio codec chip's operation. Devices connect to the PIAE through the audio jack connectors via audio jacks. Meanwhile, the ADC and DAC of the audio codec are connected to the MCU through I^2S, whereas the codec itself is connected to the MCU through I^2C.

2.5.2 Design Details

As shown in circuit schematic of Appendix E, the CS4245 chip is located in the middle, whereas the two 3.5 mm connectors can be found at the bottom left and top right.

We changed our audio codec chip from the TLV320AIC3204, shown in our design proposal, to the CS4245. This is because the former chip was in a leadless package, which makes it difficult and riskier to solder. Without sacrificing much performance, we chose to use the CS4245 instead. The CS4245 has almost identical features, except it does not feature PowerTune technology which assists in decreasing power consumption [12], [13].

We chose the CS4245 for many reasons. Firstly, the chip can run on 3.3 V, which is the output of the LDO regulator in our circuit [12]. Secondly, the chip can sample stereo (2 channels, as with headphones) data up to 192 kHz, which is well above the Nyquist rate of 40000 Hz required where the Nyquist rate is the minimum rate needed to sample a signal while losing no information [14]. The 40000 Hz is derived from the largest frequency we equalize, 20000 Hz, and Equation (5).

$$\text{Nyquist Rate} = \text{Bandwidth} \cdot 2 \tag{5}$$

The needed bandwidth is 20000 Hz, and the resulting Nyquist rate is 40000 Hz.

3 Design Verification

3.1 Power Subsystem

A verification for the power module is the battery lifetime. The battery lifetime must be at least 3 hours. The fully charged battery was connected to resistors with the equivalent resistance of the circuit. By checking the battery's output, the voltage showed to stay above 3.3 V for at least 3 hours, as shown in Figure 2.

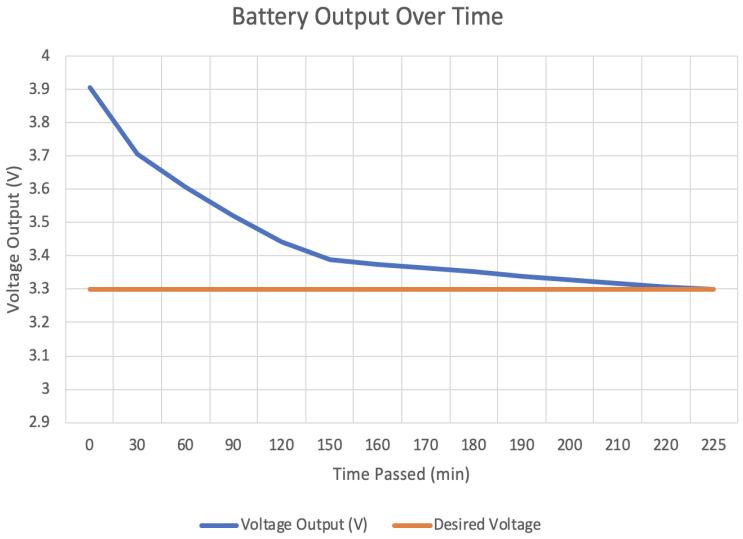


Figure 2: Lifetime of the Rechargeable Battery

To verify that the voltage regulator properly converts the battery output to 3.3 V ±5%, various voltage values were inputted into the regulator and the output voltages were read. All of the outputs were within the ±5% tolerance from 3.3 V, as shown in Table 2, so the verification was successful.

Table 2: Voltage Regulator Output with Varying Input Voltages

Voltage Regulator Input	Voltage Regulator Output
5.0 V	3.29 V
4.0 V	3.22 V
3.3 V	3.18 V

3.2 Control Subsystem

Refer to the R&V table in Appendix A.6 for the control subsystem requirements and verifications. No requirement was verified and that is primarily because we were unable to get the audio codec chip to work, as described in Section 3.4. As a result, we could neither receive data from the audio codec - invalidating the first requirement - nor send data to it - invalidating the second. Nonetheless, we were able to demonstrate during the demo that the MCU is capable of holding more than 4000 bytes and of filtering a toy signal correctly. Therefore, we verified that the MCU works to specification, and had the audio codec chip been working, it is likely that we would have succeeded in verifying our requirements.

In Figure 3 and Figure 4, we show that the on-board MCU produced the same results as the C code running on a laptop. The toy signal that was filtered was a linear chirp that runs from a frequency of 0 Hz to 22000 Hz in 5 seconds. The correctness of the filtering was verified by executing the same algorithm using Python libraries on a laptop computer, and the details of this process are omitted for the sake of brevity.

```
C-code run on computer values
Value of Filtered Signal at index 300 round 1: 1.07771
Value of Filtered Signal at index 482 round 2: -40.88402
Value of Filtered Signal at index 678 round 3: 27.94294
Value of Filtered Signal at index 711 round 4: -20.45793
```

Figure 3: Results of C code run on a laptop

```
print final_signal_right[300]
$2 = 1.07738686
p final_signal_right[482]
$3 = -40.8815956
p final_signal_right[678]
$4 = 27.9496994
p final_signal_right[711]
$7 = -20.4506683
```

Figure 4: Results of C code run on MCU

Each round refers to a multiple of output length 512 (A - N = 1024 - 512). Therefore, index 482 round 2 actually refers to index 512 + 482 - 256 = 738. The first and last 256 indices of the buffer in each round are filled with half-transformed data due to the nature of the STFT and ISTFT, and are thus ignored.

