SuperFret

Owen Ball, Ashwin Godura, and Tushaar Jain

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Traditional guitar training tools can show an image of note fingering to users, but going from this image to actually placing the fingers on the strings can be difficult. With the SuperFret system, LEDs on the guitar fretboard show users exactly where to place their fingers on the guitar. The system also detects where the user's fingers are located and when they strum the guitar, allowing the system to determine if the user plays the correct note. This enables users to learn the guitar more rapidly and engagingly.**

***Index Terms*—Fretboard, fret, guitar, metronome, MIDI file, NeoPixel (addressable LED), Raspberry Pi, string, strum, Teensy 4.1 (microcontroller), web app.**

I. Introduction

The SuperFret system aims to create a more intuitive guitar training tool for beginners. When first learning the guitar, beginners often struggle with translating an image of how to play a note to an actual finger placement on the fretboard, the part of the guitar where users place their fingers to change the pitch of notes. Traditional tools show beginners tabs or images of where to put their fingers, which they must first interpret, then look at the fretboard to place their fingers. For new guitar players, this increases the complexity of learning the guitar. Since beginners are already looking at the fretboard when playing a note, indicating where to put their fingers directly on it is intuitive. By using LEDs, or light-emitting diodes, to indicate to users where to place their fingers, the process of playing new notes and songs is expedited and made more natural for beginners.

While more advanced guitar players can learn to sight-read guitar tabs and images of notes, these skills take time to develop and build muscle memory. Jumping straight into reading tabs and notes can be overwhelming when learning guitar. The SuperFret system targets absolute beginner guitar players trying to pick up a guitar and play for the first time. Indicating to beginners where to put their fingers enables them to build finger dexterity and the skills to play notes without being inundated with foreign guitar notation. This removes one of the major hurdles beginner guitar players face, making playing the guitar more approachable and enjoyable.

The SuperFret system also detects the position of the user's fingers and when they strum, allowing them to receive real-time feedback to ensure they are playing the correct notes and strumming at the right time. A web app displays that feedback to the user, allowing them to see their progress and determine where to improve.

Guitar training resources are not a novel idea, with private teachers, training apps, and accessories being commonplace. Private teachers are costly, running around \$40-\$90 an hour [1]. This results in many individuals favoring personal training tools, such as apps showing them where to put their fingers and listen to their playing. While tools like this are affordable, they require users to look at a screen to determine what note to play and then try to match their fingers to the image on the screen. By integrating LEDs on the fretboard, the SuperFret system makes it easier for users to place their fingers in the correct location.

A handful of existing training tools integrate LEDs onto the fretboard, but these systems use audio to detect what the user is playing. These systems require a fairly quiet environment and take longer to analyze what note was played. The SuperFret system directly detects the user's finger locations, thus enabling rapid feedback and more accurate analysis of the user's playing.

Overall, the SuperFret system allows beginner guitar players to learn to play notes and basic songs quickly quickly. The system determines if the user is playing correctly and provides feedback and control over the system through a web app interface

II. Use-Case Requirements

The target users of the SuperFret system are beginner guitar players looking to improve their skills and play basic songs. As such, the use case requirements are informed with beginners in mind. Beginner guitar players should find the overall experience of the web app and hardware intuitive, as the goal of the project is to remove barriers to entry. From picking up the system to strumming notes, users should only need around 5 minutes to get started with the system. Users shall be able to upload MIDI files (file format for representing music) for songs they want to practice. The system should also support selecting between various playing modes to suit how the user wants to practice.

The system shall handle notes down to 1/8th notes at 100 beats per minute (BPM). This corresponds to 200 notes per minute maximum, or around 3 notes a second, faster than most beginner guitar players can handle. The target tempo should be indicated at a volume that is audible over the guitar. The system should be able to identify the user's finger placement and strumming with 99% accuracy, corresponding to approximately 1-2 missed notes per minute by the system. This is far lower than the number of mistakes the user makes, so this accuracy is sufficient for the system.

Additionally, the system must look, feel, and play like a standard bass guitar. This ensures users can apply the skills learned on the SuperFret system to other guitars.

III. Architecture and Principle of Operation

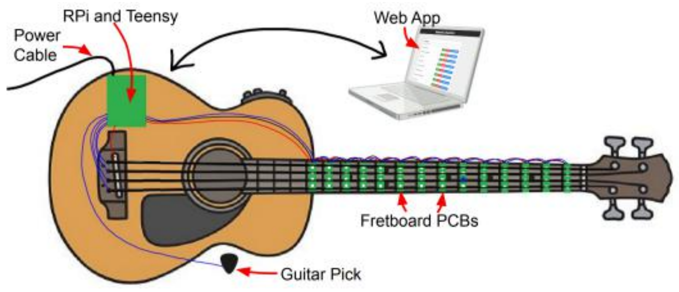


Fig. 1. A depiction of the SuperFret system. Guitar image from [2]

The overall system is shown in Fig. 1. The user interacts with the system through their personal computer by accessing a web app. They upload songs and choose which ones to practice on. When ready to practice, they click "Start" on the screen, place their fingers on the lit-up LEDs, and strum. Statistics about their playing are aggregated and displayed on the web app. Additionally, when a user starts a song, the web app renders a "virtual" guitar that mirrors the physical guitar and more easily allows users to visualize the song.

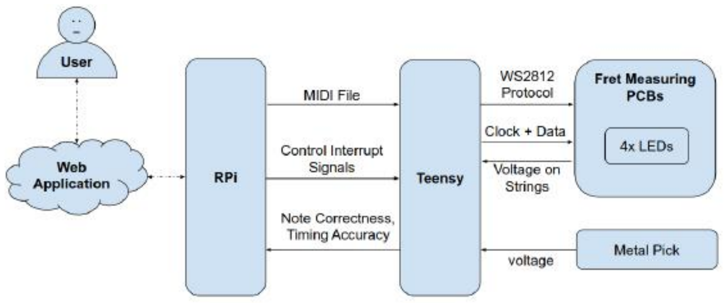


Fig. 2. High-level Architecture Block Diagram.

Overall, the system is composed of 3 parts – the web application ("web-app") hosted on a Raspberry Pi 4B ("RPi"), a Teensy 4.1 microcontroller, which is the brain of the embedded system, and the electronic hardware on the guitar. The user interacts with the system through the web application, which allows them to upload songs they want to learn, choose songs to practice, and receive statistics on their playing. The user uploads songs as MIDI files, which encode note and timing information for the song. The MIDI file is interpreted by RPi and visualized as falling notes on the virtual guitar. The RPi also converts the notes into (fret, string) coordinates on the fretboard, which are passed to the Teensy. The Teensy uses the coordinates and lights the corresponding LED on the fretboard to guide finger placement. The LEDs reside on Printed Circuit Boards (PCBs), 15 of which are embedded along the fretboard. The PCBs also contain circuitry to determine which note the user has fingered on the fretboard. Other electronic hardware on the guitar includes a metal pick and accompanying circuitry for strum detection. By detecting which note the user's fingers are on and when they strum it, the Teensy can determine deviations from the notes and timing information specified in the MIDI file and send aggregated statistics back to the RPi for display on the web app.

A. *Web Application*

As shown in the high-level block diagram (Fig. 2), the RPi hosts both the web app and communicates with the Teensy microcontroller. The web app is written in Python using the Django web framework, which combines the frontend, backend, and database into one Model-View-Controller design pattern (Fig. 3) to create web endpoints that the user can access via a web browser. From the website itself, the server provides all the functionality required for the user to control the guitar in an intuitive interface. User input is processed and forwarded to the microcontroller through 3 different communication "streams": bidirectional communication with the Teensy over UART accounts for 2 streams, and the third is for interrupt signals originating from the RPi that control the state machine (Fig. 9) inside the Teensy.

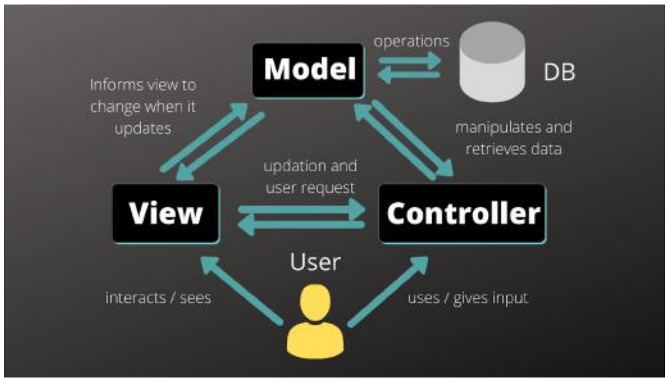

Model
operations
DB
Informs view to change when it updates
manipulates and retrieves data
updation and user request
View
Controller
User
interacts / sees
uses / gives input

Fig. 3. Django Web Frame Work Implements Model-View-Controller [9]

B. *Teensy and Embedded System*

Besides the 3 previous streams, the Teensy communicates with the electronic hardware on the right side of Fig. 2 through 4 streams.

First, the Teensy specifies the color of each NeoPixel LED on the fretboard through the protocol for WS2812 LEDs, which is the chipset the NeoPixel implements.

The Teensy determines where on the fretboard the user has pressed on a string by detecting the electrical contact between each of the 4 strings with 14 frets. This is done by applying a voltage stimulus to one of the 14 frets and then reading the voltage on each of the 4 strings. A high reading on a string indicates that the string is pressed down against the fret on which the voltage stimulus is being applied. By putting D-Flip-Flops between each fret, the voltage stimulus is clocked "down" the fretboard as if the D-Flip-Flops formed a shift register. This way, only 2 signals are required to create the voltage stimuli for the frets, as opposed to having 14 signals, with one per fret.

