FAST FRET TRAINER FINAL REPORT

By

Eli Hoon

Omeed Jamali

Murtaza Saifuddin

Final Report for ECE 445, Senior Design, Spring 2025

TA: Eric Tang

07 May 2025

Project No. 24

Abstract

FastFretTrainer is a system for testing a guitar player's ability to reproduce notes upon being prompted. The trainer consists of three main components: the App, Base, and Fob. The Fob is a small battery-powered PCB mounted to the guitar, which sends a digitized recording of the guitar's output via Bluetooth to the Base. The Base is connected to a PC via USB, over which it transmits the received Bluetooth data and receives commands to drive the LCD display. The App produces a user interface to give guitarists feedback and does the backend data processing to compare the played note to the requested note. Once a user selects a practice mode, a countdown begins, then the App sends a request to the hardware, which, once the data is ready, writes the recorded guitar signal back to the PC. Finally, the App's algorithm will judge the correctness of the played note.

Contents

1. Introduction

Beginner guitarists often have trouble learning to play the guitar since the notes are not marked on the instrument. The lack of markings often intimidates lower skill players because its wide range of notes is often overwhelming. We designed FastFretTrainer to make learning easier. FastFretTrainer utilizes its three components: the App, Base, and Fob to train and test the user on knowledge of the notes on the guitar neck. The Fob sits on the guitar itself and is responsible for sampling the guitar signal from the output jack upon request and then sending that data to the Base via Bluetooth. The Base serves as an intermediate between the Fob and the computer, it receives the sampled signal from the Fob and sends it to the PC via USB and also controls the LCD display on the Base enclosure to provide basic user feedback. The App does the Fast Fourier Transform (FFT) computation that enables us to compare the played note to the note that the App asked the user to play. The App is responsible for the flow control of the whole project, meaning that the Fob is waiting for the PC to tell the Base that sampling should begin before it will send any data back.

Figure 1 showcases our block design; there are several blocks included within our general design. First, under the Fob, the Amplifier Subsystem exists to both amplify the input guitar signal and add a DC offset. This DC offset is needed because our ADC Subsystem is only able to sample positive voltages up to 3.3 V. The ADC is responsible for correctly sampling the input guitar signal. The Bluetooth Subsystem sits on the Fob's ESP32 and is responsible for sending the sampled data to the Base PCB. Also on the Fob, the Power Supply Subsystem generates the positive voltages required to both power and bias the op-amp circuit, while also creating the negative voltage required to correctly bias the Amplifier's op-amp circuit. On the bottom right of Figure 1, the Base has its own ESP32 which is used for receiving the data that the Fob is sending. Both the Fob and the Base have the same USB to UART Subsystems which are responsible for programming the microcontrollers as well as data transmission in the case of the Base. The Base has a much simpler power generation scheme as it only requires 3.3 V to power the ESP32 and 5 V to power the LCD display. The LCD Subsystem is responsible for taking data from the App and displaying basic feedback. Finally, the Computer Subsystem houses our App and the serial connection over which the Base and PC communicate.

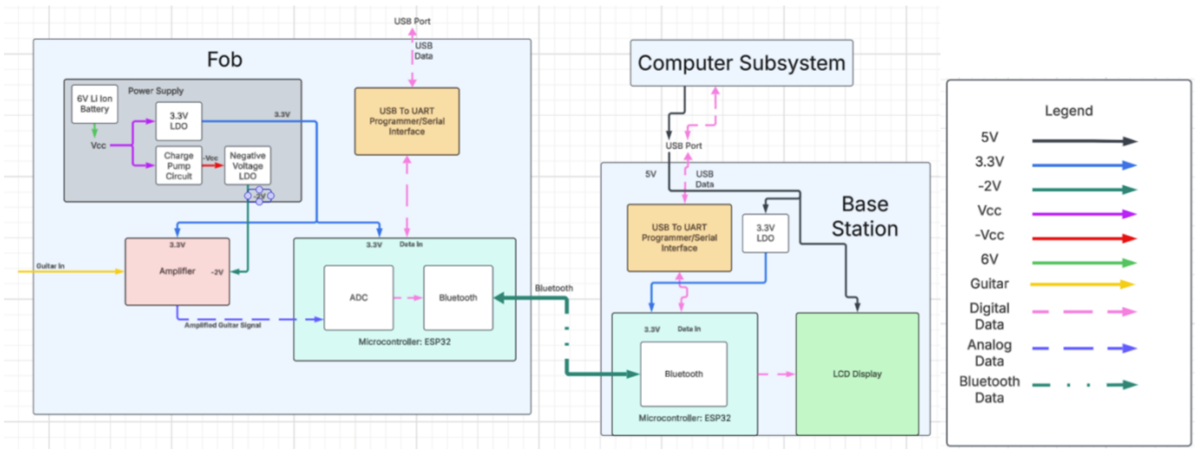


Figure 1. General Design Block Diagram

The high-level requirements for our design are:

- The fob must be able to communicate with the base station wirelessly from a distance of 1.5 meters without data loss.
- The local application on the laptop should be able to compare frequencies of the played and expected notes accurately after receiving data from the base station.
- The LCD display of the base station should be able to display basic values from the local application like how far off the note played was in cents.

If each of the above requirements are met, then our project will function as intended. If our device is able to wirelessly send the sampled data without loss, our App can accurately give feedback, and the LCD display can receive some of that feedback to display, then our project will succeed. From a subsystem level our design remained the same through the semester. However, there were changes in the schematic design that will be addressed in the following sections.

2 Design

Our project is split into three major design components: Software, Fob, and Base design. Below we discuss the design decisions and details for each block. This section also individually addresses the firmware design in our project.

2.1 Software Design

The main responsibility of our Software system is to help us reach our high-level requirement of giving accurate feedback to the user. By using digital signal processing, we give feedback to users on how well they played a note using cents as a measure. Cents are a metric used in music to compare the frequency of two notes. This is calculated via a logarithmic relation between the played and expected note seen in Equation (1). On the guitar, each fret is separated by 100 cents. This is what makes this metric key in our use case. By using cents, we are able to pinpoint the note that a user played so we can give them direct feedback on how to adjust and correctly play the note. Our Python backend is responsible for performing a frequency analysis to find and compare the user's played note to the expected note while communicating with the base station to send and receive data. The role of our frontend is displaying processed data via a local app written with the Flask library [1] incorporating HTML and CSS for styling. We felt that Python was the best choice for our software system due to its versatility when it comes to both data processing and support for advanced visualization through a plethora of libraries. An initial concern was that it wouldn't be powerful enough to process data optimally compared to other languages like C++, however it worked out well for our use case.

$$Cents = 1200 \cdot \log_2\left(\frac{f_{Played}}{f_{Expected}}\right) \tag{1}$$

2.1.1 Data Processing Backend

The first step in our backend's data processing flow is receiving the digitized signal from our base station. This is done using the PySerial library [2], which performs a the first step of the handshaking process further described in our Firmware Section 2.4. Once the header bytes of the sampled signal are found, the backend then unpacks the raw bytes into the true values representing the signal.