Finally, we must address the latency of the filtering itself. Although this was not codified in the R&V table, it is an important aspect of the control subsystem. Given the parameters during the demo, our main MCU program loop has to be faster than 11 ms, as shown below in Equation (6).

$$\frac{1}{44000 \text{ samples/second}} \cdot 512 \text{ samples} \cdot 1000 \text{ milliseconds/second} = 11.6 \text{ milliseconds} \tag{6}$$

During the demo, we showed that our filtering itself ran in 19 ms, thus being far too slow for any other necessary operations to be executed, let alone to meet the requirement of 11 ms. However, if we set $n = 6$ (refer to Algorithm 1), our filtering only requires 49 ms as opposed to a total loop time of 58 ms. This is shown in Equation (7).

$$\frac{1}{44000 \text{ samples/second}} \cdot 2560 \text{ samples} \cdot 1000 \text{ milliseconds/second} = 58.2 \text{ milliseconds} \tag{7}$$

This means that all other operations would have to be less than 9 ms, which may or may not be unreasonable depending on how long it takes to receive data from the audio codec chip. Nevertheless, that would mean that the filtering does meet the time constraints, which speaks to the control subsystem's success.

3.3 User Interface Subsystem

To verify that the screen subsystem worked, we wrote a character buffer of size 64 bytes to the screen. If the screen display shows the character buffer, we have been successful. The LCD screen is shown in Figure 5 with each of our names and it shows our ability to successfully write to the screen.

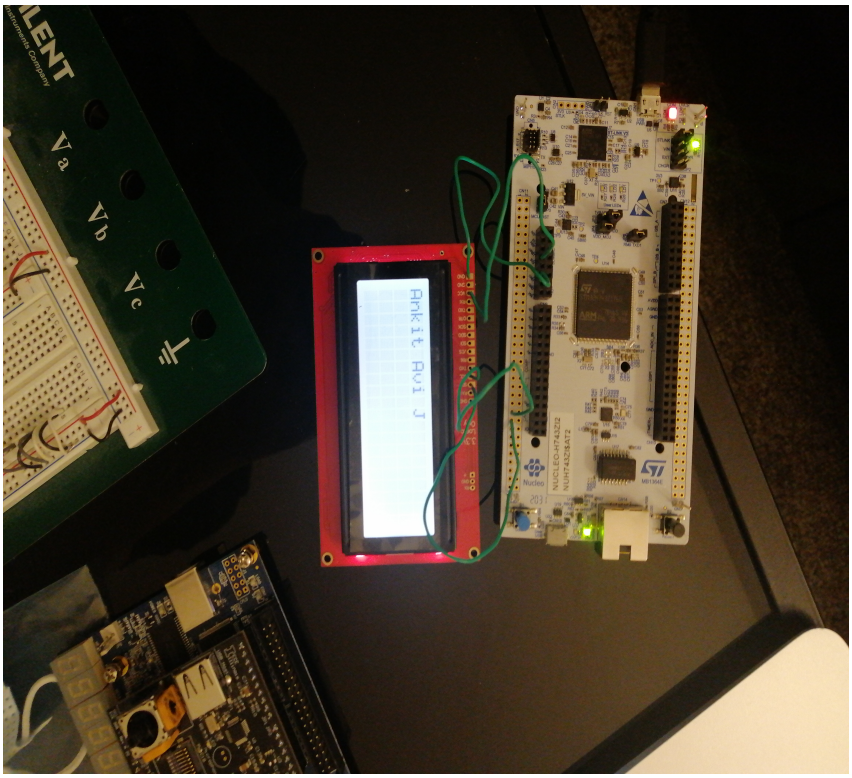


Figure 5: PCB A Board Design

Next, we verified that the buttons and interrupts worked. In software, we first wrote an infinite loop that can only be broken by an interrupt, and then a print statement to write after that infinite loop. In theory, the statement is unreachable unless we are able to break out of the infinite loop. Since our interrupt was successful, we were able to break out of the loop and reach that print statement.

3.4 Audio Input/Output Subsystem

Unfortunately, we were not able to verify any of the requirements for the audio input/output subsystem. These requirements, along with their corresponding verification procedures, are located in Appendix A.8. Verifications failed because we inadvertently shorted the audio codec chip during breadboarding, most probably due to a flaw in the circuit design. As a result, the chip was rendered unusable, and we were unable to test the functionality of the entire subsystem. If our product is to ever be viable for commercial release, this is the first issue with our project we must remedy.

3.5 Portability

One of the high level requirements is for the PIAE to be portable. This is accomplished through having a dimensions upper limit of 16 by 12 by 6 cm. Physical designs were created, as shown in the figures in

Appendix F. Figures 5 and 6 show the board designs for the PCBs that are in the physical designs. The box used for the PIAE was measured to be within the limitations.