The Teensy also detects when the note is strummed by detecting when a voltage stimulus on the pick is conducted to a particular string. This is a change from the design report, where we intended on using audio-based detection of strumming.

C. Electronic Hardware

Each fret is associated with a fretboard PCB, which contains 4 LEDs, one per string. The fretboard PCB also has a D-flip-flop to receive the voltage stimulus from the previous fretboard PCB, apply the stimulus to the current fret, and forward the stimulus to the next fretboard PCB. Strums are detected using a pick with a metal electrode that applies a stimulus to the guitar strings.

The RPi is connected to the Teensy using a custom Pi-Hat PCB, which handles power distribution and I/O between all the system components. Power for the system can be provided either through a wall adapter or using a lithium polymer battery connected to a buck converter.

IV. Design Requirements

To meet the use-case requirements, several critical design specifications were established for both the hardware and firmware components, as well as the web application of the SuperFret system. For the hardware and firmware, achieving a latency of less than 50ms from strum detection to LED response (the threshold of human visual perception) is paramount to provide users with real-time feedback during practice sessions. Additionally, the hardware and firmware must support a strumming rate of up to 3.3Hz, or 200 strums per minute. The system should indicate the target tempo at a minimum volume of 70dB, which was found to be audible over the guitar notes.

To meet the use case requirement of being playable like a standard guitar, the system shall support ~2.5 octaves of notes, corresponding to 14 frets. Consequently, the guitar must support 60 individually addressable LEDs, 4 for each fret and 4 for the open string indicators. The rest of the board is unnecessary, as beginner users rarely use the highest notes on a bass guitar. The system shall support illuminating the entire fretboard at half brightness to enable arbitrary patterns to be displayed on the fretboard. Half brightness was selected to balance visibility and system current draw.

Safety is a key consideration, as the guitar strings are driven to 3.3V. According to IEC TS 60479-1, currents below 500µA through the body are imperceptible and safe. Therefore, the current that flows through the user under normal operating conditions should be under 500µA. Under abnormal operating conditions, such as if the system gets wet while being used, the current through the body should not exceed 1mA (the maximum current that can pass through a human body without impacting the user's muscles) [3].

The web application's design requirements focus on enabling the user to control the guitar and start/stop songs. The file upload capability should support up to 1GB of custom MIDI files for a personalized learning experience. The web application shall update in accordance with the user's playing within 250ms to ensure a cohesive user experience.

These design requirements ultimately ensure that the SuperFret system achieves the defined use-case requirements and provides a positive user experience. The quantitative specifications are summarized in Appendix Table I.

V. Design Trade Studies

A. Single-Board Computer vs Microcontroller

The main computer selected for the project was the Raspberry Pi 4B. The processing tasks associated with this project consist of running a web application, controlling the fretboard LEDs, reading from the fret sensors, and processing statistics. Both a single-board computer (SBC) and a WiFi-equipped microcontroller could perform these tasks. Single-board computers are typically worse at handling real-time interaction with their environment because the processor also handles the overhead of running the computer's operating system. Additionally, hosting the web app can introduce delays that do not meet input and output (I/O) latency requirements. Running the system off a WiFi-equipped microcontroller like the ESP32S3 would enable high-speed I/O. However, running the web app in parallel to this on the microcontroller would be challenging due to the single-threaded nature of most microcontrollers. Running the system off a microcontroller would also introduce significant restrictions on the web interface's functionality due to the microcontrollers' limited memory. For these reasons, we chose to pursue a split architecture, with an SBC running the high-level control of the system, namely running the web app, storing user-uploaded music, and coordinating the system's overall state. A microcontroller runs the real-time I/O without worrying about hosting a web app, allowing the target latencies to be achieved. This has the added benefit of improving our ability to parallelize work, with one team member working on the SBC and one on the microcontroller, rather than team members having to coordinate pushing and pulling software changes. The SBC chosen was the Raspberry Pi 4B due to its widespread documentation and support, and the microcontroller chosen was the Teensy 4.1 due to its plentiful GPIO pins and high clock speed.

B. Microcontroller Choice

Members of the group were already familiar with using several microcontrollers typically used in electronic projects, and familiarity was the main driving force behind selecting a microcontroller. We considered the Arduino UNO, Arduino Mega, Raspberry Pi Pico, Teensy 4.0, and Teensy 4.1. Of these, we wanted a microcontroller with fast clock speed to enable multiple tasks and enough memory to store a MIDI file's worth of data.

We found a benchmark that showed the Teensy class of microcontrollers were the fastest computers of the ones we were familiar with:

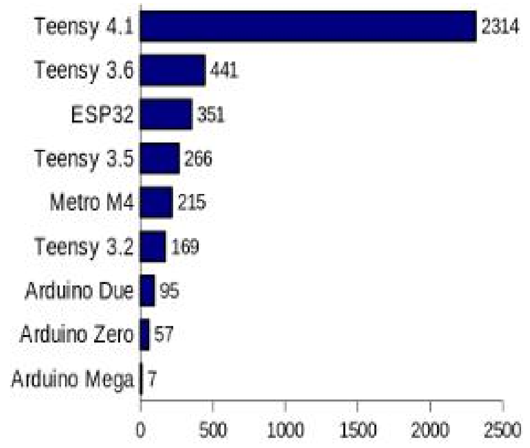


Fig. 4. The "CoreMark" CPU Performance Benchmark [4], [5]

We conservatively estimated the typical training song would be 2 minutes, with up to 200 notes per minute, and each note would take 5 bytes to specify in the MIDI format (2 for the duration in "delta ticks" and 3 for the event). Thus, we required a microcontroller with at least 2kB of memory. We eliminated the Arduino UNO, which only has 2kB of SRAM [6].

Furthermore, we desired a microcontroller with a great deal of flexibility in the I/O. The Teensy 4.1 supports 8 different hardware serial ports, almost all pins can act as interrupts, and all I/O pins are capable of functions such as pulse width modulation (PWM).

After considering the degree of prior experience, CPU performance, memory, I/O, and availability, we selected the Teensy 4.1 because it was strong across each desired trait, and we already had access to it, making it the cheapest option.

C. *Fret-Sensing Implementation*

To determine the user's finger placement, the system uses the 'switch' formed when the user presses a string into a fret. GPIO pins on microcontrollers are limited, and wires interfere with the comfort and usability of the guitar. A switch array can be employed to reduce pin and wire count. By driving each fret to 3.3V one by one and then reading the voltage on each string, the detection of any strings touching the 3.3V fret can be performed. This requires 18 GPIO pins - 4 for the strings and 14 for the frets. This still requires 14 wires to be run from each fret to the microcontroller. Since a switch array necessitates that each fret is driven to 3.3V one at a time, the GPIO count can be reduced to

$$4 \text{ Strings} + \text{ceiling}(\log_2(14)) = 8 \qquad (1)$$

pins using a decoder circuit. However, this would require decoding circuitry next to each fret, which would take up the limited space available. By using a "shift-register" style approach, with each fret requiring only a single D-flip-flop, the system can use only 6 GPIO pins, 4 for the strings, 1 clock line, and 1 data line. Excluding power wires, this solution, shown in Fig. 5, requires only 2 wires between each fret. These include a shared clock line and the data outputted by the previous fret's D-flip-flop. The only tradeoff of this implementation is that each fret requires a D-flip-flop, but this drastically outweighs requiring 14 individual wires for each fret.

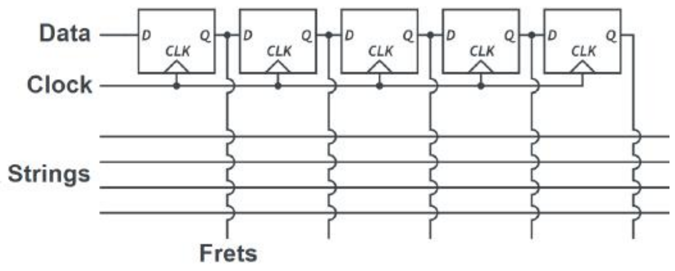


Fig. 5. A 6 GPIO method for reading finger positions

D. *Fretboard PCB Design*

Due to the finger placement sensing implementation making use of a D-flip-flop next to each fret, and the design requiring 4 addressable LEDs per fret, implementing a PCB to mount these components is the ideal solution. It would be possible to use commercial off-the-shelf (COTS) LED strips and run a separate wire to each fret, but it is not possible to buy LED strips with the exact spacing needed for the guitar strings. Additionally, this would require many wires for the finger placement sensing, as discussed previously.

There are a handful of ways to implement PCBs along the fretboard. The first way is to remove and replace the guitar's fretboard with a single PCB. This would completely eliminate the need for external wires along the fretboard but would introduce mechanical challenges. Since the fretboard holds the frets in place, we would need to devise a new way of mounting the frets securely, and we would need to perfectly match the spacing of the original fretboard to keep the guitar in tune. Additionally, this would require completely removing the guitar's fretboard, which can be challenging to perform due to the glue between the fretboard and the rest of the guitar. These factors increase the risk associated with the project, so we chose not to pursue removing the fretboard.

The other two implementations involve creating individual PCBs mounted next to each fret. This approach allows the fretboard to remain mounted to the guitar and removes the need to replicate the spacing between the frets on a PCB perfectly. These PCBs can be mounted on the fretboard or placed in carved-out channels next to each fret. The advantage of placing the PCBs on top of the fretboard is that no mechanical modification to the guitar fretboard is necessary. The disadvantages of this approach are that the fretboard is curved, as shown in Fig. 6, and that the frets only extend above the fretboard by 1.2mm.