Numpy.fft [3] is then used to take the FFT of the signal so that the peaks at each frequency can be analyzed. The frequency our algorithm is concerned with is the fundamental frequency of the signal, marked with the orange x in Figure 2 below. The fundamental frequency represents the note played on the guitar. It is found at the first significant peak of the FFT, not necessarily the peak with the highest magnitude. While isolating this frequency could be a bit tricky with all the overtones adding additional peaks to the FFT, we used an algorithm that would find a relative threshold of magnitudes so that only significant peaks would be taken into account. This threshold only considers peaks above the average magnitude of the FFT added to an eighth of the maximum magnitude (shown in the dotted red line below in Figure 2 below). In addition to our threshold for the FFT magnitudes, we also used a threshold of 75 Hz for the frequencies since the lowest expected frequency of a note played on a 6-string guitar in standard tuning is about 82.5 Hz. Median filtering was also used to smooth the FFT for cleaner analysis. To isolate the fundamental frequency once some additional processing is done on the FFT, the find_peaks function from the signal extension of the SciPy library [4] is used to store the frequencies of all the peaks in a list. The frequency at the first index of this list corresponds to the fundamental frequency we are looking for.

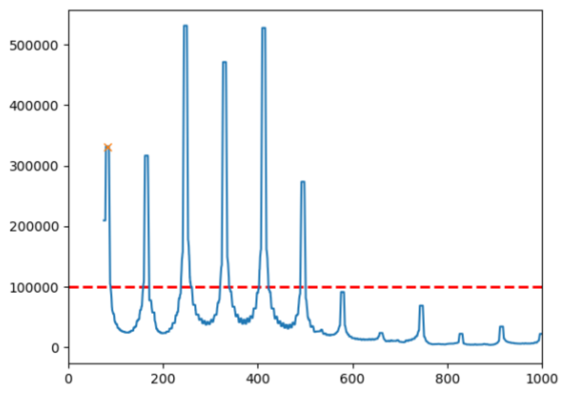


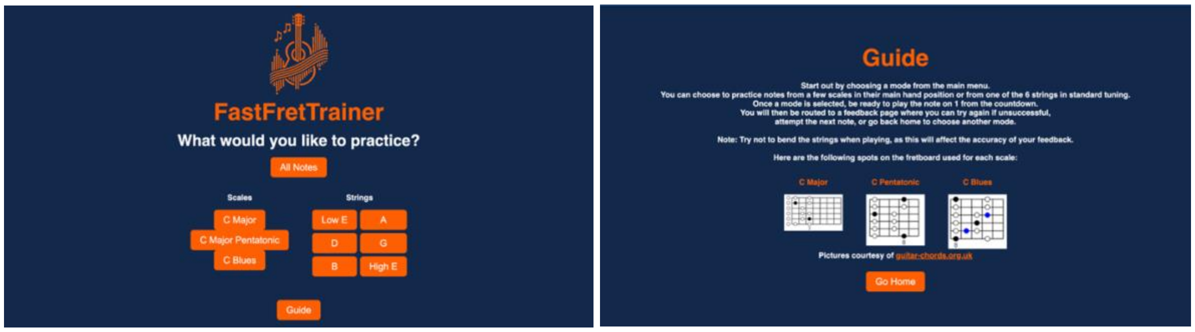
Figure 2. FFT (X axis – Frequency, Y axis – Magnitude)

Once the fundamental frequency of the signal has been found, we then use it as the played frequency in our cents calculation shown in Equation (1). After the cents have been calculated, feedback is then shown on our app and sent back to the base station to be displayed on the LCD. This feedback including the cents is converted to bytes for the top and bottom lines on the display and sent via the PySerial library.

In addition to calculating the cents off from the expected note, our backend also calculates the number of frets the user must move up or down the neck to adjust for the correct note. First, a search is performed on a sorted list of all the frequencies to determine the closest note to the one played. If the closest note is outside of the scope of the selected mode (string or scale), the user is asked if they are on the correct string or scale via the frontend. Otherwise, using an additional sorted list only containing the notes within the current mode, the distance is calculated between the indices of the played and expected notes and is used to determine how many frets or spots on the scale the user must move. The algorithms and operations described in this section are packaged into functions called in our integrated App code.

2.1.2 App Frontend

Our App uses a local Flask application to provide a graphical interface and give more complex feedback than what is shown on the LCD display. While Python is used to setup the app, route the pages, and perform the data processing in the backend, HTML and CSS are used to style our app and improve its usability. When the app is launched, the user is asked to reset both the Fob and the Base Station to establish their connection. Once they have been connected, the user is then taken to the home screen shown in Figure 3 below.



Guide

Start out by choosing a mode from the main menu.
You can choose to practice notes from a few scales in their main hand position or from one of the 6 strings in standard tuning.
Once a mode is selected, be ready to play the note on 1 from the countdown.
You will then be routed to a feedback page where you can try again if unsuccessful,
attempt the next note, or go back home to choose another mode.

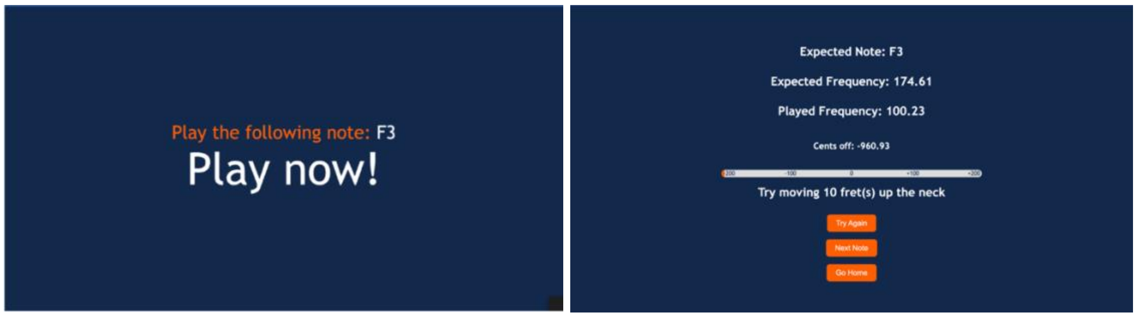
Note: Try not to bend the strings when playing, as this will affect the accuracy of your feedback.

Here are the following spots on the fretboard used for each scale:

Pictures courtesy of guitar-chords.org.uk

Figure 3. App Home (Left) and App Guide (Right)

At the home screen, the user can choose from a variety of modes. These include individual strings on a guitar in standard tuning (beginner level), a few commonly used scales (intermediate level), and a combination of all notes up to the 12th fret on the strings of the guitar (advanced level). There is also a guide that the user can view so that they can learn how to use the app once they choose a mode shown in Figure 3. The buttons shown on each page are used to navigate through the app in the event of a click. As shown in the guide, specific positions of scales were used for our implementation. There are a variety of ways each scale can be played, but we chose the most common positions that are initially taught and used. The scale diagrams shown in Figure 4 were pulled from the website Guitar Chords Scales and More. [5]. Once the user selects a mode, they are asked to play a specific note and are given a three second countdown. They are then asked to play when "Play Now" is shown as seen in Figure 4 below.



Play the following note: F3
Play now!