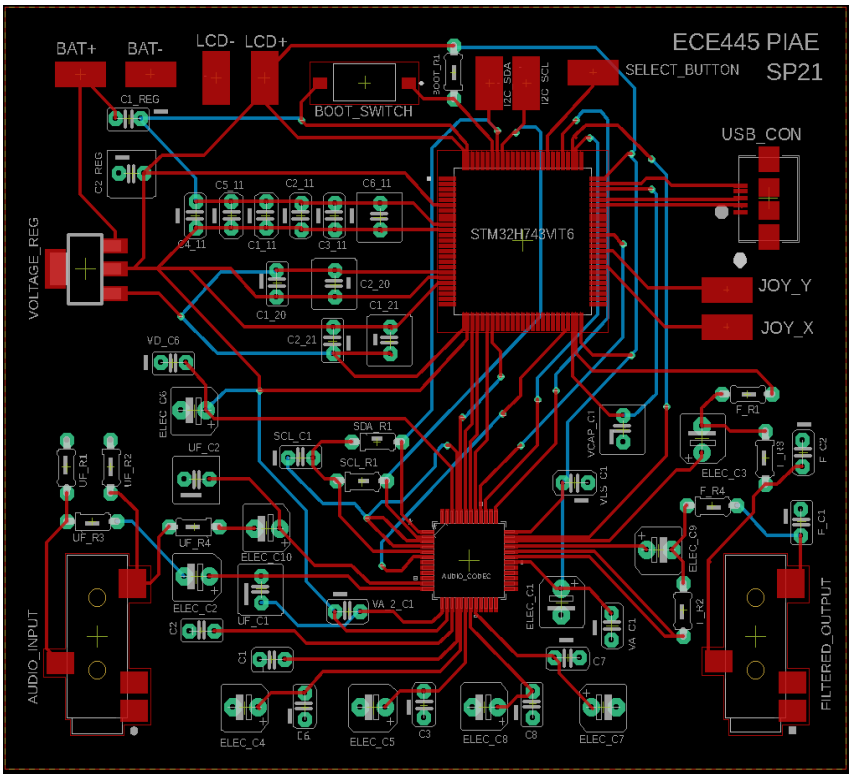


Figure 6: PCB A Board Design

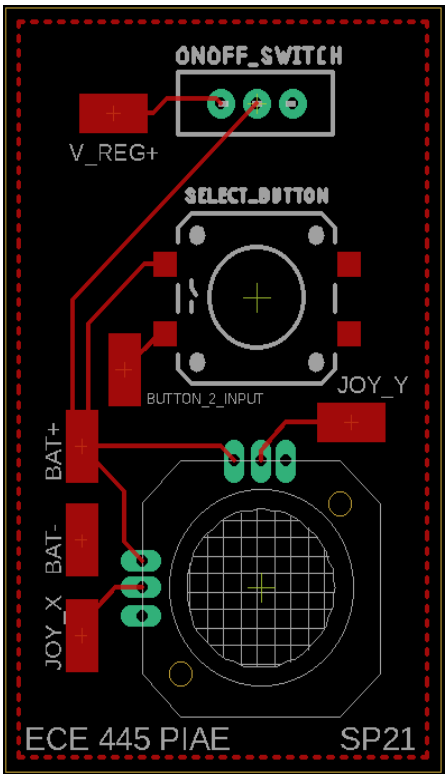


Figure 7: PCB B Board Design

4 Cost

4.1 Parts

All parts used for the design of the PIAE are listed in Table 3 along with their associated unit prices.

Table 3: Cost of Parts

Name	Manufacturer	Part Number	Quantity	Unit Price ($)
USB LiIon/LiPoly Charger	Adafruit	259	1	12.50
USB 2.0 Cable A-Male to Mini-B 3 Feet	AmazonBasics	B00NH13S44	1	2.00
3.5 mm Jack Connector SMD	CUI Devices	SJ-43514-SMT-TR	2	1.05
Audio Codec	Cirrus Logic	CS4245	1	6.43
Male to Male 3.5 mm Cable 4 Conductor	YCS	4330104966	1	5.97
Microcontroller	STMicroelectronics	STM32H743VIT6	1	12.29
DIP Switch	CUI Devices	DS04-254-2-01BK-SMT	1	0.70
Voltage Regulator 3.3V	STMicroelectronics	LD1117V33	1	0.55
Li-Ion Batteries	Adafruit	328	1	14.95
Mini 2-Axis Analog Thumbstick	Adafruit	2765	1	2.50
Tactile Button SMD (12 mm)	SparkFun Electronics	COM-12993	1	0.60
Mini Power Switch SPDT	SparkFun Electronics	COM-00102	1	1.50
SerLCD 20x4	SparkFun Electronics	LCD-16398	1	25.00
Micro USB Type B Connectors	Amphenol FCI	10104110-0001LF	1	0.78
Miscellaneous Components (resistors, capacitors, etc.)	—	—	—	7.50
Total	—	—	—	94.32

4.2 Labor

We estimated that the average salary of a graduate in Electrical Engineering is $35 per hour. Furthermore, the three group members spent an average of ten hours a week on the project. Therefore:

$$\text{Total Labor Cost} = 3 \text{ people} * \$35/\text{hour} * 10 \text{ hours/week} * 9 \text{ weeks} * 2.5 = \$23625$$

5 Conclusion

5.1 Accomplishments

Two of the high-level requirements were met; we were able to use the eight enumerated frequency bands for the PIAE's filters, and we were able to meet the dimensional requirements. Furthermore, the filtering of the data is correctly executed by the current software, the power system is able to supply adequate voltage for three hours to the device, and the user interface subsystem works as intended. With perhaps a few minor touch-ups, customers will be able to easily and intuitively select a desired frequency band and increase its corresponding gain. Finally, our device was successfully housed in an enclosure that reduces the chance of component damage due to occurrences like weather hazards.

5.2 Uncertainties

Unfortunately, we were not able to verify many of our requirements. First and foremost, we were unable to get our audio codec chip working, which means that we do not know whether our design of the Audio Input/Output Subystem is successful. This is a critical component of our project, as it is a prerequisite to any filtering itself. The audio codec chip's failure in turn meant that we were unable to verify any of the requirements of the control unit, and although we are certain the data pipelining works with toy data, we need to verify whether it works with real audio data. We were also unable to meet the first high-level requirement as well, since we were unable to document the latency of the audio codec chip's operations as well as the latency of reading from the chip through I^2S, which factor into the 100 ms in the requirement.

Secondly, we are unsure of whether our filtering in software is fast enough to meet the time constraint imposed on the entire software loop by the high sampling frequency. Although the filtering algorithm can meet this constraint with certain parameters, we have not been able to verify that other operations in tandem, such as reading in data through I^2S, are fast enough to satisfy the constraint.

