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 will show users exactly where to place their fingers on the guitar. The system will also detect 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 will allow users to learn the guitar more rapidly and engagingly.**

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

I. Introduction

This project 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 the fretboard makes sense. 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. When learning guitar, 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 will enable 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 will also detect the position of the user's fingers and when they strum, allowing the user to receive real-time feedback to ensure they are playing the correct notes and strumming at the right time. A web app will display 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. They also require quiet environments to analyze the user's playing and cannot provide feedback until the user strums. 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 also require quiet environments to analyze the user's finger location. The SuperFret system will directly detect the user's finger locations, thus enabling a more accurate analysis of the user's playing.

Overall, the SuperFret system should allow beginner guitar players to quickly learn to play notes and basic songs. The system will determine if the user is playing correctly and provide 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 various other training activities, such as playing finger exercises and scales.

The system shall handle notes down to $1/8^{th}$ 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 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 will likely be far lower than the number of mistakes made by the user, so this accuracy will be sufficient for the system.

III. Architecture and Principle of Operation

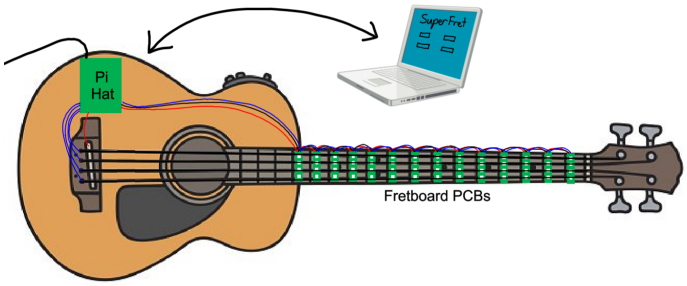

Fig. 1. A depiction of the physical system. Guitar image from [2]

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 scales 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 progress are aggregated and displayed on the web app.

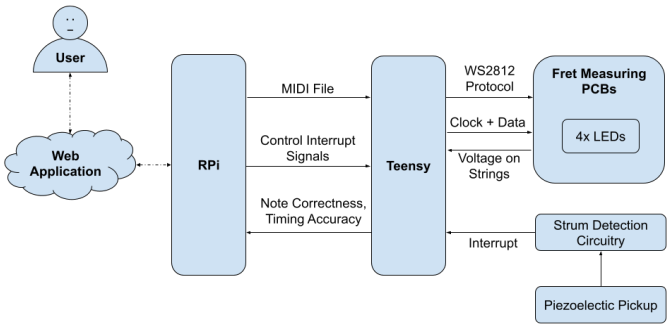

Fig. 2. High-level Architecture Block Diagram.

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 scales 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 passed from the web app to the Teensy, which uses the file as a reference to guide the user to play the correct notes. LEDs on the fretboard guide finger placement, and the Teensy lights them according to the target note it parses from the MIDI file. The LEDs reside on Printed Circuit Boards (PCBs) along the fretboard. The PCBs also contain circuitry for determining which note the user has fingered on the fretboard. Other electronic hardware on the guitar includes a piezoelectric sensor 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 the database into one Model-View-Controller design pattern (Fig. 3) to create web endpoints that the user can access via a web browser. There are 3 communication "streams" between the RPi and the Teensy to relay information from the user to the Teensy: bidirectional communication with the Teensy over UART accounts for 2 streams, and the third is for interrupt signals that control the state machine (Fig. 9) inside the Teensy.

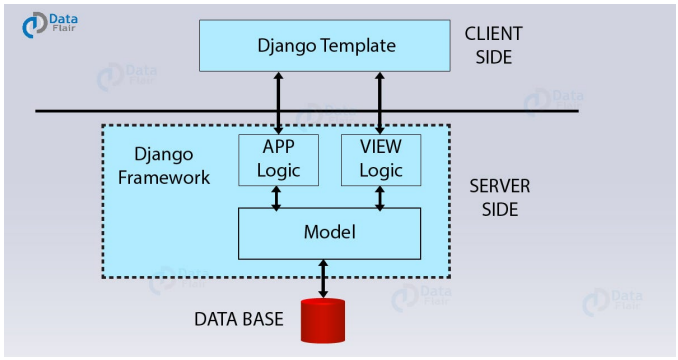

Fig. 3. Django Web Frame Work Implements Model-View-Controller

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 for creating 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 monitoring an interrupt produced by the Strum Detection circuitry.