Expected Note: F3

Expected Frequency: 174.61

Played Frequency: 100.23

Cents off: -960.93

Try moving 10 fret(s) up the neck

Figure 4. App Countdown Page (Left) and App Feedback Page (Right)

The timing of this process is a crucial aspect of the design. It's important for us to give the user enough time to adjust and play the expected note. It is also important that the user plays the note exactly when they are prompted so that the signal that is sampled after the "Play Now" is strong and doesn't decay, which is especially key for higher notes. After the note is played and the app processes the sampled signal, the user is then taken to a page with feedback shown in Figure 4 above.

As shown in Figure 4, the user can see the played and expected frequencies as well as the cents off shown both as a value and a position on a scale from –200 to +200 cents. If the note played is further off than this range, it will just be shown at the corresponding end of the scale. A threshold of +/- 50 cents is used to determine whether the note played was correct or incorrect to account for potential noise or being slightly out of tune. In addition to the metrics previously mentioned, the user can also see the number of frets they need to adjust detailed in Section 2.1.1. If the user played the note incorrectly, they have the option to try again, move to the next note, or go back to the home screen. Otherwise, an alternate screen is shown that allows them to move to the next note or navigate home. Essentially, the app can be considered as a state machine with the states styled in HTML and CSS and Python controlling the transitions to each state.

2.2 Fob Design

The Fob's major responsibility is to be able to amplify and correctly sample the input guitar signal, then send it via Bluetooth to the Base for processing. The Fob contains four major systems, the amplifier, power supply, USB/UART, and ESP32. The following sections will describe the design decisions behind each system. There will be a small section that mentions the ESP32 from a physical design perspective, however, the discussion will be brief since most of the work related to the ESP32 will be mentioned in the firmware section of this report.

2.2.1 Amplifier Subsystem

The voltage level of a typical guitar signal before amplification is ~50 mVpp, thus amplification is required if the signal is to be accurately sampled. One way to do this is to buy a purpose-built audio amplifier that is designed to amplify with little noise in the audio range. However, we chose not to do this because many of those amplifier designs expect to be driving an 8 Ω load (speaker) and are overkill for our low power design as they require high output power. We also were planning on driving a high impedance load, namely the input of an ADC, so the capability to drive a speaker is not required. Thus, we chose our summing op-amp design because it enables the highest level of circuit control, while remaining relatively simple from an implementation perspective. A key design constraint that affects our amplifier design is the fact that our ADC can only sample positive voltage, namely 0 V to 3.3 V. Since our guitar signal is essentially a sum of sinusoids, negative voltage components are present. That means our design must add a DC offset to our amplified guitar signal. After iterating on our design to ensure correct operation, Figure 5 showcases the final design.

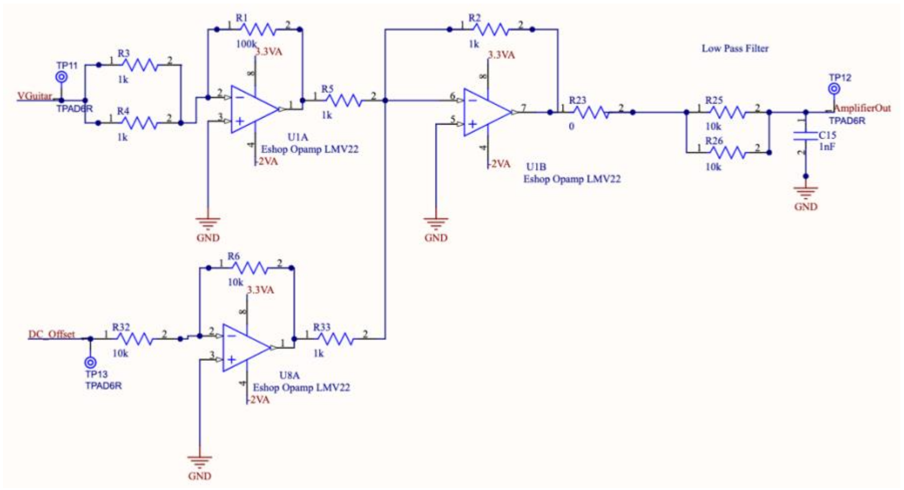


Figure 5. Amplifier Schematic

Figure 5 can be split into four major parts, each with their own relevant design equations. First, op-amp U1A serves as an inverting amplifier for the input guitar signal. All op-amps are in the inverting configuration because they are less sensitive, have a better small signal response and similar large signal response when compared to the non-inverting configuration [6]. The U1A op-amp setup is designed to follow Equation (2).

$$V_{AmplifiedGuitar} = -V_{Guitar} \cdot \frac{R_1}{\left(R_3 \backslash\backslash R_4\right) + R_{GuitarPickup}} \tag{2}$$

$$V_{AmplifiedGuitar} = -V_{Guitar} \cdot \frac{100,000}{500 + 7,500} = -12.5 \cdot V_{Guitar} \tag{3}$$

Equation (3) shows that with our resistor values and a guitar pickup resistance of 7500 Ω, the input guitar signal is amplified 12.5 times larger than its original. Depending on how hard the string was picked, the input guitar voltage will change. The typical voltage seen in our testing was 50 mVpp or 25 mV maximum amplitude. If the input guitar signal is 50 mVpp then after amplification the output will be 625 mVpp which is excellent for our application as we need to ensure that after adding a DC offset, the signal still remains within 0 V to 3.3 V. The next op-amp is the unity inverting buffer placed on the DC offset, namely U8A in Figure 5. The Resistor network surrounding the op-amp follows Equation (4).

$$V_{BufferedDCoffset} = -V_{DCoffset} \cdot \frac{R_6}{R_{32}} = -V_{DCoffset} \tag{4}$$

The purpose of this op-amp is to buffer the DC offset source from the rest of the amplifier circuit. This was done because in a previous iteration of the design, no buffering was done which yielded unexpected and incorrect output. The value of the DC offset is 1.65 V. This was chosen since it gives the most room above to 3.3 V and below to 0 V, maximizing the amount the guitar signal can be amplified without clipping. Since now both the DC offset and input guitar signal have been buffered, amplified to differing degrees, and negated, they must be summed and then negated again to produce an output within the ADC's range. That functionality is accomplished by op-amp U1B in Figure 5. Similarly to the previous two op-amps, U1B inverts and amplifies according to the input and feedback resistances as shown in Equation (5).

$$V_{Output} = -\left(V_{AmplifiedGuitar} \cdot \frac{R_2}{R_5} + V_{BufferedDCoffset} \cdot \frac{R_2}{R_{33}}\right) \tag{5}$$

$$V_{Output} = -\left(V_{AmplifiedGuitar} + V_{BufferedDCoffset}\right) \tag{6}$$

$$V_{Output} = 12.5 \cdot V_{Guitar} + V_{DCoffset} \tag{7}$$

As Equation (7) shows, the final output voltage after summing includes the amplified guitar signal plus the DC offset as requested. Finally, to limit high frequency noise in the output signal going to the ADC a low pass filter was included.

$$f_{3dB} = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot (5,000\,\Omega) \cdot 1\,nF} = 31,830.9\,Hz \tag{8}$$

The 3dB cutoff frequency shown in Equation (8) was chosen to be significantly large enough to limit attenuation of our desired amplifier output, but also low enough to cut out as much high frequency noise as possible. All these pieces together create an amplifier which can amplify the input guitar signal and add a DC offset while keeping the output between 0 V and 3.3 V and reducing high frequency noise.