5.3 Ethical considerations

Our ethical considerations extend firstly to issues with volume control. Audio at extremely loud volumes damages human hearing over time [15]. The IEEE Code of Ethics requires us "to hold paramount the... health and welfare of the public" [16], and therefore, the PIAE should not damage our user base's hearing without their knowledge. To this effect, we had planned to limit the volume of the PIAE's output audio, to clip the volume at 100 dB and to warn users that listening to sound louder than 75 dB could damage their hearing [15]. However, we did not have enough time to implement these features. Therefore, we would either need to halt the production of our product until both the aforementioned uncertainties are resolved and these safety features are implemented, or we would need to release an instruction manual or warning to users with the product. The manual or warning would inform users that boosting certain frequency ranges in their music may increase the volume to unsafe levels.

We are also concerned with the power unit. Lithium batteries, which are used for the power unit, may produce fire or explode when they are used incorrectly or damaged [17]. To mitigate this, the batteries we

are using come with protection circuitry that prevent over-charging, under-charging, and output shorts [18]. Nevertheless, we could also include such information in any manual or warning issued to users - specifically asking them not to allow the device to come into strong collisions, such as those caused by throwing or dropping the device. Finally, the housing compartment for the device affords some protection, as any explosion would first have to penetrate the compartment before damaging anything else.

5.4 Future work

Aside from fixing the problems with our device, we also envisioned improvements to the speed of the filtering, the user interface subsystem, and the portability of the device. Furthermore, we could increase the number of frequency bands from eight to ten. This increase is relatively simple to implement in software, and would allow users to more finely tune their music to their preferences or needs.

We can improve the speed of the filtering in one of two ways. Either we can optimize the code that we currently have, or we can move the filtering to a Digital Signal Processor or to hardware, like to a Field Programmable Gate Array. The former suggestion is unlikely to work, because we used much of the Arm Digital Signal Processing functionality for the filtering code and little original code of our own, which indicates that the MCU itself might be too slow. Therefore, the latter solution is more promising.

Although the user interface subsystem is adequate, we could further improve it by including a more advanced screen. Currently, we are using 20x4 LCD screen, whereas we could be using a screen with a much higher resolution such as 128x64. This would allow for much more detailed visuals, for instance displaying each frequency band as a graphic rather than in text. In turn, users would be able to more quickly understand what gain each filter currently implements, rather than being forced to scroll through every frequency band to do so.

Finally, despite meeting the high-level requirements for dimensionality, the device is slightly uncomfortable to put into one's pocket. Specifically, the enclosure is quite tall, which means it is likely to press on the top of a pocket, and protrude visibly. Nevertheless, the product fits inside the average pocket, making the size a good base to work with [19]. Therefore, in order to increase commercial success, we would need to investigate some method of shrinking the height of the product to fit the average pocket.

References

[1] "Vital Signs: Noise-Induced Hearing Loss Among Adults — United States 2011–2012". CDC, https://www.cdc.gov/mmwr/volumes/66/wr/mm6605e3.htm. Accessed 14 Feb. 2021.

[2] Large commercial equalizer. https://www.cheapham.com/8-band-eq-by-w2ihy-special/. Accessed 14 Feb. 2021.

[3] Small Portable equalizer.https://www.amazon.com/Syba-SD-DAC63106-Control-Headphone-Amplifier/dp/B00TYYMHQS. Accessed 14 Feb. 2021.

[4] "How Latency Affects User Engagement." Pusher, 18 Feb. 2020, blog.pusher.com/how-latency-affects-user-engagement/.

[5] "Understand Low-Dropout Regulator (LDO) Concepts to Achieve Optimal Designs." Analog Devices, www.analog.com/en/analog-dialogue/articles/understand-ldo-concepts.html.

[6] "STM32H743VI - STMicroelectronics." https://www.st.com/en/microcontrollers-microprocessors/stm32h743vi.html (accessed May 05, 2021).

[7] "FFT Convolution vs. Direct Convolution." https://ccrma.stanford.edu/ jos/ReviewFourier/FFT_Convolution_vs_Direc (accessed May 05, 2021).

[8] "Short-time Fourier transform," Wikipedia. Dec. 28, 2020, Accessed: Apr. 05, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Short-time_Fourier_transform&oldid=996725594.

[9] Fast Fourier Transform (FFT). http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html. Accessed 1 Mar. 2021.

[10] Ryan Marcus, "Cookbook formulae for audio EQ biquad filter coefficients," Github Gist. https://gist.github.com/RyanMarcus/d3386baa6b4cb1ac47f4 (accessed Apr. 05, 2021).

[11] "Hann (Hanning) window - MATLAB hann." https://www.mathworks.com/help/signal/ref/hann.html (accessed Apr. 05, 2021).

[12] "CS4245 — Cirrus Logic." https://www.cirrus.com/products/cs4245/ (accessed Apr. 05, 2021).

[13] "TLV320AIC3204 Very-Low-Power Stereo Audio CODEC With PowerTune™ Technology." Texas Instruments, https://www.ti.com/product/TLV320AIC3204?qgpn=tlv320aic3204. Accessed 5 Mar. 2021.

[14] "Nyquist Rate." COSMOS, https://astronomy.swin.edu.au/cosmos/n/Nyquist+Rate. Accessed 5 Mar. 2021.

[15] "Hearing Loss - Symptoms and Causes." Mayo Clinic, https://www.mayoclinic.org/diseases-conditions/hearing-loss/symptoms-causes/syc-20373072. Accessed 14 Feb. 2021.

[16] IEEE Code of Ethics. https://www.ieee.org/about/corporate/governance/p7-8.html. Accessed 14 Feb. 2021.