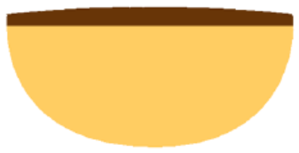


Fig. 6. Cross section of a guitar fretboard. The fretboard surface is curved, making PCB mounting difficult [7].

The curved surface of the fretboard makes mounting rigid PCBs directly to the fretboard difficult. A flexible PCB would resolve this issue by allowing the PCB to conform to the shape of the fretboard. However, since the frets only protrude from the fretboard by 1.2mm, the total height of the LEDs and the PCB must be below 1.2mm. The addressable LEDs being used have a height of 1.6mm, so to ensure these do not get in the way while the user plays the guitar, the PCBs sit in recessed channels in the fretboard. These channels can be flat on the bottom, meaning flexible PCBs are no longer necessary. Due to the higher costs and lead times associated with flexible PCBs, we pursued 14 rigid PCBs, each placed into carved-out channels next to the frets.

The primary challenge of this selected implementation is the carving of the channels into the fretboard. Guitars feature a metal support rod running along their length that must be avoided. Additionally, the wooden fretboard holds the frets in, and removing material from around them may loosen the fret. We created the channels using a Dremel and files, and during this process, one fret was knocked off the fretboard but could be replaced.

VI. System Implementation

Appendix Figure I has a more detailed technical block diagram of the system as a whole, beyond what was shown in Fig. 1. The system consists of three main subsystems – the user frontend hosted using the RPi, the physical hardware used to interface with the guitar and user, and the microcontroller system directly interacting with this hardware.

A. Raspberry Pi and Web App Subsystem

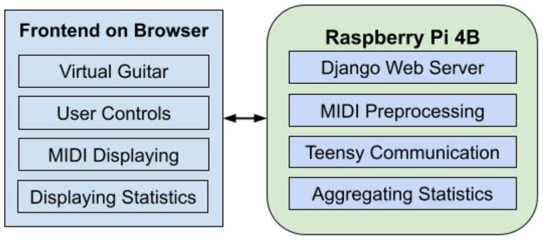


Fig. 7. Block diagram for the RPi and web app. Zoomed-in crop of block diagram in Appendix Figure I.

1) Django Web Server

The RPi hosts a web server powered by Python's Django Web Framework. This server creates a local endpoint reachable via a browser that responds with an HTML page containing all the functionality needed for the user to communicate with the guitar. Specifically, the web server implements these endpoints:

http://a2superfret.wifi.local.cmu.edu:8000/
- home - retrieves the home page
- addfile - uploads a file to
- deletefile/{songname} - deletes a file
- startfile/{songname} - tells guitar to start song
- stopFile - tells guitar to stop song
- getStats - get the user's statistics of previous songs

A SQL database houses all the file and user information to achieve a consistent state for the server. Each entry in the database represents a song and contain:
- name: name of the song/file
- file: the file path to the MIDI (actual file is stored in a separate folder)
- active: a boolean to store if the song is currently being played, keeps track of state

Additionally, because of the virtual guitar additions to the design, there became a need for extensive computation on the frontend. To address this, when the user begins a new song, the webserver returns a webpage that has a javascript client embedded within it. This client constantly communicates with the guitar over http on behalf of the user to request strumming information during a song. Upon every update from the server, the client can refresh the virtual guitar with current information allowing the physical guitar and the virtual guitar to appear completely synchronized.

2) MIDI Pre-Processing

A MIDI file is organized into 1 header section and at least 1 "Track" section. The header specifies timing information to determine some timing info and the number of track sections that follow. Each track section specifies a tempo, notes, and duration information.

Before forwarding the user's MIDI file to the Teensy, the RPi lightly pre-processes it using the pretty_midi [10] python library so the Teensy is not burdened with parsing through information it does not need. For example, the MIDI Header and Track sections contain byte counts, the instrument's name, and other preamble that the Teensy does not need. So, the RPi can strip that extraneous information out and send an "abridged" MIDI file, so the Teensy only needs to parse the essential tempo, timing, and note information.

Additionally, this parsed MIDI file also provides enough information for the virtual guitar's functionality allowing the reuse of our code for two separate components at the same time, lowering development time and reducing computational load on the system.

3) Teensy Communication

The RPi runs a UART communicator process to establish and maintain a connection between the Teensy and the Pi over a specified port. Its job is to receive user requests from the web server and convert them into interrupt signals, which can then be sent to the teensy and vice-versa. Upon a start request from the user, the web server tells the UART communicator to send a specific file to the microcontroller by first sending a "file_transmission" (Fig. 9) interrupt, followed by all the file data. One idea was to send the file one packet at a time as needed, but this was abandoned due to UART latency concerns and since it would complicate the real-time Teensy firmware. When a fileStop command is issued, the UART communicator interrupts the Teensy by raising its STOP GPIO pin to high.

B. *Teensy/Embedded Subsystem*

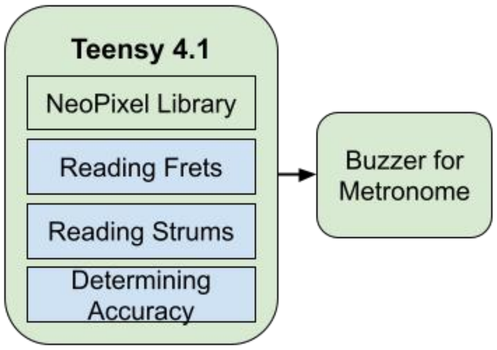


Fig. 8. Zoomed-in crop of block diagram in Appendix Figure I. The Teensy microcontroller is the glue between the User Interface and the electronic hardware. The Teensy's software is structured as a state machine.

1) *State Machine*

A state machine controls the high-level decisions made by the Teensy. There are two classes of inputs to the state machine - interrupts generated by the RPi (shown in purple in Fig. 9), which are based on the user's interaction with the system, and inputs originating from the operation of the system itself (shown in red in Fig. 9).

When the system is first turned on, or "idling," it starts in the "WAIT TO START" state. Once the user selects a song on the web app, the RPi asserts a GPIO pin high, causing a rising edge on the "file_transmission" digital pin of the Teensy. This causes the Teensy to enter the "RECEIVING SONG" state to listen to the RPi over UART for a stream of (fret, string) coordinates. Once the RPi transmits the file, it asserts the same pin low, and the Teensy interprets the falling edge as the end of file transmission.

Having received the MIDI file, the Teensy transitions to the "PARSING SONG" state, where it parses the file. Then, it transitions to the "USER EXPERIENCE" state, where it lights up the LED for the first note of the song and waits for the user to start playing the guitar by strumming. When the first strum is sensed, the Teensy lights up LEDs, reads frets, and continues detecting strums.

Once the user finishes playing the song (the last note is reached), the "WAIT TO START" state is entered again. The Teensy can also enter this initial state if the user restarts the system through the web app.

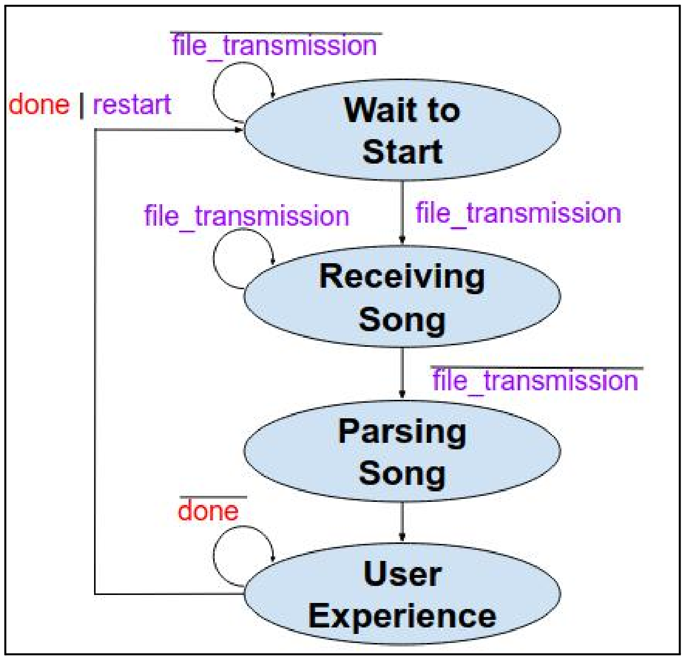


Fig. 9. State Machine for the Teensy's Software

2) *Music information*

The RPI processes the MIDI file and distills it into a few key bits of information for the Teensy: metronome speed and volume, the user experience mode (training or performance) and the note timing & coordinates. This is all the Teensy needs to determine when to light up a particular note's LED and when to expect the user to play that note.

Originally we had the RPI forward the MIDI file to the Teensy, which would reparse. But this was wasteful, and we had trouble getting the RPI and Teensy to parse the file similarly, so we stuck to just 1 parser.

3) *LED Control*

The Teensy stores a "note schedule" indicating when particular notes should be played or released. As the Teensy executes in the USER EXPERIENCE state, it compares the current time to entries in the note schedule to see if it is time for a note to be played or released. Once the particular note is determined from the note schedule, the corresponding LED position is determined by indexing into a static mapping relating notes to LED positions on the fretboard.