2.2.2 Power Supply Subsystem

On the Fob, the Power Supply Subsystem is responsible for both powering the ESP32 and biasing the amplifier circuit so that it can function as designed. This means that we need to supply both positive and negative voltages, 3.3 V and -2 V respectively. One way of doing this is to use a single positive voltage LDO to step down the 6 V output from the battery and use a Switched Capacitive Inverter to negate the input voltage. There are a couple of issues with this design, first the analog and digital positive voltages are not separated, leading to excess noise on the positive power lines from the digital circuitry. The output negative voltage also has a large 12 kHz component from the inverting chip which is undesirable. Thus, we decided to improve upon this design to best fit our use case. To remedy these issues, a second positive voltage LDO and a negative voltage LDO were added as shown in Figure 6.

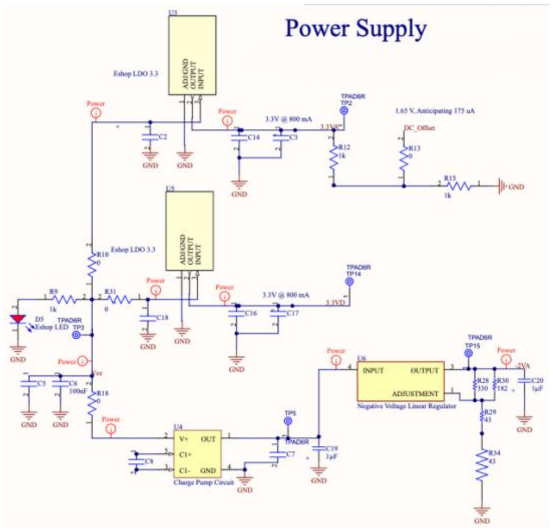


Figure 6. Fob Power Supply Schematic

The addition of the second positive voltage LDO enables our design to isolate the digital noise from the positive voltage power supply, increasing the signal integrity of our amplified guitar signal. A simple voltage divider was used with two 1 kΩ resistors to convert analog 3.3 V into 1.65 V for our DC offset.

The capacitor choices on the input and output of the positive voltage LDOs are recommended by the datasheet [7]. Since 22 μF capacitors were not immediately available, two 10 μF capacitors in parallel were used instead. The capacitive configuration surrounding the Switched Capacitive Inverter is shown in the datasheet [8] as well. Adding the negative voltage LDO reduces the presence of the 12 kHz switching frequency and brings the output negative voltage to -2 V as required by our amplifier subsystem. The datasheet [9] guided the choice of resistor values on the output of the negative voltage LDO to choose the correct output voltage.

2.2.3 USB/UART Subsystem

Another key component of this design is the ability to program and debug the ESP32 microcontroller. There are two main ways to allow serial communication with the ESP, using an external UART programmer, or including an on-board USB to UART conversion IC. Since this same design is used on the Base for serial communication with the PC, it is desirable to have a simple interface for users to connect to our PCB. Therefore, we chose to include the USB to UART conversion in our design. With our current design a user only needs a MicroUSB cable, not a UART programmer module that doubles as a serial interface. We utilized the FT232RL chip and a MicroUSB port in our design. Our implementation connects the TX/RX pins to the interface of the FT232RL chip and connects the USB data pins from the FT232RL chip to the Micro USB connector to facilitate serial communication. The basic design around the FTDI chip was taken from its datasheet [10], we modified the final design to include automatic programming with the MOSFET circuit shown in Figure 7. The design for the MOSFET circuit was inspired by a similar version of the circuit constructed using BJTs that is common on ESP development kits [11]. Generally, the automatic programmer uses the RTS and DTR signals to toggle the enable and IO0 pins such that the ESP is reset into bootloader mode and is ready to program. Looking toward the Micro USB connector show on the right, the data and power pins are connected to a TVS diode array to clamp transient voltage spikes and prevent damage to sensitive components downstream. The idea to include reverse voltage and TVS protection came from looking at the ESP development kit's MicroUSB connector implementation [11].

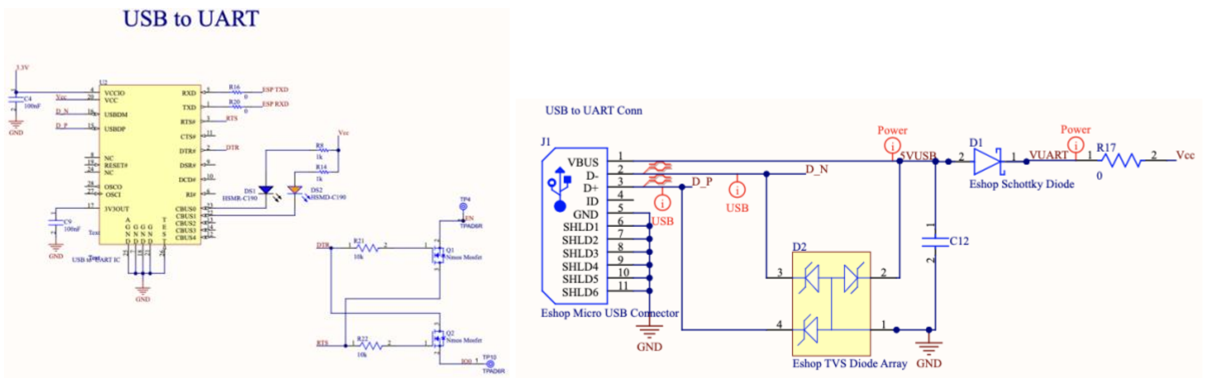


Figure 7. USB to UART (Left) and Micro USB Schematic (Right)

2.2.4 ESP32 Physical Design

The brain of our project is the ESP32 microcontroller. It is responsible for sampling on the Fob, and serial communication with the PC via the Base. Not to mention that the two ESPs are supporting the Bluetooth transfer of audio data. The design on the Fob and the Base differ slightly but are mostly the same. The only difference is that the Fob ESP has a GPIO used by the ADC for sampling, while the Base is using two GPIO pins to drive the I2C protocol required to interface with the LCD display. Both ESPs interface with the USB to UART design in the same way. The general design concept outside of the ADC and the LCD display interfaces was decided upon after reading through our ESP 32's datasheet [12]. The datasheet made it clear that the reset pin must be pulled high to prevent random resets without a press of the switch. It also made it apparent that we would require a button that could pull IO0 low, as a backup to our automatic programming circuit.

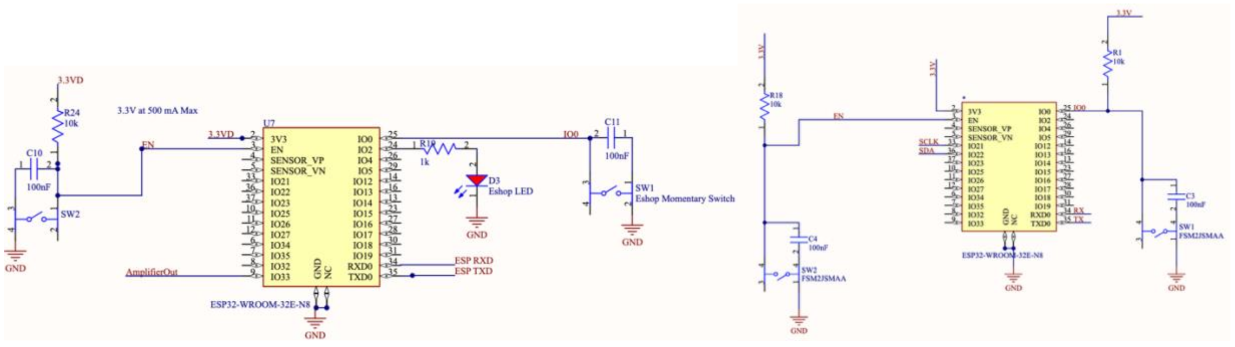


Figure 8. Fob ESP Design (Left) and Base ESP Design (Right)

Figure 8 shows that the designs are quite similar even though they may be laid out differently. The main differences being that on the Fob IO33 is used for the ADC to sample the amplified guitar signal and on the Base IO21 and IO22 are used to generate SCLK and SDA for the I2C LCD display.