[17] "Preventing Fire and/or Explosion Injury from Small and Wearable Lithium Battery Powered Devices." Occupational Safety and Health Administration, 20 June 2019, www.osha.gov/dts/shib/shib011819.html.

[18] A. Industries, "Lithium Ion Polymer Battery - 3.7v 2500mAh." https://www.adafruit.com/product/328 (accessed May 05, 2021).

[19] "Women's Pockets are Inferior.," The Pudding. https://pudding.cool/2018/08/pockets/ (accessed May 05, 2021).

Appendix A Requirement and Verification Table

Table 4: Requirements and Verifications of the Rechargeable Battery

Requirement	Verification	Verification status (Y or N)
1. The battery must supply +3.7 V +15%/-5% power.	1. To check that the battery has an output voltage in the acceptable range, we will do the following: (a) Ensure that the battery is fully charged, reading +4.2 V using a DMM across JST pins. (b) Let the battery power the circuit for 30 minutes. (c) Measure the voltage across the JST pins using an oscilloscope to verify that the voltage is +3.7 V +15%/-5%.	Y
2. The lifetime of the battery is at least 3 hours ±5%	2. If measurement is not possible at a given time, simply turn off the PIAE and turn it back on when measurement becomes possible again. To measure how long the device lasts when it is on, we will do the following: (a) Ensure that the battery is fully charged, reading anywhere from +4.2 V to +3.7 V using a DMM across JST pins. (b) Let the battery power the circuit and begin a timer. (c) Measure the voltage across the JST pins of the battery every 30 minutes using a DMM, reducing the interval to 10 minutes after 2 hours and 30 minutes. (d) When the voltage of the battery is less than +3.3 V, record the time elapsed, and verify it is at least 3 hours ±5%.	Y

Table 5: Requirements and Verifications of the Voltage Regulator

Requirement	Verification	Verification status (Y or N)
1. The voltage regulator should regulate the output voltage of the batteries to 3.3 V ±5% and maintain an output current of 700 mA ±5%.	1. Use a 3.3 V voltage regulator IC chip is to ensure that regardless of the battery output voltage, 3.3 V will be supplied to the STM32 microcontroller so it can operate safely. We will also check that 800 mA will be outputted from the regulator. We will verify the chip as follows : (a) Connect a variable voltage source in series with our IC chip input pin. (b) Connect the ground pin to the appropriate ground in the circuit. (c) Connect a resistor in series with the output of our IC chip. (d) Connect probes across the resistor to check the voltage drop across the resistor. (e) Start the variable voltage source at 3.3 V. Increase the voltage and check if the voltage across the resistor. If it is 3.3 V ±5% consistently, we have been successful. (f) Use a DMM to measure the current that the regulator is outputting in the PCB, and verify that it is 800 mA ±5%. If our current is within this range, we are successful.	Y

Table 6: Requirements and Verifications of the Control Subsystem

Requirement	Verification	Verification status (Y or N)
1. The microcontroller must be able to receive and store audio data of size 4000 bytes incoming from the audio codec chip.	1. To check if the microcontroller receives audio data from the audio codec chip, we will do the following: (a) Load the audio codec driver into the microcontroller flash memory. (b) The power module supplies an appropriate amount of power so the microcontroller is operational. (c) Plug the audio source into the line-in audio jack and start sending the data. (d) Verify that the data is accessible in memory. stored in the appropriate address specified by the audio codec driver.	N
2. The microcontroller must be able to output modified audio data through the audio codec.	2. To check the microcontroller is able to modify and output audio data, we will do the following: (a) Power on each device and transmit audio data to the microcontroller as specified by Process 1. (b) Access audio data stored in memory address specified by audio codec driver. (c) For each frequency band, shift the decibel value by precisely +5 dB using the navigation system. (d) Plot the FFT of both the original signal and the modified signal and compare. If the plots match a +5 dB shift, we have modified the signal correctly. (e) Transmit the data through the headphone jack in the PCB to a speaker to verify that the audio is being output through the microcontroller.	N

Table 7: Requirements and Verifications of the User Interface Subystem

Requirement	Verification	Verification status (Y or N)
1. The screen must display a maximum of 80 characters.	1. Check the LCD screen can display 80 characters in the following manner : (a) Connect the microcontroller and the LCD screen (b) Create and run a script in the microcontroller that makes the screen display 80 characters. (c) Create and run a script in the microcontroller that makes the screen display 81 characters. (d) If the screen successfully displays 80 characters with the first script, but unsuccessfully displays 81 characters with the second script, then the verification is a success.	Y

Continued on next page

Table 7 – continued from previous page

Requirement	Verification	Verification status (Y or N)
2. The thumbstick must change the screen display, depending on what direction it is pushed, and the current screen display must be selected by the push button.	2. Check that the thumbstick changes the display in the following way: (a) Connect the microcontroller, the thumbstick, the push button, and the LCD screen appropriately (b) Create a script in the microcontroller which can create five different displays depending on the thumbstick input. Each display must have two unique characters, one being a number for the x-direction and the other being a letter for the y-direction. The first display will be the starting display. The other four display options must correspond to the four directions of the thumbstick. The current display should show the two unique characters of all currently selected displays, if any. (c) Push the joystick in each corresponding direction. (d) Select each display with the push button (e) The verification is successful if all combinations of settings are shown to be selected and unselected.	Y