C. *Electronic Hardware Subsystem*

The interaction between the Raspberry Pi, the Teensy, the guitar, and the user is provided by a series of hardware components. These consist of sensing components to take in information from the environment, components that provide user feedback, and various power and data interconnects. A block diagram overview of the hardware components is shown in Fig. 10.

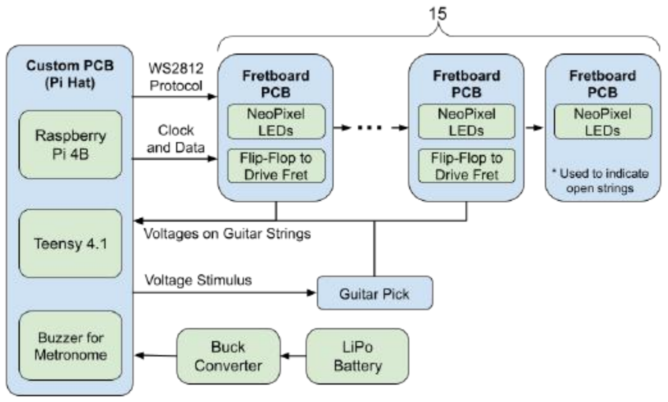


Fig. 10. Zoomed-in crop of block diagram in Appendix Figure I, focusing on the electronic hardware.

1) Strum Detection

The system must determine when the guitar is strummed to know if the user played the desired note correctly. To accomplish this, we originally designed a circuit to take in the guitar's audio signal and output a digital signal indicating when the guitar is strummed.

This system used the piezoelectric sensor integrated into the guitar rather than an external microphone to reduce external interference. This sensor converts the mechanical motion of the guitar strings into a voltage.

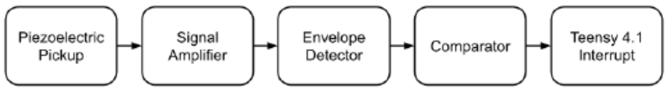


Fig. 11. Block diagram of the strum detection circuitry

The block diagram planned for the strum detection is shown in Fig. 11. The schematic corresponding to this block diagram is shown in Fig. 12

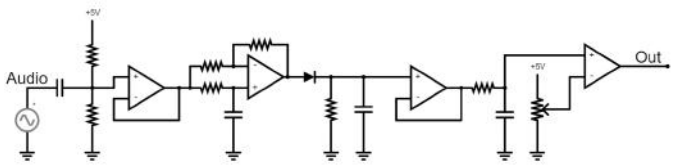


Fig. 12. Physical implementation of the strum detection circuit

This circuit analyzes the audio amplitude and generates a digital output if it is over a set level. Various changes were made to the circuit, such as including a differentiator circuit to look for sharp jumps in audio amplitude, but we ultimately changed approaches. We found that audio-based detection was not reliable enough to meet our accuracy requirements, as it often picked up on external noises or extraneous noises made by the user.

The selected solution was to drive a guitar pick to 3.3V and to then read off the voltage on each string, similar to how the finger placement sensors function. This solution has the added benefit of detecting which string is strummed, without having to Fourier transform the guitar audio. This solution requires a custom metal guitar pick with a wire running to it in order to drive the pick to 3.3V when desired.

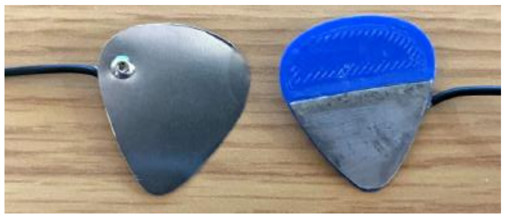


Fig. 13. Guitar picks with metal electrodes. The left pick has an electrode on both sides,, while the right pick only has an electrode on one side. They can be easily interchanged to match the user's preference.

2) Fretboard PCBs

To connect the addressable LEDs and drive each fret to 3.3V individually, our system integrates a fretboard PCB next to each guitar fret. The addressable LEDs require 5V, ground, and a data-in pin. They also have a data-out pin that connects to the data-in of the next LED in the series. There is a 0.1µF capacitor across the power rails next to each LED to ensure proper LED functionality. The LEDs used are SK6812 NeoPixel LEDs, which support write speeds of up to 800kHz. For 60 LEDs, this corresponds to around 2ms to write to all the LEDs. Each fretboard PCB has a D-flip-flop, forming one large shift register across all the fretboard PCBs. The output of a D-flip-flop is connected to the adjacent fret of the guitar. While we originally planned on using a direct connection between each flip-flop, we found that the propagation time of the signal between adjacent frets caused timing violations, so a low-pass filter was added on the data lines to allow the clock signal to arrive at the next flip-flop first.

Using the Teensy, a logical high can be clocked into the first fretboard PCB, which can be shifted to the next PCB, allowing each fret to be driven high one at a time. A 3.3kΩ resistor and forward-biased diode connect the D-Flip-Flop and a fret to limit the current that could flow to 1mA. While a fret is driven high, the voltage on each guitar string is read, allowing the Teensy to determine which strings were contacting the fret being driven high. The diode prevents any issues relating to one fretboard PCB trying to drive a string low while another drives it high. Using a diode, the string would be driven high in this case.

The fretboard PCB design is shown in Fig. 14. The top half of the board contains the 4 addressable LEDs, D1-D4, and the bottom half contains the D-flip-flop and supporting passive components. The pads on the right side of the board and the bottom left of the board enable the boards to be daisy-chained together, which reduces wiring complexity. Since the string spacing of a guitar changes slightly between each fret, we created two different sizes of PCB. While 14 different sizes would be optimal to ensure the LEDs line up perfectly with the strings at low production quantities, this drastically increases the price and complexity of assembly. As such, we chose to use two different sizes of fretboard PCBs. 8 of the boards have LEDs that are slightly closer together and are used for the open string indication and for the frets at the end of the guitar. 7 of the boards have a wider LED spacing and are used for the frets closer to the guitar's body.

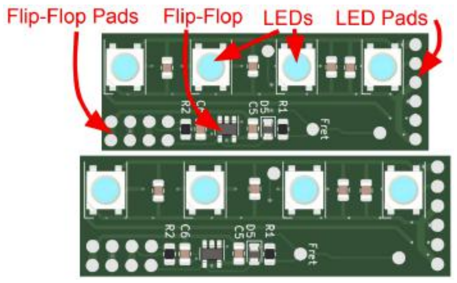


Fig. 14. The small fretboard PCB (top) and large fretboard PCB (bottom).

To prevent users from being able to touch the metal pads on the fretboard PCBs, and to prevent the fretboard from having an uncomfortable feel, we created covers for each of the fretboard PCBs.

3) Pi Hat PCB

The RPi and Teensy require numerous connections for UART and interrupts, external power, and various input and output devices. To implement these connections, a Pi "Hat" is used. A Pi-Hat is a PCB that plugs directly into the 40-pin header on the RPi, as shown on the right side of the board in Fig. 15.

The Pi-Hat filters any noise in the 5V power supply connected to the Hat via a barrel jack and distributes this to the Teensy, RPi, and fretboard PCBs. The Pi-Hat also connects the Teensy and RPi with 10 I/O lines and a UART channel so they can communicate.

The Pi-Hat has a number of I/O ports on the left side that are connected to the fretboard PCBs, LEDs, and strings. For the LEDs, a logic level converter is used to convert the 3.3V signal from the Teensy to a 5V signal for the LEDs.

An active buzzer acts as a metronome by beeping in short pulses to indicate the target tempo to the user while they are playing. Active buzzers can be driven by simply pulling an output pin on the Teensy, either high or low, making this design for the metronome simple to implement on the firmware side.

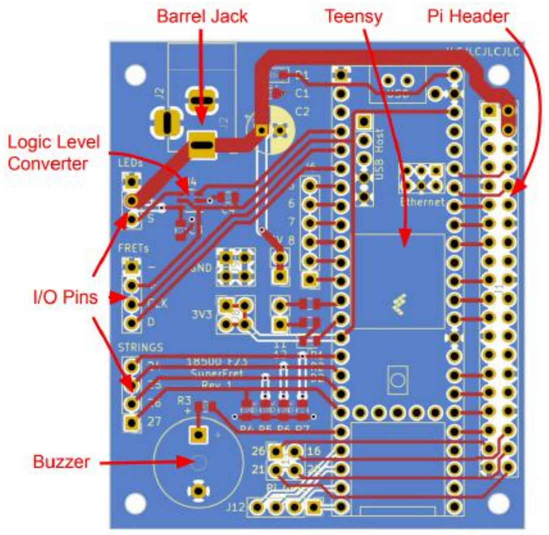


Fig. 15. Pi-Hat PCB layout

4) Power Supply

The system is powered using a 5V DC wall adapter, which connects to the Pi Hat using a 5mm barrel jack connector. The total expected current draw is 2.0A for the Pi, 0.15A for the Teensy and flip-flops, and 1.5A for the LEDs at half brightness. This sums to 3.65A, so a 5A power supply was chosen for the project. For user convenience, the system can also be powered by a battery. An 11.4V 2200mAh lithium polymer battery is connected to a 5A 5V buck converter, which then connects to the 5V power rail of the Pi-Hat PCB

VII. Testing, Verification, and Validation

To validate our solution and design requirements, comprehensive tests were conducted. The aim was to scrutinize the real-time responsiveness of the SuperFret system and its ability to handle diverse playing conditions. Refer to Table I for a summary of the test results.