2.3 Base Design

The Base station consists of an ESP32 microcontroller, LCD connector, USB/UART chip, and MicroUSB port. The Base station is responsible for receiving the sampled audio data from the Fob, writing to the serial buffer for the PC to read, and for displaying feedback on the LCD screen. The Base station is powered via the 5 V supply given from the MicroUSB port when plugged into a computer. We decided to do this instead of using batteries because the computer will also collect data from the serial buffer of the ESP32 which relies on a physical connection. Since the ESP32 requires a 3.3 V power supply, we use an LDO to drop down the 5V input to 3.3 V. This ensures power is within the bounds of the ESP32's specification. The USB/UART design is the same as the Fob's (reference section 2.2.3 for details). See section 2.2.4 for discussion about the Base ESP32 physical implementation.

2.3.1 LCD Subsystem

Figure 9 shows an image of the 4-pin LCD connector that sits on the Base station. We are utilizing an LCD display with an I2C adapter due to its ease of configuration and control.

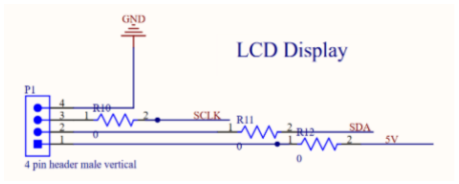


Figure 9. 2LCD Connector

We connect power, ground, serial clock and serial data pins which are driven from GPIO pins on the Base station's ESP32. When the user plays a note and feedback is displayed on the app, we send a more basic form of feedback to be displayed on the LCD. This means that the software is controlling the LCD display. We made this design decision because we are doing our FFT computation on the software side, so it's easiest to implement.

An alternative approach to this would have been to utilize an LCD display without an I2C adapter. This adds unnecessary complexity as we would need to drive all 8 data lines to the LCD display versus just one utilizing an adapter.

2.4 Firmware Design

The firmware we designed for our project uses the ADC on the Fob to digitize the input guitar signal, and Bluetooth on both ESP32s to transfer the data from the Fob to the Base.

2.4.1 ADC/Wireless Subsystem

We designed firmware to program and configure the ESP32's ADC and Bluetooth. The ADC was configured on the Fob's ESP32 as the Fob is responsible for digitizing the input guitar signal. Bluetooth was configured and setup on both ESP32s, and the Base was configured as a master device (the Base will connect to the Fob's MAC address).

We utilized ADC continuous mode on the ESP32 as it provides a high sampling frequency (44.1 kHz) to ensure an accurate FFT. We chose to use a 44.1 kHz sampling rate since it is an industry standard. Each time we sample a guitar signal, we collect 5,512 samples. Equation (11) shows the FFT resolution is dependent on the sampling frequency and the number of samples we collect.

$$f_s = 44,100 \tag{9}$$

$$N_{Sample} = 5512 \tag{10}$$

$$FFT_{Resolution} = \frac{f_s}{N_{Samples}} \tag{11}$$

Increasing the number of samples would result in a smaller bin width, which would increase the resolution of the FFT. We chose to collect 5,512 samples as we had enough space on the ESP32, and this would lead to our design being highly performant.

We struggled with continuously sampling data since guitar notes decay quickly, especially if they're high frequency. To have a high accuracy FFT we needed to sample when the signal is at its strongest, which occurs right when the guitar note is played. We designed a Bluetooth handshaking protocol represented in Figure 10 capture the signal at its strongest point.

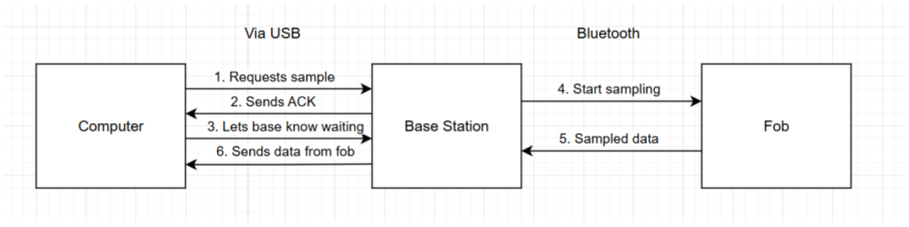


Figure 10. Bluetooth Handshaking Protocol

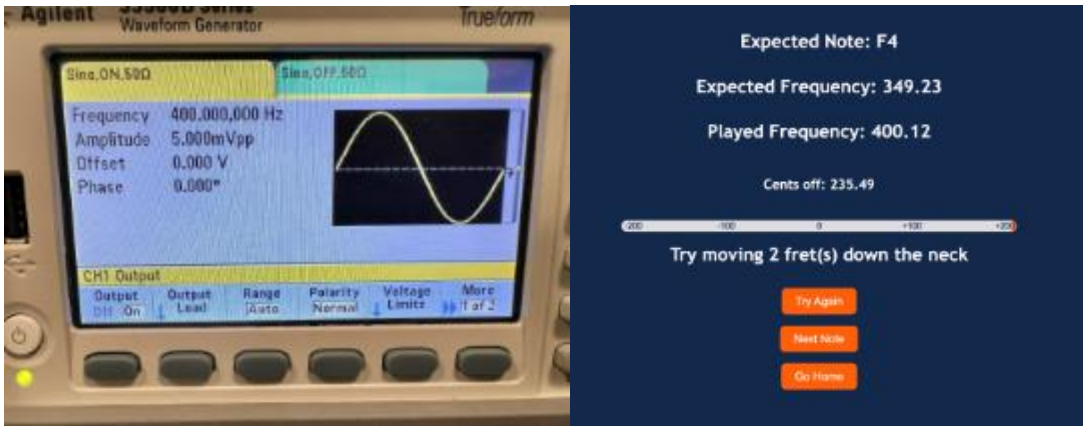
Initially the computer requests a sample from the Base station, and upon acknowledgement from the Base, that request is transferred to the Fob via Bluetooth to signal the system to begin sampling. This request begins when the App alerts the user to "Play Now", which is described in section 2.1.2. Once the data is sampled, it's sent to the Base via Bluetooth and then written to the serial buffer for the App to begin data processing.

3. Design Verification

Thorough verification needs to be completed to ensure proper functionality of the project. To view the full list of requirements and verifications view Appendix A.