C. Electronic Hardware

Each fret is associated with a PCB, which contains 4 LEDs, one per string. The PCB also has a D-Flip-Flop to receive the

voltage stimulus from the previous PCB, apply the stimulus to the current fret, and forward the stimulus to the next PCB.

The Strum Detection circuitry is analog and takes the electrical signal from the piezoelectric pickup integrated into the guitar as an input. The signal is put through several signal processing stages, ultimately generating a digital value indicating whether the guitar was strummed. This digital signal serves as an interrupt for the Teensy and is used as a control signal to the internal state machine running on the Teensy.

IV. Design Requirements

To meet the use-case requirements, several critical design specifications have been 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 system must support down to 1/8th notes at 100 BPM to accommodate different tempos, and it should indicate the target tempo at a minimum volume of 70 dB, ensuring the signal is audible. The feedback mechanism is designed to provide visual and audible cues for in-time playing accuracy with a response time of at most less than 100ms, ensuring that the user can practice with a consistent tempo.

The fretboard must incorporate 56 individually addressable LEDs to offer detailed visual guidance for different notes and scales. The rest of the board is unnecessary, as beginner users rarely use the upper portion of the guitar. Since scales require most of the board to be lit up at the same time, the system should allow 2/3 of the fretboard to be illuminated at half brightness, striking a balance between visibility, convenience, and safety.

Safety is a key consideration, as the guitar strings will be 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 pause songs. The file upload capability should support up to 1GB of users' MIDI files for a personalized learning experience. The display of practice statistics, rhythm and accuracy scores, and song upload must respond to user input within 0.25 seconds, given a reasonably functioning network.

These design specifications ensure SuperFret meets the defined use-case requirements.

The quantitative specifications are summarized:

Specification	Value
Strum to LED latency	<50ms
Total system	100 beats per minute support
LEDs	> 56 individually addressable LEDs
Safety	< 1 mA through body
File storage	1GB
Network delay	< 0.25 second
Finger placement Detection	99% accuracy
lighting up the correct LED(s)	100% accuracy

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 app, 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, the hosting of the web app can introduce delays that will result in not meeting 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 will run the real-time I/O without worrying about hosting a web app, allowing the target latencies to be achieved. 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:

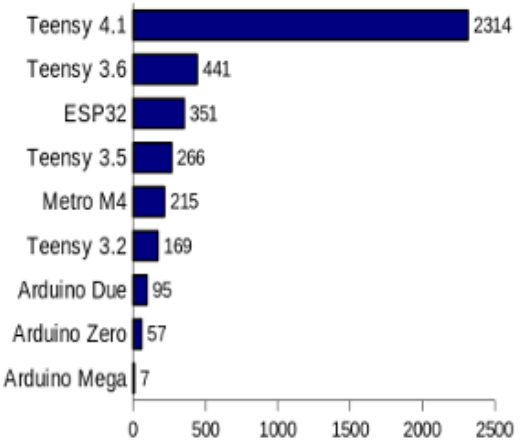

Fig. 4. The "CoreMark" CPU Performance Benchmark [4], [5]

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].

After considering the degree of prior experience, CPU performance, memory, 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 experience of the guitar. To reduce pin and wire count, a switch array can be employed. 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

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

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. This solution, shown in Fig. 5, requires only 2 wires between each fret, 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 needs a D-flip-flop, but this drastically outweighs requiring 14 individual wires for each fret.