A. *Results for Latency*

An oscilloscope was utilized to precisely measure the time delay between initiating a strumming action and the corresponding LEDs being written to. After multiple measurements, we determined the latency to be 1.85ms which was well under our target. This test held critical significance as it directly addressed the design requirement of achieving a latency of less than 50 milliseconds, ensuring that the system provides instantaneous feedback to the user to prevent the guitar from feeling sluggish during practice sessions.

Simultaneously, the web app's network delay test evaluated the responsiveness of the web application and the ability of the virtual guitar to stay up-to-date with the physical guitar. The test involved high frame rate video (240 frames per second) to capture the delay through the entire system from the user strum to the website updating. After rewatching 12 iterations of the test, we aggregated the data to get an average total delay of 215ms which was under our target of 250ms. This result ensures that users experience a smooth and responsive interface when interacting with the web application, aligning with the design specifications and user expectations.

B. *Results for Accuracy*

For accuracy testing, a strum identification test was designed to assess the system's ability to identify strums accurately. We quantified the system's ability to correctly identify strums by performing 200 1/8th note strums at up to 300 BPM on each string. The success criteria for this test were determined by calculating the percentage of correctly identified strums, directly addressing the accuracy requirements outlined in the design specifications. At 300 BPM, we achieved 99% accuracy for this test, clearly indicating the system's competency in identifying and responding to strumming actions.

In addition to strum identification, a finger placement test was conducted to evaluate the system's accuracy in detecting the placement of fingers on different string and fret positions. This involved systematically placing a finger on each fret and string location and using code to light up the LED under the pressed location. We verified that the LED came on under each finger position. We then repeated this process multiple times and calculated the percentage accuracy, which quantified the system's precision in detecting finger placement. Using this method, we achieved 100% finger placement accuracy, since under no conditions did the system not register a finger press or misidentify a press. This test directly validated the accuracy requirements for finger placement detection as specified in the design specifications. Overall, these tests were crucial in ensuring that the SuperFret system not only met the theoretical design trade-offs but also demonstrated robust performance aligned with the specific use-case requirements for the project.

Accuracy for the LEDs was verified by displaying patterns on the fretboard to ensure all LEDs were indexed properly. We then loaded a variety of MIDI files into the system via the RPi and played them on the guitar. We used a tuning tool to verify that the system always instructed the user to play the correct note as indicated in the MIDI file. One important note is that the system may shift notes up or down an octave if the provided MIDI file falls outside the guitar's range.

C. Safety

As per IEC TS 60479-1, humans can not perceive currents below 500μA, and currents below 1mA do not impact muscles [3]. We used a lab bench ammeter capable of measuring down to 0.01μA to verify this. Under normal conditions, participants would contact the 3.3V guitar string with 1 hand and a ground signal with the other. A 10kΩ potentiometer would be between the 3.3V source and the string, and the potentiometer would initially start at 10kΩ. While monitoring the current, the potentiometer's resistance was turned to 0Ω, and the current was recorded. If the current ever reached 1mA while lowering the potentiometer resistance, the test would be stopped. We found that the current passing through a user was at most 0.80μA. To test the maximum current the strings carry, we used the ammeter to connect the fret to a string and verify that no more than 1mA flows. Our results showed that the maximum average current through a short circuit in the system was only 5.37 μA. This is an average since each fret is driven high for only 10μs out of a loop time of 1555μs. Using an oscilloscope and a series shunt resistor, we found the current spiked to 0.762mA for 10μs, which is below our required value.

D. User Experience

For user experience evaluation, subjective tests were conducted to gather feedback on the web application and hardware components. The questions assessed the users' perception of the system's usability and effectiveness. Refer to Appendix Table II for the specific questions asked and a summary of the results.

Users were asked to interact with the web application and provide ratings on a scale of 1 to 10 for categories such as the intuitiveness of the interface, readability of statistics, and responsiveness of the virtual guitar. These subjective evaluations were averaged to create a quantitative metric for the overall user experience with the web application. For example, a user-friendly interface is crucial to the system's success, as it directly impacts the accessibility and satisfaction of the users.

Similarly, users were requested to evaluate the hardware components, considering factors like comfortability, LEDs' effectiveness, and the metronome's volume and pitch. Ratings on a scale of 1 to 10 for each category were averaged to provide a quantitative measure of the overall user satisfaction with the physical components. Comfortability is vital for sustained practice sessions, while the effectiveness of LEDs and the metronome directly impact the user's ability to follow guidance and maintain rhythm during practice.

The system's success in meeting the user-centric design goals was quantified by aggregating the user ratings. The results of our test showed that the physical components were very well built and that the guitar was very responsive and effective at showing users where to play. However, it also showed there was room for improvement in the look of the website. These user experience evaluations were essential for obtaining qualitative and quantitative insights into the effectiveness and user-friendliness of the SuperFret system. The feedback gathered from users was invaluable in making iterative improvements to enhance the overall user experience, ensuring that the SuperFret system fulfilled the technical specifications and was well-received by its target audience of beginner guitar players.

VIII. Project Management

A. Schedule

The Gantt chart in Appendix Figure II shows the project timeline for the semester. The tasks are divided into Electrical, Firmware, and Software, with Owen, Tushaar, and Ashwin leading these categories. Scheduled weekly 2-hour meetings between team members occur to perform integration between systems and discuss design considerations to prevent integration issues at the end of the semester. Time was provided at the end of the semester for the final integration of the systems, and team-wide tasks such as working on presentations and reports are also listed. Highlighted bars indicate progress on the listed task. One change to the schedule was the addition of the virtual guitar since our team decided to put more emphasis on its completion. Additionally, we experienced some schedule delays due to previously mentioned issues with the flip-flops. This resulted in pushing back the ordering of both the fretboard and Pi-Hat PCBs and delayed Tushaar's ability to interface with the fretboard sensors. Mechanical modifications to the guitar took longer than expected as well, but this was compensated for by the assembly and testing of the PCBs being ahead of schedule

B. Team Member Responsibilities

As shown in the schedule, the work was divided into 4 main areas - overall project management, web app, firmware, and electronics. All members were responsible for staying up to date on the overall project timeline and keeping the timeline for their area on track.

Ashwin focused on the web app and wrote software on the RPi to host it. He also wrote software to send MIDI files to the Teensy and receive statistics on how the user is doing from Teensy.

Owen designed the electronic hardware, which involved the PCBs on the fretboard, the strum detection circuitry, and the interface board that allowed signals to pass between the Teensy and RPi.

Tushaar focused on the firmware, the glue between Ashwin and Owen's areas. This involved writing the Teensy's software for interfacing with the RPi and the electronic hardware that Owen designed.

C. Bill of Materials and Budget

The total bill of materials used on this project totals to $168.00. Appendix Table III shows the full breakdown of the parts used on the system as well as some of the unused parts. We acquired some of the more expensive components, such as the RPi 4B and Teensy 4.1 from the ECE department and Roboclub. The primary changes from the design report are a more detailed breakdown of the components on the Pi-Hat PCB, specific pricing for the revised fretboard PCBs, and the addition of a LiPo battery and a buck converter for power. Due to our change to a metal pick, we no longer needed the microphone listed in the design report Appendix Table IV indicates the reasons why components were not used. Additionally, excess parts were ordered for some components as spares and to hit Digikey price breaks. For example, excess flip-flops were ordered since at one point we believed the issues with the flip-flops may have been ESD damage or heat damage caused when soldering.

D. Risk Management

Several critical risks were identified when planning the project, each requiring careful consideration and mitigation strategies to ensure a smooth design implementation.

One risk involved detecting which string the user strummed. This is necessary for determining if the user put their fingers in the correct position but strummed wrong. While we initially planned on using audio amplitude to detect strums, we realized that this would be insufficient for detecting which string was played. While from the beginning we knew an electrode based guitar pick would solve the issues, we hoped to resolve this issue without using a pick, as this makes the system more intrusive and limits how users can play. We experimented with capacitive touch sensing on the strings, but this method turned out to be too slow and could not function alongside the finger placement sensing. We also investigated inductive guitar pickups used on many electric guitars. However, these systems combine the 4-6 strings of the guitar into a single output, meaning that we would not be able to detect which string is strummed. Finally, we looked at performing an FFT of the guitar audio on the Teensy. However, this proved to be too slow to meet our desired latency requirements. We ultimately ended up switching to our fall-back plan of an electrode-pick. This did have the added benefit of removing the need for audio based strum detection, which increased the accuracy of the system. The Pi-Hat was designed with multiple free digital and analog I/O pins however in case we found a sensor that could be used to indicate which string was played.

The ambiguity in fret-string contact due to multiple ways to play the same note posed another risk. Generally, we want notes around the same time to be played around the same fret. This prevents users from needing to move their arms rapidly up and down the guitar. To address this, we developed an algorithm to determine which alternative of the same note is most appropriate to play. The algorithm would take in multiple parameters such as the note value, the previous note value, and the time since the previous note was played and compute a *(fret, string)* tuple output which specifies to the system where the next note will be played. However, we noticed the algorithm was imperfect and generated valid but not necessarily optimal outputs. So throughout the development cycle, we adjusted the algorithm when we noticed discrepancies by adding more edge-case handling and tiny tweaks that helped improve the fingering on the guitar. The constant maintenance of the algorithm helped translate the MIDI file notes and guitar notes much more simply and mitigate the component's risk.