3.1 Software Verification

Correct functionality for the software side of our product requires taking the FFT on the incoming digitized guitar signal correctly, and displaying that feedback on the app. Also being able to easily use the app is necessary for a good user experience. For specific requirements, refer to the PC section of Table 1 in Appendix A. To verify that we were taking the FFT and reporting the frequency correctly, we utilized a signal generator and set it to send a 400 Hz sine wave. If the FFT is taken correctly, and our frequency isolation algorithm functions, our app will report that the "played" frequency as 400 Hz. Figure 11 below shows the 400 Hz sine wave input to the Fob and the feedback showing the fundamental frequency to be 400.12 Hz which aligns with our requirements.



Frequency 400.000,000 Hz
Amplitude 5.000mVpp
Offset 0.000 V
Phase 0.000°

Expected Note: F4

Expected Frequency: 349.23

Played Frequency: 400.12

Cents off: 235.49

Try moving 2 fret(s) down the neck

Figure 11. Sine wave and feedback shown on app

3.2 Fob Verification

The main purpose of the Fob is to sample the amplified guitar signal and send the data over Bluetooth. This means that our Amplification Subsystem, Power Subsystem, and USB/UART Subsystem must all be functional to accomplish our purposes.

3.2.1 Amplifier Subsystem Verification

The input guitar signal contains negative voltage components, and our ADC requires the input signal to reside within 0 V and 3.3 V. This means that we must add a DC offset to our input such that it is not clipped when the ADC is sampling. To verify that this system performs as expected, two tests will be run: one with a sine wave generated by a signal generator, and another with the guitar connected. Since the average guitar input signal is about 50 mVpp and it is amplified by a factor inversely related to the output source resistance as shown in Equation (2), for an equivalent effect to be created using a signal generator with an output resistance of 50 Ω, the signal amplitude must be about 3.4 mVpp. Our test input will use a 5 mV amplitude or 10 mVpp to simulate a worst-case scenario. For our amplifier to pass verification, both the simulated input from the signal generator and the input from the guitar must be amplified and be within the 0 V to 3.3 V range. The results are shown in Figure 12. To read the requirement set for the amplifier, check Appendix A.

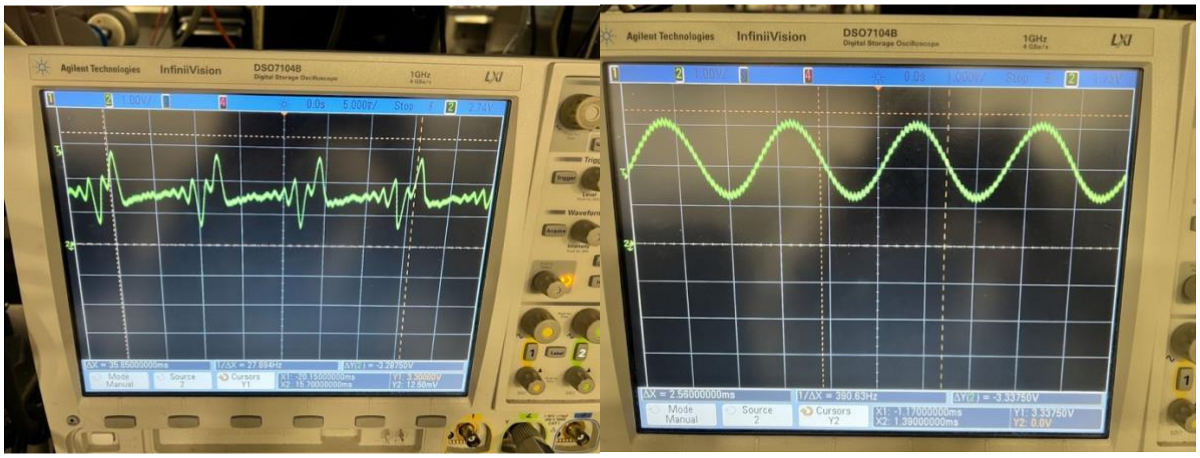


Figure 12. Guitar Test (Left) and Signal Generator Test (Right) (X axis – Time, Y axis – Amplitude)

3.2.2 Power Supply Subsystem Verification

The Fob is battery powered, and the Base is powered via the MicroUSB connector plugged into a computer. Refer to the power supply section of Table 1 in Appendix A for specific requirements. To verify that the Base was getting sufficient power, we probed pins for the FTDI chip and LDO to ensure they were getting 5 V of power. In terms of the Fob, we are using four 1.5 V batteries in series which gives us 6 V, and we need to step this down to 3.3 V to power the ESP32. We also need to ensure that the output of the negative voltage LDO is at –2 V for biasing the op-amps. Figure 13 shows waveforms of the output voltages (negative and positive voltages respectively). You can see that the negative LDO outputs close to –2 V, and the power circuit successfully steps down the supply voltage to within 10 % of 3.3 V.

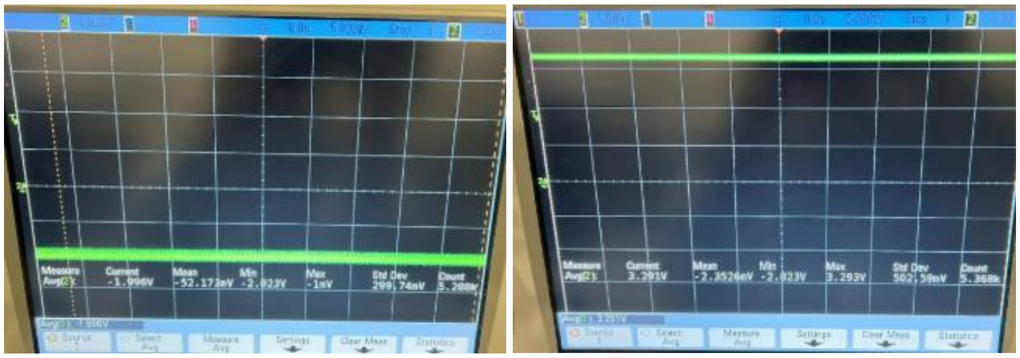
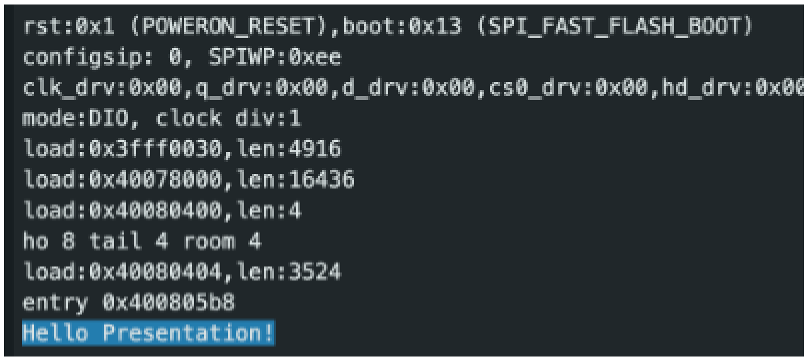


Figure 13. Negative and Positive Output Voltages on Fob (X axis – Time, Y axis – Amplitude)

3.2.3 USB/UART Subsystem Verification

Correct functionality for the USB/UART subsystem will allow automatic programming of the ESP32 and serial communication. Refer to the USB to UART section in Table 1 in Appendix A for specific requirements. When the components on the Fob and Base were soldered onto the PCBs, we ensured continuity between TX/RX pins on the FTDI chip and ESP32 respectively and the data pins from the FTDI chip to the MicroUSB connector. To verify serial communication was functioning correctly, we flash the ESP32 with basic code that will print to the serial terminal. If a message gets printed to the serial terminal and we can view it, this verifies functionality. Figure 14 below shows the ESP32 being reset, running our code, and a printing a message to the serial terminal.



```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:4916
load:0x40078000,len:16436
load:0x40080400,len:4
ho 8 tail 4 room 4
load:0x40080404,len:3524
entry 0x400805b8
Hello Presentation!
```

Figure 14. ESP32 Print Message for Verification

3.3 Base Verification

For the Base to function correctly, we verify hardware connectivity of the USB/UART chip to the ESP32, and to the MicroUSB connector which ensures proper programming and serial communication. Refer to the USB to UART section in Table 1 from Appendix A for specific requirements. USB/UART verification is covered in section 3.2.3. For the Base, the main functional requirements are for the LCD display.