Table 8: Requirements and Verifications of the Audio Input/Output Subsystem

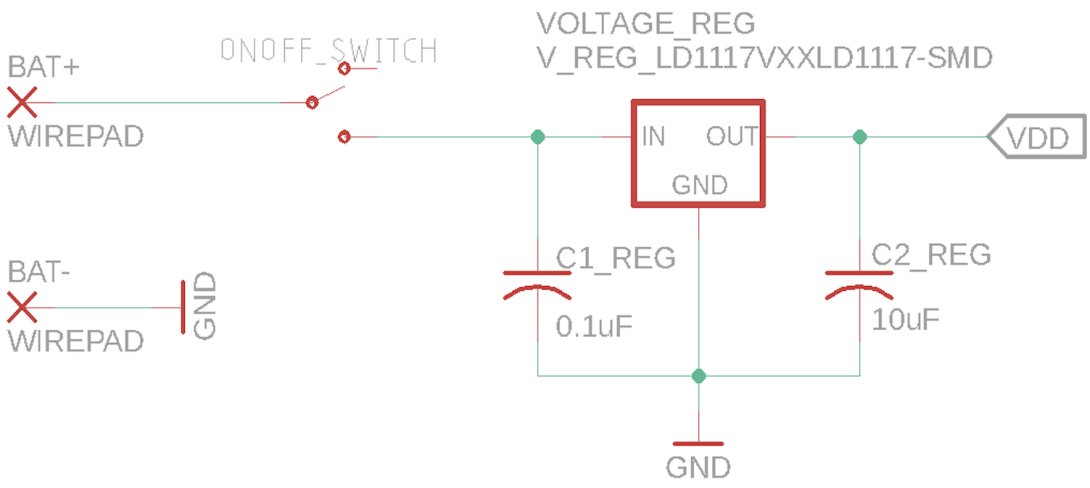
Requirement	Verification	Verification status (Y or N)
1. Requirement The audio codec must have a total system latency of less than 10 ms.	1. Check that the total system latency of the audio codec is less than 10 ms in the following manner: (a) Create a Python script using a laptop that allows timestamping when playing sound through the headphones port and recording sound from an in-built microphone. (b) Subtract the timestamp of incoming audio chunks from the corresponding outgoing chunks on the microcontroller to determine the average latency of filtering over 100 chunks. (c) Use the script to play a sound to the PIAE, with the external listening device being some sort of speaker such as headphones that are placed near the in-built microphone. (d) Correlate the sound that was played with the recording of the microphone and find the time delay of the microphone recording. (e) Subtract the microcontroller filtering latency from this time delay, and check that the resulting number is less than 10 ms.	N