A final risk we had to manage while working on the project was the D-flip-flops. In our original design, we directly connected the output of each flip-flop to the input of the next flip-flop, as is typically done when creating a shift register. However, during our testing, we discovered that on a single clock edge, a signal could pass through up to 3 flip-flops. We eventually discovered that this issue was caused by the propagation time of the signals in the wires since we had used high-speed flip-flops. The oscilloscope we used while debugging could not handle sub-nanosecond time steps, making this issue difficult to locate. Although we eventually located and resolved the issue, we formulated a backup plan and instituted a date at which we would switch to the backup plan. The plan was to run an individual wire to each fret instead of using the flip-flops. Our final Pi-Hat PCB was ordered before this issue was resolved, and as such, you can see the numerous Teensy I/O that were broken out to pads in case we switched to this plan.

IX. Ethical Issues

Although this product aims to help beginners learn to play the guitar, this is also the population most susceptible to problems arising from the misuse or failure of the project. In terms of user safety, there is a very small risk of a shock. While the system does have the user touching 3.3V, this is through a current limiting resistor and poses no risk, as explored previously. The only risk associated with the project is the improper use of the wall power adapter, which could introduce a safety hazard if the user improperly uses the adapter or if the adapter fails. Besides the power supply, a hardware failure of the system would not introduce any safety risks but would rather hinder the training effectiveness of the project.

The user is also subject to the risks when using the web app hosted on the Pi. While highly unlikely, it could be imagined that if a malicious individual could gain access to the Pi and manipulate the web app, they could acquire some information from the user's computer, such as browser session tokens and cookies. In terms of misapplication, the Pi connects to the internet. If the default Pi password is not properly changed, the Pi can be hacked relatively easily and used to perform some damage to the network it is connected to, depending on the network's security.

When it comes to data, our project uses MIDI files that we acquire from the internet. Ideally, these files would not contain malicious information that would cause our system to become a security threat. A thorough analysis of our system's security risks would be prudent.

X. Related Work

Fret Zealot [8] is an existing product that is similar to ours. It is a guitar learning tool hosted on a website with features such as song tutorial videos and online guitar courses. They also sell a set of guitar LEDs that allow users to learn chords and songs, similar to our project.

However, this product lacks finger placement and strum detection on the guitar and relies on a microphone. Thus, the guitar cannot provide feedback regarding whether notes were played correctly, rapidly, and accurately. Our product also separates itself by collecting this data and displaying dynamic songs moving at the user's pace. It also displays the timing and accuracy information to the user, allowing them to observe their skills increase over time. However, Fret Zealot's approach to guitar learning offers them distinct advantages. The most prominent is that their LEDs are detachable, which allows users to pick their own guitar for learning instead of us deciding. Overall, our solution offers a more interactive experience for the user.

XI. Summary

The SuperFret project aimed to develop a system to assist beginner guitar players in improving their skills and playing basic songs. Learning new songs, practicing tempo, and drilling finger exercises were made simple through our interactive design, according to over 15 test users. Thus, we met our design requirements.

The system's user-friendly interface, real-time feedback through LEDs, and guidance enhance the learning experience. The web application allows users to upload their favorite songs for practice, promoting an enjoyable and tailored learning journey. The system's ability to handle notes down to 1/8th at 100 BPM and accurately identify finger placement and strumming with a 99% accuracy rate ensures a supportive and effective practice environment.

However, the design of both our hardware and software components limits the use of the guitar to single sequential notes. This means that playing chords and sustaining notes is not supported under our current architecture. Additionally, communication between the website and the physical guitar is severely unoptimized for low latency as it uses a simple http protocol. This creates a small yet noticeable 215ms latency between the virtual guitar and the physical one. Although the Superfret system is a powerful tool for learning the guitar, it could still benefit from more development, given these limits.

A. Future Work

Many more exciting enhancements could extend our project's use cases if given more time. Our team particularly wanted to create an additional mode for displaying scales on the guitar. This mode would display all the notes corresponding to a scale (a subset of the possible 12 notes) while allowing the user to practice only playing notes on a scale. We think this could be a very effective tool for learning the patterns that scales create on the fretboard. Additionally, since our system already has finger sensing and strum detection, we could implement a mode where a user can play whatever they want on the guitar, and the resulting song will be automatically recorded and stored in midi format for the user to playback and edit. This could promote creativity in users and entice them to explore the guitar further.

B. Lessons Learned

Throughout the development of this project, our team came across some challenges that eventually turned into learning experiences for us. The first challenge was carving out channels across the wooden neck of the guitar to install our PCB boards flush against the surface of the fretboard. We initially did not plan for this to take long, but the process of hand-grinding the channels took a day's worth of time. We underestimated the level of effort required to work with wood which set us back in our schedule. For future projects that deal with handling and modifying wood materials, we recommend having a thorough plan in place with sufficient time in your schedule dedicated to woodworking. It will take a lot of time to complete, so one must ensure it is accounted for in their team schedules.

Another learning lesson came when trying to implement synchronization between the virtual guitar and the physical guitar. Upon the addition of the virtual guitar, it was obvious that it had to remain synchronized at all times with the physical guitar. The complexity of our solution quickly blew up as the front-end simulator had to request an update from the RPi, which requested an update from the Teensy microcontroller and then sent a response back. We suddenly had to keep track of the state of three different system environments, which was proving hard to do. We settled with a slightly messy information pipeline that could have been more optimized. With 20/20 vision, the lesson that we learned from this was just how different the programming for distributed computing is. A more thoughtful approach from a distributed systems philosophy would have helped streamline the communication protocols and make the code cleaner and easier to reason with. For future teams that require distributed computing for their project, we highly recommend researching appropriate communication protocols that serve their projects' needs instead of trying to develop their own (for us, Remote Procedure Calls would have worked very well). A good framework could even abstract out a lot of the complexity of this problem.

Glossary of Acronyms

BPM – Beats per Minute
COTS – Commercial Off-The-Shelf
GPIO – General Purpose Input Output
I/O – Input and Output
MIDI – Musical Instrument Digital Interface
PCB – Printed Circuit Board
RPi – Raspberry Pi
SBC – Single Board Computer

References

[1] "Live online guitar lessons: Learn guitar online," Lesson With You, https://lessonwithyou.com/guitar-lessons/ (accessed Oct. 13, 2023).
[2] "Acoustic bass guitar stock clipart: Royalty-free," Freeimages, https://www.freeimages.com/premium-clipart/acoustic-bass-guitar-4992596?ref=clipartlogo (accessed Sep. 28, 2023).
[3] "IEC TS 60479-1" International Electrotechnical Commission. (2018). IEC 60479-1:2018 Effects of current on human beings and livestock (accessed Sep. 23, 2023)
[4] P. Stoffregen. "Teensy® 4.1 Development Board." https://www.pjrc.com/store/teensy41.html (accessed Sept. 28, 2023)
[5] P. Stoffregen. "CoreMark - CPU Performance Benchmark." https://github.com/PaulStoffregen/CoreMark#coremark---cpu-performance-benchmark (accessed Sept. 28, 2023)
[6] "Arduino Memory Guide" https://docs.arduino.cc/learn/programming/memory-guide (accessed Oct. 10, 2023)
[7] A. Matthies, "Guitar neck shapes &amp; fretboard radius explained," Guitar Gear Finder, https://guitargearfinder.com/guides/guitar-neck-shapes/ (accessed Oct. 12, 2023).
[8] "Best way to learn guitar: How to learn guitar at home," Fret Zealot, https://www.fretzealot.com/ (accessed Oct. 13, 2023).
[9] "Model View Controller", Open Genuis IQ, https://iq.opengenus.org/model-view-controller-django/ (accessed Dec. 15 2023)
[10] "pretty_midi", Craffel, https://craffel.github.io/pretty-midi/ (accessed Dec. 15 2023)