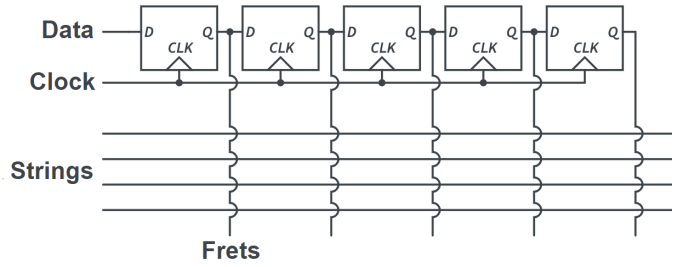

Fig. 5. A 6 GPIO method for reading finger positions

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, as this would require many wires, 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 perfectly replicate the spacing between the frets on a PCB. 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 1.5mm.

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 around 1.5mm, the total height of the LEDs and the PCB cannot exceed around 1mm. 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 will have to 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.

VI. System Implementation

Appendix Table 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 Subsystem

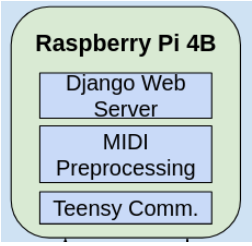

Fig. 7. Block diagram for the RPi. Zoomed-in crop of block diagram in Appendix Table I.

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 will house all the file and user information to achieve a consistent state for the server. Each entry in the database will represent a song and contain:

- name: name of the song/file
- file: the file path to the MIDI (actual file will be stored in a separate folder)
- active: a boolean to store if the song is currently being played
- type: either a song or scale

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 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.

3) Teensy Communication

The RPi will run a UART communicator process to establish and maintain a connection between the Teensy and the Pi over a specified port. Its job will be to receive user requests from the web server and convert them into signals, which can then be sent to the teensy and vice-versa. Upon a start request from the user, the web server will tell 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 network latency concerns. When a fileStop command is issued, the UART communicator will interrupt the Teensy by raising its PAUSE GPIO pin to high. Any other communication needed will look very similar.

B. Teensy/Embedded Subsystem

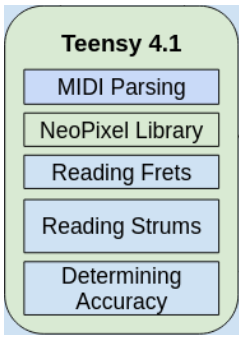

Fig. 8. Zoomed-in crop of block diagram in Appendix Table I. The Teensy microcontroller is the glue between the User Interface and the electronic hardware. The Teensy's software is structured as a state machine.

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 or scale 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 bytes constituting the MIDI file. 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 "WAIT FOR STRUM" state, where it parses the file and waits for the user to start playing the guitar by strumming. The `strum` input is a digital signal generated by the strum-detection circuitry detailed in the following "*Electronic Hardware*" section. When the first strum is sensed, the Teensy enters the "USER EXPERIENCE" state, where it lights up LEDs, reads frets, and continues detecting strums.

Once the user finishes playing the song (the end of the MIDI file is reached), the "WAIT TO START" state is entered again. The Teensy enters the PAUSED state if the user pauses the system through the web app. The Teensy enters the initial state if the user restarts the system through the web app. Otherwise, it waits until a strum is detected to resume the user experience.