Continued on next page

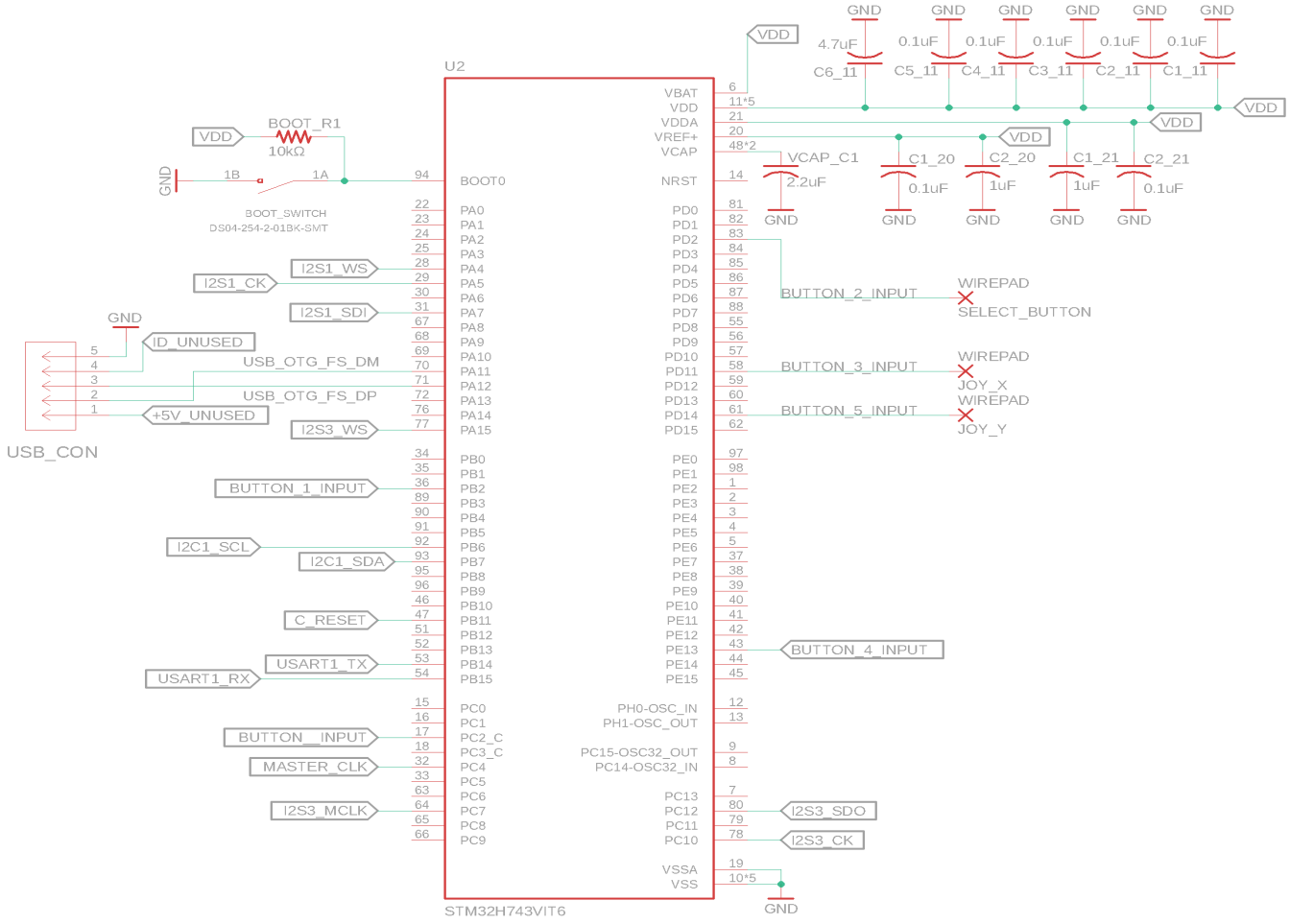
Table 8 – continued from previous page

Requirement	Verification	Verification status (Y or N)
2. The audio codec must be able to sample the audio data at a rate of at least 40,000 Hz.	2. Check that the audio codec sends at least 40,000 samples a second to the microcontroller in the following manner: (a) Create a program in the MCU that timestamps data incoming from the audio codec. (b) Connect the MCU and the audio codec, and connect some form of audio output to the audio codec. (c) Increment a counter for each sample received by the MCU. (d) Once the number of samples reaches 40,000, subtract the timestamp corresponding to the first of the 40,000 from the last and check that it's less than one second. (e) This experiment should be repeated 10 times, and if all 10 experiments are successful, then verification is complete.	N

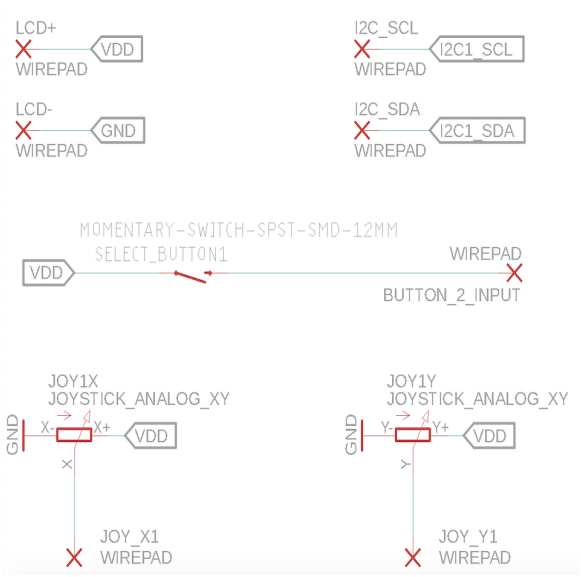
Appendix B Power Subsystem Circuit Diagram



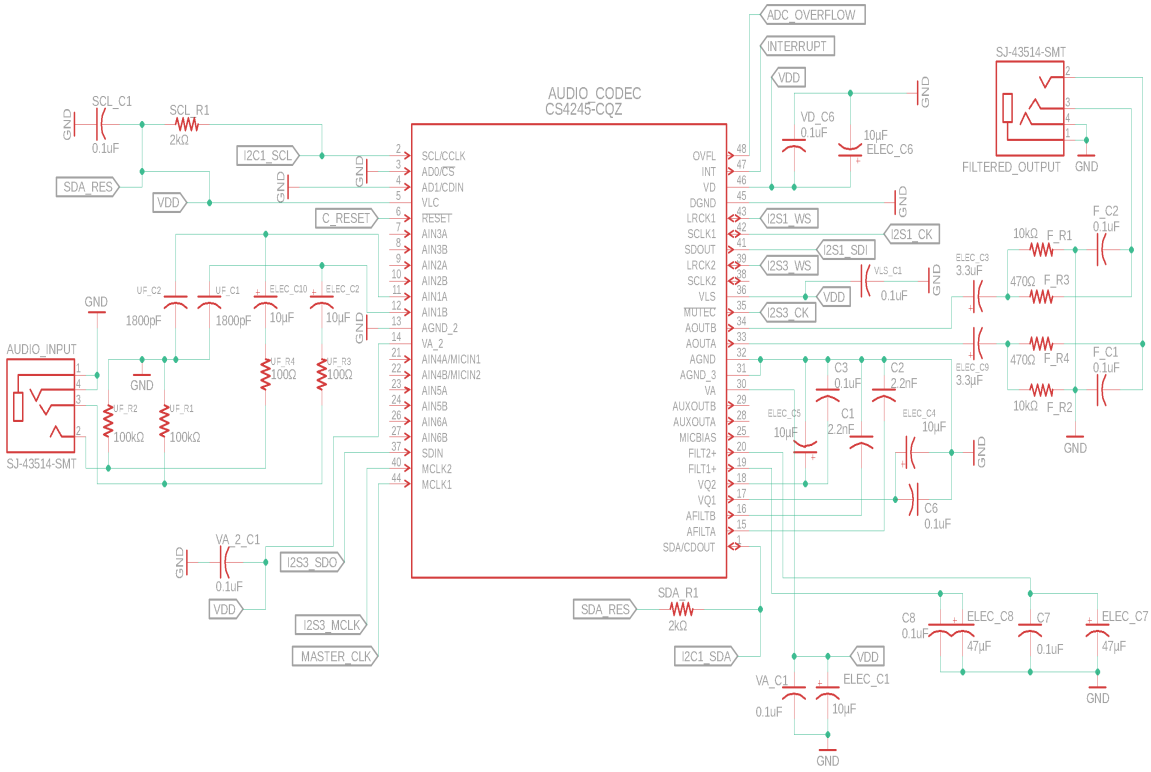
Appendix C Control Subsystem Circuit Diagram