APPENDIX

Figure I: System Block Diagram

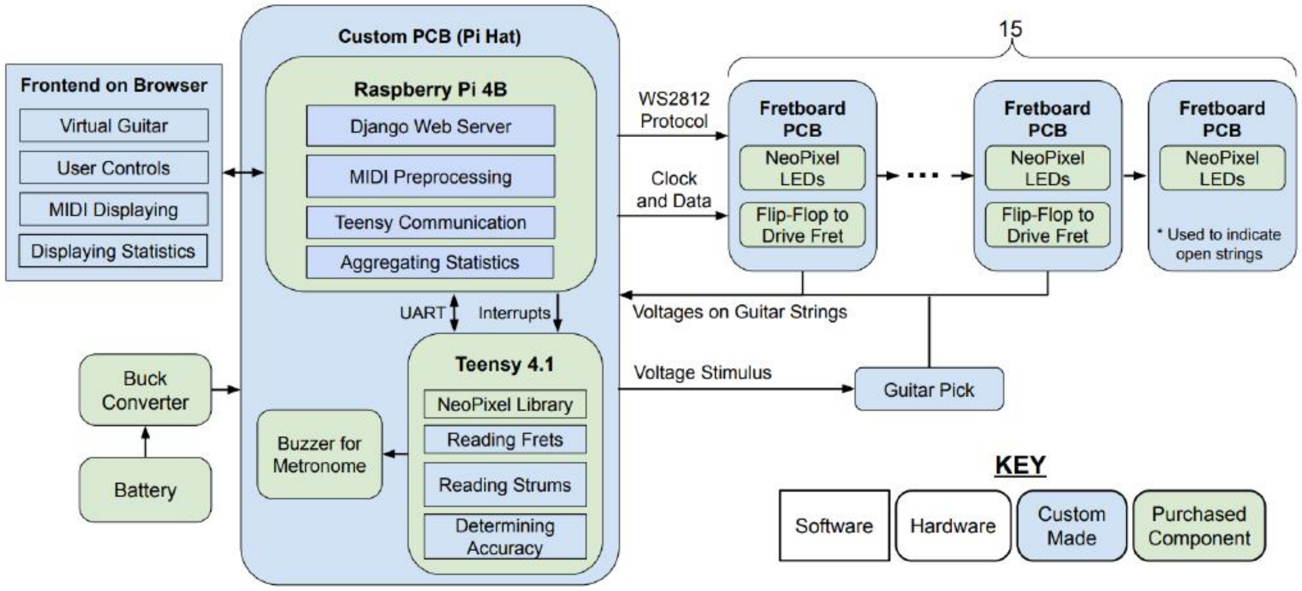


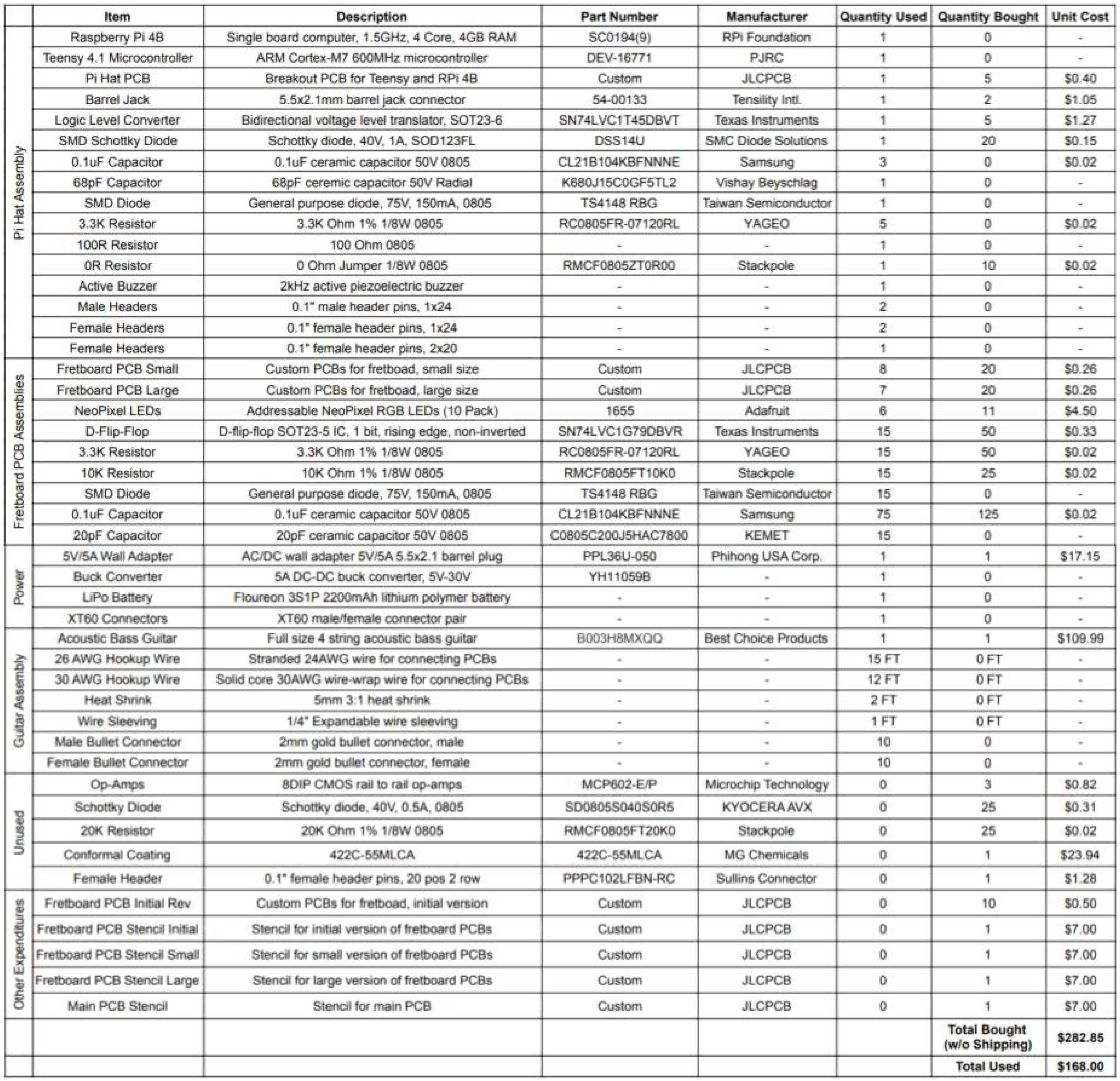
Table I: Test Results

Metric	Target	Actual
MIDI to fretboard LED conversion accuracy	100%	**100%**
Finger placement detection accuracy	≥99%	**100%**
Strums per minute supported	≥200	**300**
Strum detection accuracy	≥99%	**99%**
Latency from strum to LEDs updating in response	≤50ms	**1.85ms**
Latency from strum to web app updating in response	≤250ms	**215ms**
Average current through body possible	≤1mA	**5.37μA**
Total system current with all LEDs at ½ brightness	<4.5A	**0.96A**

Table II: User Survey Results

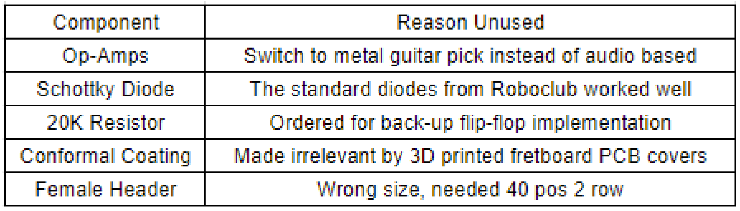
Question	Average Rating (1-10)	Sample Size
How intuitive is the web app interface?	9.3	8
How responsive is the web app?	10.0	8
How effective if the on-screen guitar at guiding you?	9.0	4
How aesthetically appealing is the web app?	7.0	4
How effective are the LEDs at indicating finger placement?	10.0	8
How non-intrusive are the guitar modifications?	9.9	8
How responsive is the guitar hardware?	9.5	8
How effective is the system at teaching you to play basic songs?	9.5	4
How would you rate the overall system experience?	9.8	8