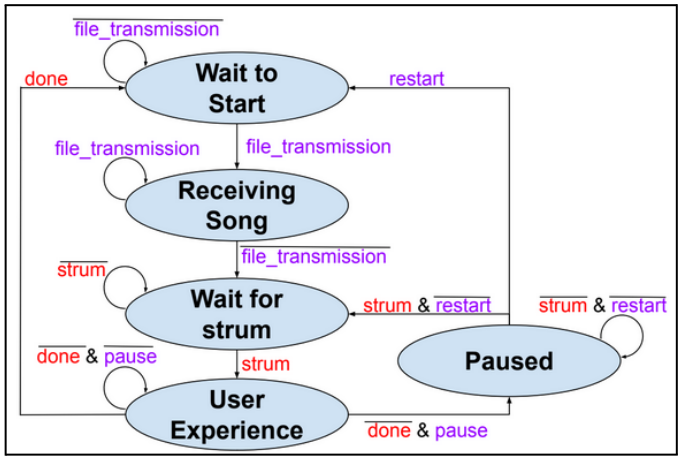

Fig. 9. State Machine for the Teensy's Software

2) MIDI Parsing

The "abridged" MIDI file coming from the RPi only contains the timing and note information the Teensy needs to determine when to light up a particular note's LED and when it should expect the user to play that note.

From a timing perspective, the goal is to find the number of seconds to wait before a note is played and how long before the note is released. However, the MIDI file specifies such information as "MIDI ticks" elapsed, a somewhat arbitrary time unit. MIDI files typically specify two conversions to transform ticks into seconds. One is the number of MIDI Ticks per quarter note (TPQN), and the other is microseconds per beat, or tempo. Since a beat is defined to be the length of a quarter note, the following formula holds:

$$\mu s\ duration = \frac{MIDI\ tick\ duration}{TPQN} \times tempo \qquad (2)$$

This allows conversion of durations specified in terms of "MIDI Ticks" into absolute time units that can be measured and timed on the Teensy.

TPQN is specified as the 4th line in the MIDI file header (Fig. 10). The tempo is specified towards the beginning of the "MTrk" (Track) portion of the file by the byte sequence `FF 51 03 XX XX XX`. "`XX XX XX`" is the hexadecimal representation of the number of microseconds per beat. Since 1 beat corresponds to a quarter note, the tempo sets the length of a quarter note. When such parameters are not specified, defaults outlined in the MIDI file standard are used [8]. The default TPQN is 48, and the default tempo is $\frac{500{,}000\ \mu s}{beat}$. The following formula is used to express the tempo in the more familiar BPM [9]:

$$BPM = \frac{60\ seconds}{1\ minute} \cdot \frac{1{,}000{,}000\ \mu s}{1\ second} \cdot \frac{1}{tempo}$$

```
4D 54 68 64 (MThd)
00 00 00 06
00 01 (format 1 = one or more simultaneous tracks)
00 03 (3 tracks)
01 80 (0x180 = 384 ticks/quarter note)

4D 54 72 6B (MTrk)
00 00 01 D2 (Chunk length 0x01D2 = 466 bytes follow)
00
        FF 58 04 04 02 18 08 (time signature)
00
        FF 51 03 08 52 AE (tempo = 0x0852AE = 545,454 us/beat ⇒ 2.1812 seconds / measure ⇐⇒ 110 BPM)
```

Fig. 10. MIDI TPQN and tempo parsing example for an excerpt from a "Twinkle Twinkle Little Star" MIDI file [10]. Here, the TPQN is 384, and the tempo is 545,454 microseconds per beat.

The "track" portion of the MIDI specifies events such as playing a particular note ("Note ON" event) and releasing a note ("Note OFF" event) [11]. The duration to hold a note and wait before playing the next note is specified in terms of MIDI ticks. This number is encoded using a Variable Length Encoding (VLE) Format [11] [12]. Fig. 11 shows an example of parsing the variable length encoding "`81 40`" into a tick duration. The figure also shows Note ON and OFF events for playing the note C4 twice.

```
00 (0 delta MIDI ticks)
   90 3C 32 (0x90 = Note ON event, 0x3C = Note 60 (C4))
81 40 (1|000|0001 0|100|0000 ⇒ 000|0001|100|0000 = 192 ticks)
   80 3C 00 (0x80 = Note OFF event, 0x3C = Note 60 (C4))
81 40
   90 3C 32
81 40
   80 3C 00
```

Fig. 11. MIDI note and duration parsing example. This is an excerpt from a Track portion of a MIDI file for Twinkle Twinkle Little Star [10].

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.