3.3.1 LCD Subsystem Verification

Refer to the LCD section in Table 1 from Appendix A for specific requirements. Figure 15 below shows the results from our verification tests for the LCD display. When the app computes the FFT of the played note and it is within the 50 cents threshold of the expected note, the app marks it as correct and displays a “Correct!” message on the app. The computer system sends write commands to the LCD to provide basic feedback.

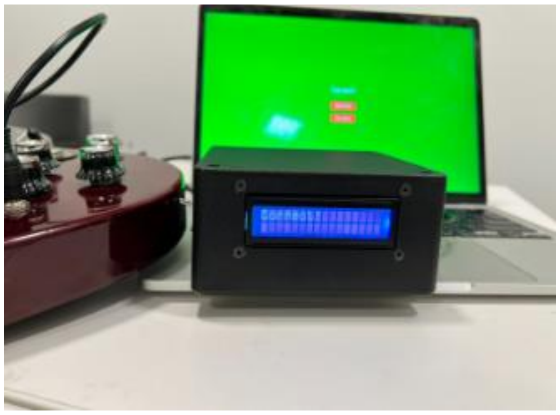


Figure 15. LCD Display Verification

3.4 Firmware Verification

Once the hardware for the Fob and Base PCBs was built and working correctly, we verify the ADC and Bluetooth. Refer to the ADC and Wireless sections of Table 1 in Appendix A for specific requirements.

To verify Bluetooth Serial connection between the Base and the Fob, we first connect the two devices together (the Base connects to the Fob's MAC address). Then we attempted to write to the serial terminal of the other device (i.e the Fob would write to the Base device, and the Base would read and print the message to its serial terminal). Once this basic connection is working, we connect a signal generator to the ADC GPIO pin on the Fob's ESP32. We read data from this pin, collect that data into a buffer, and send the buffer of data to the Base and attempt to read it. Once this basic data transfer is working, we begin reading a sine wave to verify more complex input signals are also supported.

To verify the ADC's functionality, we connected a signal generator to the ADC on the ESP32 and start sampling. We digitize the data and send it to the Base since the Bluetooth has already been verified. We then takw the digitized signal on the software end, and recreate a plot of the analog signal and it matches. Figure 16 below is a recreated software plot of a digitized 400 Hz sine wave.

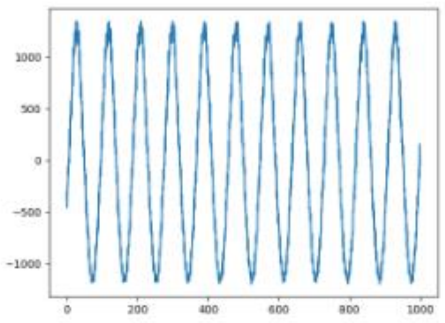


Figure 16. Recovered Sine Wave (X axis – Time, Y axis – Amplitude)

This ensures verification of the ADC and Bluetooth, as well as completes full software integration of the project.

4. Costs

4.1 Parts

Table 2 contains all the parts used to build both the Fob and Base PCBs. Each PCB has its own subtotal which will be summed to find the final parts total price.

Table 2 Parts Costs

Part	Manufacturer	Retail Cost ($)	Number Required	Actual Cost ($)
Fob				
FT232RL	FTDI	5.17	1	5.17
LM828M5	Texas Instruments	0.65	1	0.65
Tantalum Capacitors	Kyocera AVX	0.11	2	0.22
AA 1.5V Li ion batteries	Duracell	2.00	4	8.00
LM337KVURG3	Texas Instruments	1.75	1	1.75
Resistors	Multiple	0.02	34	0.68
Capacitors	Multiple	0.015	20	0.30
Micro USB	Amphenol ICC	0.26	1	0.26
N Channel Mosfet	International Rectifer	0.37	2	0.74
LVM922 Op-amp	National Semiconductor	0.73	2	1.46
AZ1117CD-3.3TRG1	Diodes	0.64	2	1.28
ESP32-WROOM-32E	Espressif Systems	2.80	1	2.80
Subtotal				23.31
Base				
ESP32-WROOM-32E	Espressif Systems	2.80	1	2.80
FT232RL	FTDI	5.17	1	5.17
Resistors	Multiple	0.02	34	0.68
Capacitors	Multiple	0.015	20	0.30
Micro USB	Amphenol ICC	0.26	1	0.26
N Channel Mosfet	International Rectifer	0.37	2	0.74
AZ1117CD-3.3TRG1	Diodes	0.64	2	1.28
LCD Header	TE Connectivity	0.11	1	0.11
LCD Display	SunFounder	8.95	1	8.95
Subtotal				20.29
Total				**43.60**

Assuming tax and shipping adds an extra 15 %, this brings our total price to 1.15*$43.60 = **$50.14**.

4.2 Labor

Assuming that each team member works, on average 5 hours a week on this project and is paid $50 per hour of work over this 16 week semester, our labor costs are:

$$\frac{\$\ 50}{hour} \cdot \frac{5\ hours}{week} \cdot 16\ weeks \cdot 2.5 = \$\ 10,000 \qquad (12)$$

Since our team has three members, this brings our total labor cost to 3 * $10,000 = $30,000

The machine shop worked on our project for approximately 4 hours, thus bringing their labor total to 2.5 * 50 * 4 = $500.

Thus, our total cost for the whole project is $500 + $50.14+ $30,000 = $30,550.14.

5. Conclusion

5.1 Accomplishments

Looking back upon completion of our project, we feel that there are a lot of areas in which we succeeded. Most importantly, we were able to meet all our high-level requirements. However, on top of our high-level requirements, we surpassed our initial goals for usability and feedback. Our guide makes it much clearer how the App flows for the user once they select a mode. Giving feedback on the number of frets to adjust instead of just the number of cents is also more useful to users and is easier to wrap their heads around if they aren't familiar with musical metrics like cents. These additional features as well as the way in which our App is styled make for an App that is both easy to navigate and visually appealing. On the lower-level side, the resolution at which we can sample is ample for our use-case. Through our own tests, we also felt that our project was not only an interesting prototype for a senior design class, but also a useful tool for guitar players at all levels. Outside of our technical accomplishments, we also learned a lot about metrics used in music, real world applications of signal processing, and writing firmware for efficient Bluetooth communication on microcontrollers like the ESP32.