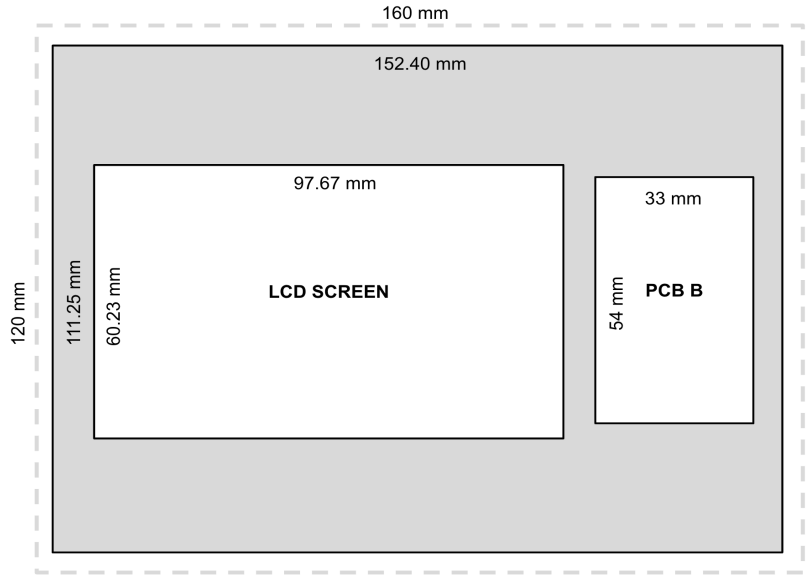
Appendix D User Interface Subsystem Circuit Diagram

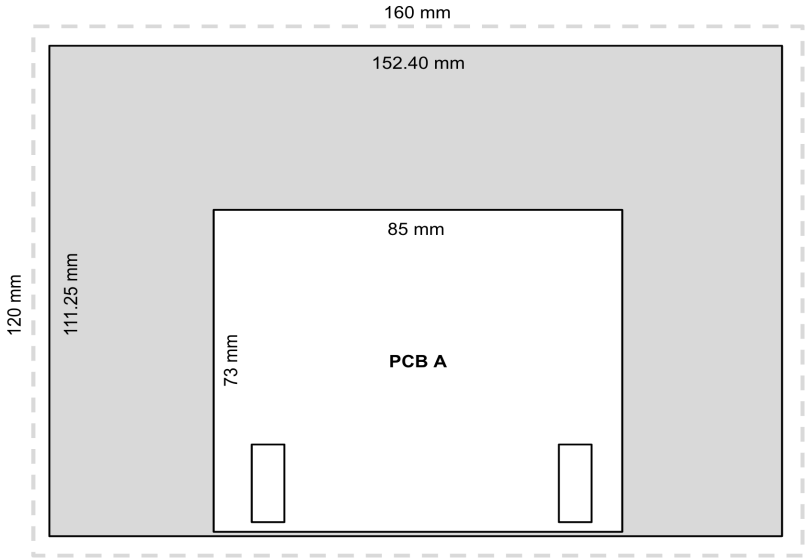


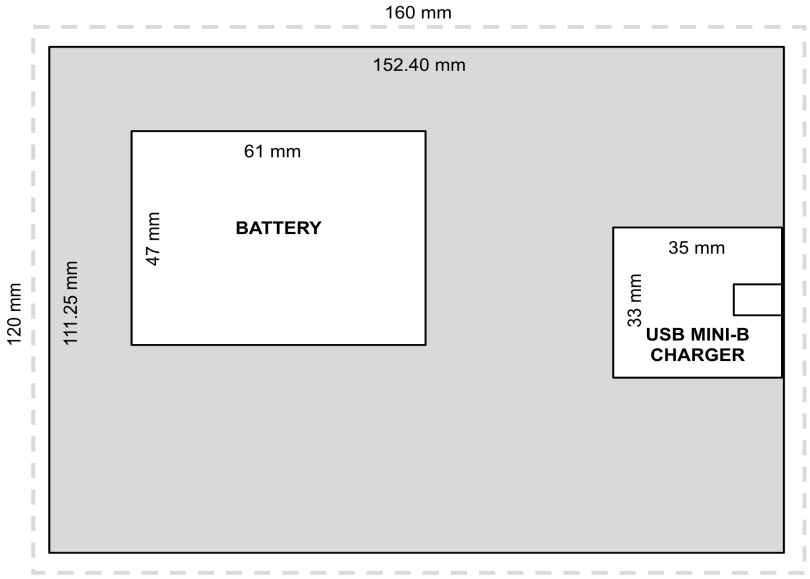
Appendix E Audio Input/Output Subsystem Circuit Diagram

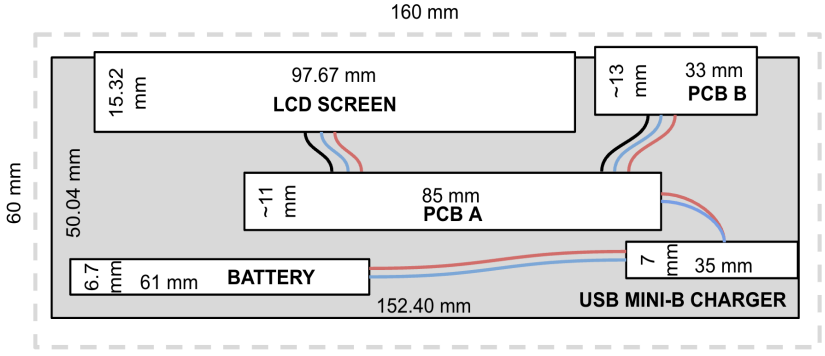


Appendix F Physical Design of the PIAE






160 mm
152.40 mm
61 mm
BATTERY
47 mm
120 mm
111.25 mm
35 mm
33 mm
USB MINI-B
CHARGER


160 mm
15.32 mm
97.67 mm
LCD SCREEN
~13 mm
33 mm
PCB B
60 mm
50.04 mm
~11 mm
85 mm
PCB A
7 mm
35 mm
6.7 mm
61 mm
BATTERY
152.40 mm
USB MINI-B CHARGER