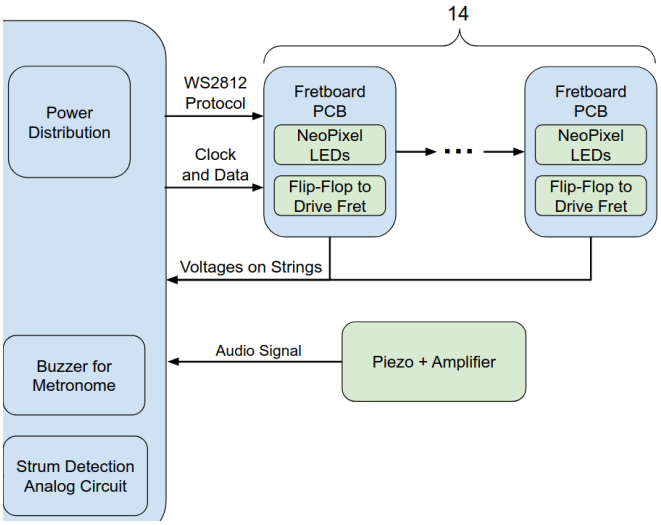

Fig. 12. Zoomed-in crop of block diagram in Appendix Table I, focusing on the electronic hardware.

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, a circuit takes in the guitar's audio signal and outputs a digital signal indicating when the guitar is strummed.

Rather than using a microphone to pick up the sound produced by the guitar, our system uses a piezoelectric sensor integrated into the guitar. This sensor converts the mechanical motion of the guitar strings into a voltage. This process is subject to significantly less noise than using a microphone, which can be affected by ambient noise.

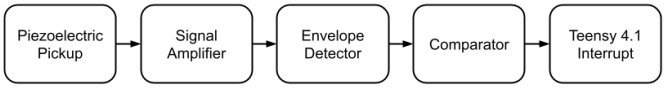

Fig. 13. Block diagram of the strum detection circuitry

The block diagram for the strum detection is shown in Fig. 13. The physical circuitry corresponding to this block diagram is shown in Fig. 14.

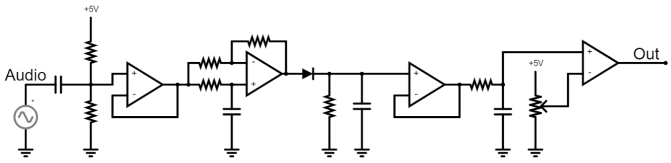

Fig. 14. Physical implementation of the strum detection circuit

The circuit first takes in an audio signal, adds a DC offset of around 1.5V, and then buffers it. Then, the buffered signal is amplified about its mean value. The amplified and shifted signal is fed into an envelope detector, which converts the audio waveform into a signal indicating its amplitude. Finally, this signal is fed into a comparator whose threshold is set by a trim potentiometer. The comparator outputs a digital signal that is fed to an input pin on the Teensy, allowing it to detect when a strum has occurred.

2) Fretboard PCBs

To connect the addressable LEDs and drive each fret to 3.3V individually, our system integrates a 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 ~56 LEDs, this corresponds to around 2ms to write to all the LEDs.

Furthermore, 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. Using the Teensy, a logical high can be clocked into the first PCB, and this can be shifted to the next PCB, allowing each fret to be driven high one at a time. A 3.3kΩ resistor connects 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 PCB design is shown in Fig. 15. The top half of the board contains the 4 addressable LEDs, D1-D4, and the bottom half contains the D-flip-flop and current limiting resistor. 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. The boards will be conformal coated for safety and to prevent accidental shorting occurring through the metal strings.