Table III: Bill of Materials



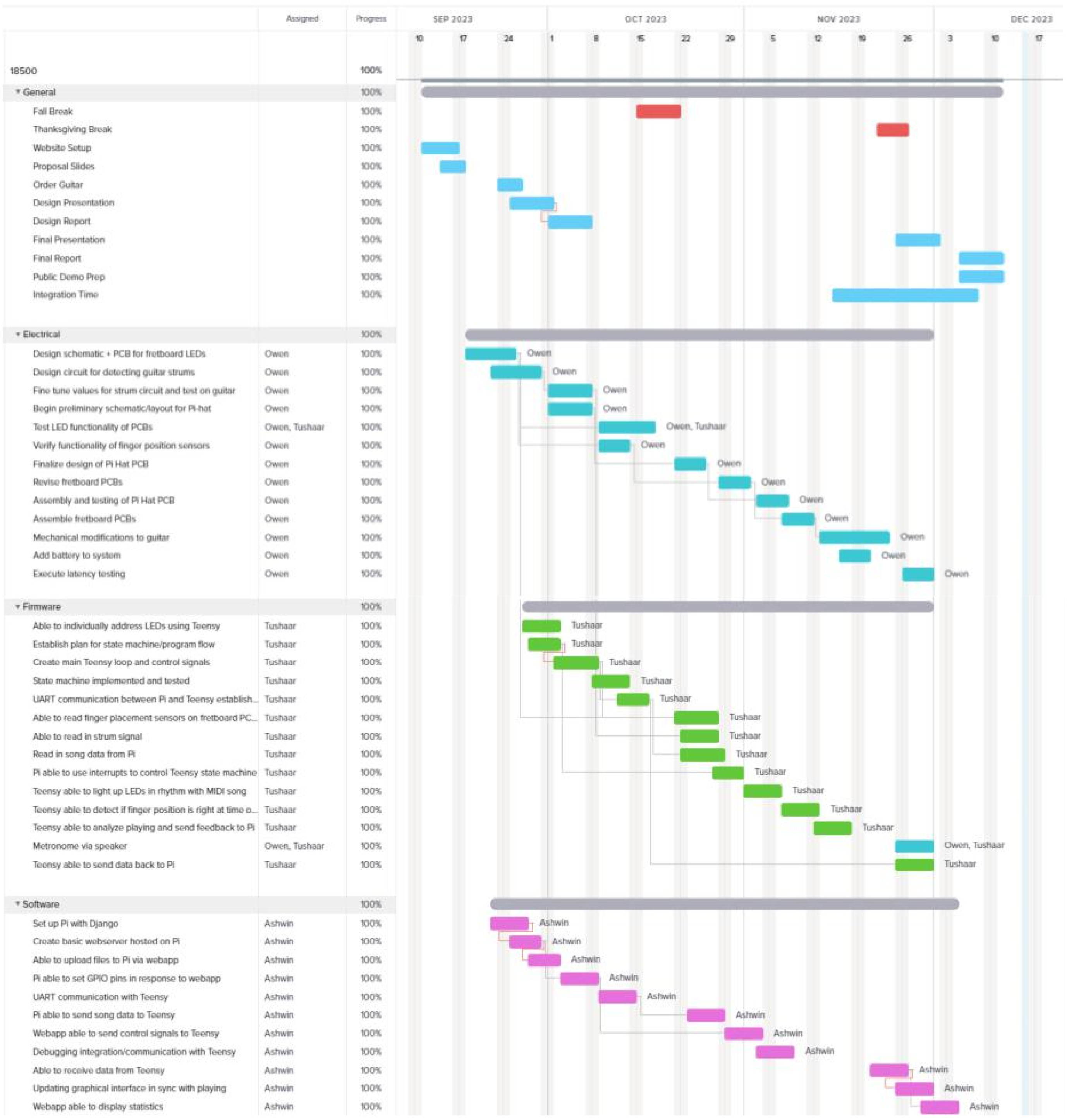
	Item	Description	Part Number	Manufacturer	Quantity Used	Quantity Bought	Unit Cost
Pi Hat Assembly	Raspberry Pi 4B	Single board computer, 1.5GHz, 4 Core, 4GB RAM	SC0194(9)	RPi Foundation	1	0	-
	Teensy 4.1 Microcontroller	ARM Cortex-M7 600MHz microcontroller	DEV-16771	PJRC	1	0	-
	Pi Hat PCB	Breakout PCB for Teensy and RPi 4B	Custom	JLCPCB	1	5	$0.40
	Barrel Jack	5.5x2.1mm barrel jack connector	54-00133	Tensility Intl.	1	2	$1.05
	Logic Level Converter	Bidirectional voltage level translator, SOT23-6	SN74LVC1T45DBVT	Texas Instruments	1	5	$1.27
	SMD Schottky Diode	Schottky diode, 40V, 1A, SOD123FL	DSS14U	SMC Diode Solutions	1	20	$0.15
	0.1uF Capacitor	0.1uF ceramic capacitor 50V 0805	CL21B104KBFNNNE	Samsung	3	0	$0.02
	68pF Capacitor	68pF ceremic capacitor 50V Radial	K680J15C0GF5TL2	Vishay Beyschlag	1	0	-
	SMD Diode	General purpose diode, 75V, 150mA, 0805	TS4148 RBG	Taiwan Semiconductor	1	0	-
	3.3K Resistor	3.3K Ohm 1% 1/8W 0805	RC0805FR-07120RL	YAGEO	5	0	$0.02
	100R Resistor	100 Ohm 0805	-	-	1	0	-
	0R Resistor	0 Ohm Jumper 1/8W 0805	RMCF0805ZT0R00	Stackpole	1	10	$0.02
	Active Buzzer	2kHz active piezoelectric buzzer	-	-	1	0	-
	Male Headers	0.1" male header pins, 1x24	-	-	2	0	-
	Female Headers	0.1" female header pins, 1x24	-	-	2	0	-
	Female Headers	0.1" female header pins, 2x20	-	-	1	0	-
Fretboard PCB Assemblies	Fretboard PCB Small	Custom PCBs for fretboad, small size	Custom	JLCPCB	8	20	$0.26
	Fretboard PCB Large	Custom PCBs for fretboad, large size	Custom	JLCPCB	7	20	$0.26
	NeoPixel LEDs	Addressable NeoPixel RGB LEDs (10 Pack)	1655	Adafruit	6	11	$4.50
	D-Flip-Flop	D-flip-flop SOT23-5 IC, 1 bit, rising edge, non-inverted	SN74LVC1G79DBVR	Texas Instruments	15	50	$0.33
	3.3K Resistor	3.3K Ohm 1% 1/8W 0805	RC0805FR-07120RL	YAGEO	15	50	$0.02
	10K Resistor	10K Ohm 1% 1/8W 0805	RMCF0805FT10K0	Stackpole	15	25	$0.02
	SMD Diode	General purpose diode, 75V, 150mA, 0805	TS4148 RBG	Taiwan Semiconductor	15	0	-
	0.1uF Capacitor	0.1uF ceramic capacitor 50V 0805	CL21B104KBFNNNE	Samsung	75	125	$0.02
	20pF Capacitor	20pF ceramic capacitor 50V 0805	C0805C200J5HAC7800	KEMET	15	0	-
Power	5V/5A Wall Adapter	AC/DC wall adapter 5V/5A 5.5x2.1 barrel plug	PPL36U-050	Phihong USA Corp.	1	1	$17.15
	Buck Converter	5A DC-DC buck converter, 5V-30V	YH11059B	-	1	0	-
	LiPo Battery	Floureon 3S1P 2200mAh lithium polymer battery	-	-	1	0	-
	XT60 Connectors	XT60 male/female connector pair	-	-	1	0	-
Guitar Assembly	Acoustic Bass Guitar	Full size 4 string acoustic bass guitar	B003H8MXQQ	Best Choice Products	1	1	$109.99
	26 AWG Hookup Wire	Stranded 24AWG wire for connecting PCBs	-	-	15 FT	0 FT	-
	30 AWG Hookup Wire	Solid core 30AWG wire-wrap wire for connecting PCBs	-	-	12 FT	0 FT	-
	Heat Shrink	5mm 3:1 heat shrink	-	-	2 FT	0 FT	-
	Wire Sleeving	1/4" Expandable wire sleeving	-	-	1 FT	0 FT	-
	Male Bullet Connector	2mm gold bullet connector, male	-	-	10	0	-
	Female Bullet Connector	2mm gold bullet connector, female	-	-	10	0	-
Unused	Op-Amps	8DIP CMOS rail to rail op-amps	MCP602-E/P	Microchip Technology	0	3	$0.82
	Schottky Diode	Schottky diode, 40V, 0.5A, 0805	SD0805S040S0R5	KYOCERA AVX	0	25	$0.31
	20K Resistor	20K Ohm 1% 1/8W 0805	RMCF0805FT20K0	Stackpole	0	25	$0.02
	Conformal Coating	422C-55MLCA	422C-55MLCA	MG Chemicals	0	1	$23.94
	Female Header	0.1" female header pins, 20 pos 2 row	PPPC102LFBN-RC	Sullins Connector	0	1	$1.28
Other Expenditures	Fretboard PCB Initial Rev	Custom PCBs for fretboad, initial version	Custom	JLCPCB	0	10	$0.50
	Fretboard PCB Stencil Initial	Stencil for initial version of fretboard PCBs	Custom	JLCPCB	0	1	$7.00
	Fretboard PCB Stencil Small	Stencil for small version of fretboard PCBs	Custom	JLCPCB	0	1	$7.00
	Fretboard PCB Stencil Large	Stencil for large version of fretboard PCBs	Custom	JLCPCB	0	1	$7.00
	Main PCB Stencil	Stencil for main PCB	Custom	JLCPCB	0	1	$7.00
						Total Bought (w/o Shipping)	**$282.85**
						Total Used	**$168.00**

Table IV: Unused Components



Component	Reason Unused
Op-Amps	Switch to metal guitar pick instead of audio based
Schottky Diode	The standard diodes from Roboclub worked well
20K Resistor	Ordered for back-up flip-flop implementation
Conformal Coating	Made irrelevant by 3D printed fretboard PCB covers
Female Header	Wrong size, needed 40 pos 2 row

Figure II: Gantt Chart



	Assigned	Progress
18500		100%
▾ General		100%
Fall Break		100%
Thanksgiving Break		100%
Website Setup		100%
Proposal Slides		100%
Order Guitar		100%
Design Presentation		100%
Design Report		100%
Final Presentation		100%
Final Report		100%
Public Demo Prep		100%
Integration Time		100%
▾ Electrical		100%
Design schematic + PCB for fretboard LEDs	Owen	100%
Design circuit for detecting guitar strums	Owen	100%
Fine tune values for strum circuit and test on guitar	Owen	100%
Begin preliminary schematic/layout for Pi-hat	Owen	100%
Test LED functionality of PCBs	Owen, Tushaar	100%
Verify functionality of finger position sensors	Owen	100%
Finalize design of Pi Hat PCB	Owen	100%
Revise fretboard PCBs	Owen	100%
Assembly and testing of Pi Hat PCB	Owen	100%
Assemble fretboard PCBs	Owen	100%
Mechanical modifications to guitar	Owen	100%
Add battery to system	Owen	100%
Execute latency testing	Owen	100%
▾ Firmware		100%
Able to individually address LEDs using Teensy	Tushaar	100%
Establish plan for state machine/program flow	Tushaar	100%
Create main Teensy loop and control signals	Tushaar	100%
State machine implemented and tested	Tushaar	100%
UART communication between Pi and Teensy establish...	Tushaar	100%
Able to read finger placement sensors on fretboard PC...	Tushaar	100%
Able to read in strum signal	Tushaar	100%
Read in song data from Pi	Tushaar	100%
Pi able to use interrupts to control Teensy state machine	Tushaar	100%
Teensy able to light up LEDs in rhythm with MIDI song	Tushaar	100%
Teensy able to detect if finger position is right at time o...	Tushaar	100%
Teensy able to analyze playing and send feedback to Pi	Tushaar	100%
Metronome via speaker	Owen, Tushaar	100%
Teensy able to send data back to Pi	Tushaar	100%
▾ Software		100%
Set up Pi with Django	Ashwin	100%
Create basic webserver hosted on Pi	Ashwin	100%
Able to upload files to Pi via webapp	Ashwin	100%
Pi able to set GPIO pins in response to webapp	Ashwin	100%
UART communication with Teensy	Ashwin	100%
Pi able to send song data to Teensy	Ashwin	100%
Webapp able to send control signals to Teensy	Ashwin	100%
Debugging integration/communication with Teensy	Ashwin	100%
Able to receive data from Teensy	Ashwin	100%
Updating graphical interface in sync with playing	Ashwin	100%
Webapp able to display statistics	Ashwin	100%