5.3 Ethical and Safety Considerations

Throughout our work on the project, there were some ethical and safety considerations that we had to make to respect the work of others, be transparent, and ensure that no one would be harmed using our design. To make sure we are giving credit to those who laid the foundation of our work through libraries or datasheets we've used, we made sure to abide by Section 1.5 of the ACM Code of Ethics: "respect the work required to produce new ideas, inventions, creative works, and computing artifacts" [13]. For this reason, we have included references to all external resources that we have used to put together and inspire our design. In addition to giving credit where it is due, we felt that it was important to be open to suggestions and transparent about our own challenges faced, following section 1.5 of the IEEE Code of Ethics: "to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, to be honest and realistic in stating claims or estimates based on available data, and to credit properly the contributions of others" [14]. Through our weekly TA meetings, we were able to receive feedback that guided the direction of our project while being able to honestly address our own concerns and obstacles.

To ensure that our design is safe for all to use, we followed Section 1.1 of the IEEE Code of Ethics: "to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment" [14]. This mainly involved making sure our battery usage for our Fob power system was done in a safe manner and that our wireless communication wouldn't interfere with other devices or present other safety issues. When using batteries, we held ourselves accountable to ensure proper storage in a cool, dry place, check for shorts before connection, and periodically monitor the output current during operation to verify that it was within a safe range. Thankfully, the chips we used for our Bluetooth wireless communication are both FCC and BQB certified, meaning they have already been vetted through a series of industry-standard checks to ensure they are safe and reliable for wireless communication.

5.4 Future Work

While we are proud of the work we put in and what we have been able to achieve with our project, there are some areas that we recognize could use further work to significantly improve our design. On the hardware/firmware side, we think continuous sampling would be a significant improvement over our current design as it would mitigate latency introduced by needing to request samples with handshakes. In addition to continuous sampling, multithreading the sampling and sending of data would also minimize latency by allowing concurrent operations of parallelizable tasks instead of executing everything sequentially. We also feel that our enclosures could be made smaller so that the Fob could rest more comfortably on the guitar. On the software side, adding additional modes such as more scales, support for additional tunings, and the ability to practice chords would make our project much more versatile and all-encompassing for guitar players. A continuous training mode where the App would automatically navigate to the next note would also make the practice for the user flow better so that they aren't forced to click on buttons after each note. Our algorithm for finding the fundamental frequency and frets off could also be further optimized to use less data structures and reduce the runtime. We also feel that our Base station could be eliminated completely as the LCD display gives more primitive feedback compared to the App and sampled signals can be sent to the laptop via Bluetooth directly.

Bibliography

[1] Pallets, "Flask Documentation," 2010. [Online]. Available: https://flask.palletsprojects.com/en/stable/.

[2] C. Liechti, "Pyserial Documentation," 2020. [Online]. Available: https://pyserial.readthedocs.io/en/latest/.

[3] NumPy, "NumPy Documentation," 2022. [Online]. Available: https://numpy.org/doc/.

[4] SciPy, "find_peaks Documentation," 2025. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/signal.html..

[5] "Referenced Scales in Guide," 2024. [Online]. Available: https://www.guitar-chords.org.uk/guitarscales/guitarscales.html.

[6] Texas Instruments, "Op-amp Datasheet," 25 10 2011. [Online]. Available: https://rocelec.widen.net/view/pdf/3uc41osp7e/NATLS12377-1.pdf?t.download=true&u=5oefqw.

[7] Diodes Incorporated, "Positive Votlage LDO Datasheet," 9 2022. [Online]. Available: https://www.diodes.com/datasheet/download/AZ1117C.pdf.

[8] Texas Instruments, "Switched Capacitive Inverter," 5 2013. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm828.pdf?ts=1746580633297&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM828.

[9] Texas Instruments, "Negative Votlage LDO Datasheet," 1 2015. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm337.pdf?ts=1746564232702&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM337..

[10] FTDI, "FT232RL Datasheet," 5 2020. [Online]. Available: https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf.

[11] Espressif Systems, "ESP 32 Devkit Schematic," 12 2017. [Online]. Available: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf.

[12] Espressif Systems, "ESP32-WROOM-32E Datasheet," 2 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf.

[13] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: https://www.acm.org/code-of-ethics.

[14] IEEE, "IEEE Code of Ethics," 6 2020. [Online]. Available: ttps://www.ieee.org/about/corporate/governance/p7-8.html.

Appendix A   Requirement and Verification Table

Table 2   System Requirements and Verifications

Requirement	Verification	Verification status
Software Subsystem		
Accurately report the frequency of the note played on the guitar	Use a signal generator and oscilloscope to record the note frequency and compare it to what is reported	Verified (see Figure 11)
Give an accurate measurement of note difference in cents on the scale shown in the UI	Utilize an oscilloscope to measure frequency and manually calculate and compare the cents value	Verified (see Figure 11)
Properly interface with the USB connection to send data packets back to the Base Station	Call write commands via the PC for “Hello world” packets and see if they are displayed on the LCD display	Verified
Amplification Subsystem		
Signal input to the ADC must be between 3.3 V and 0 V	Use a 5 mVpp input and ensure the output is within 0 V to 3.3 V	Verified (see Figure 12)
DC offset must be half of the positive rail voltage.	Use a multimeter to ensure the offset remains within +/-15 % of the required 1.65 V	Verified
Power Supply Subsystem		
Four 1.5 V AA batteries in series will be used to power the fob and supply voltage to the ESP32 on the fob.	Measure and ensure the output remains between 5 and 6.4 V.	Verified
Ensure the power circuit steps down the supply voltage to 3.3 V	Use a multimeter to ensure we have 3.3V +/- 10 %	Verified (see Figure 13)
The output of the negative voltage LDO must be as close as possible to -2 V	Probe the output of the negative voltage LDO to ensure that the value is within 10 % of -2 V	Verified (see Figure 13)

Table 2 System Requirements and Verifications (Continued)

USB to UART		
Provide 3.3V-5.25 V +/- 0.5 % for power.	Use a multimeter to ensure Vcc is 3.3 V and no more than 5.25 V	Verified
The FT232RL must receive UART signals, convert to USB, and transfer to the PC.	Send a “Hello World” message from the ESP32.	Verified (see Figure 14)
When plugged in, the device must be recognized by a COM port.	On the PC side, The Mac should recognize the device as a USB device	Verified
The ESP32 TX/RX pins must correctly be configured to the FT232RL TX/RX pins.	Use a multimeter to check continuity between the pins respectively.	Verified (see Figure 14)
Firmware		
ADC Must be connected correctly to the amplified guitar signal	Verify channel selection and ADC configuration and data acquisition via software plotting	Verified
Bluetooth is correctly configured, and data transfer is successful up to distances of 3 meters.	Send “Hello World” packets from the fob to base	Verified (see Figure 16)
Ensure that the Fob and Base can connect reliably	Utilize the SerialBTM example sketch in Arduino IDE to connect devices together	Verified
LCD		
The I2C LCD1602 requires a 3.15 V - 3.45 V supply voltage	We can use a multimeter to probe and ensure 3.15 V - 3.45 V	Verified
The SCL and SDA pins must be correctly connected	Probe connections on ESP 32 and LCD display to ensure connection	Verified (see Figure 15)