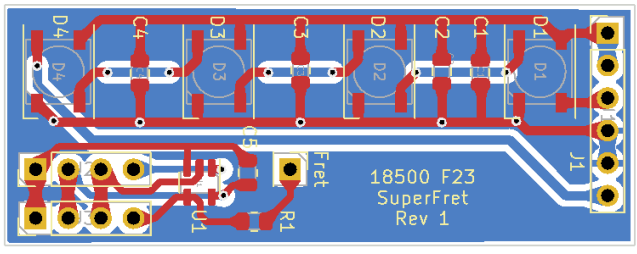

Fig. 15. Fretboard PCB layout

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" will be used. This is a PCB that plugs directly into the 40 header pins on the RPi, as shown on the right side of the board in Fig. 16.

The first task handled by the Pi Hat is filtering any noise in the 5V power supply connected to the Hat via a barrel jack and distributing this to the Teensy, Pi, and fretboard PCBs.

The second task of the Pi Hat is to connect the Teensy and RPi's GPIO pins.

The hat will also take the 3.3V logic signal outputted by the Teensy for the NeoPixels and shift it to a 5V logic signal as required by the LEDs.

We plan to use an active buzzer as a metronome to indicate the target tempo to the user while they are playing. This will beep in a short pulse once per beat of the music. Active buzzers can be driven by simply pulling an output pin on the Teensy, either high or low, making this design for the metronome easy to implement on the firmware side.

The final feature of the Pi Hat will be sets of input and output pins on the board that will enable the connection of the various peripheral devices, such as the fretboard PCB, the strum detection circuit, and the electrical connections to the guitar's string.

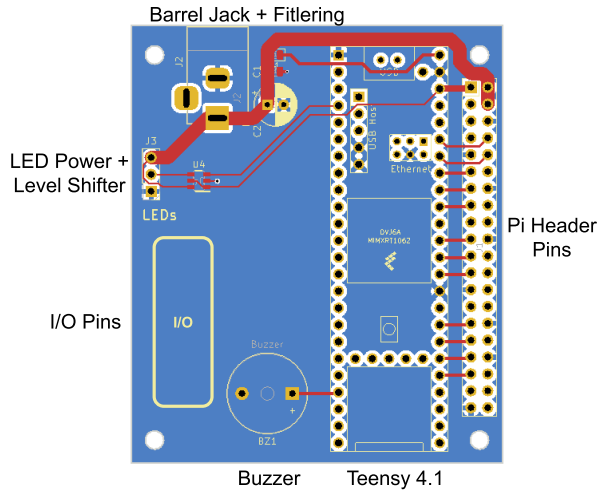

Fig. 16. Pi-Hat PCB layout (work in progress)

4) *Power Supply*

The system will be powered using a 5V DC wall adapter, which will connect to the Pi Hat using a 5mm barrel jack connector and be distributed to the various components. The fretboard PCB D-flip-flops operate on 3.3V, which will be supplied by the Teensy's internal 3.3V linear regulator. The total expected current draw is 2.5A for the Pi, 0.15A for the Teensy, 0.1A for the strum detection, and 1.7A for the LEDs at half brightness. This sums to 4.45A, so a 5A power supply was chosen for the project.

VII. Testing, Verification, and Validation

To validate the hardware latency performance, comprehensive tests will be conducted. The aim is to scrutinize the real-time responsiveness of the SuperFret system under diverse playing conditions.

A. *Latency*

An oscilloscope will be utilized to precisely measure the time delay between the initiation of a strumming action and the corresponding LEDs being written to. This test holds critical significance as it directly addresses the design requirement of achieving a latency of less than 50 milliseconds, ensuring that the system provides instantaneous feedback to the user during guitar practice sessions.

Simultaneously, the web app's network delay test will evaluate the responsiveness of the web application. The test will entail interaction with different features, such as song uploading and accessing practice statistics. The time taken for responses to be received and displayed will be measured to ascertain that the web application operates within the stipulated network delay of less than 0.25 seconds. This ensures that users experience a smooth and responsive interface when interacting with the web application, aligning with the design specifications and user expectations.

B. *Accuracy*

For accuracy testing, a strum identification test is designed to assess the system's ability to identify strums accurately. We will quantify the system's ability to correctly identify strums by performing 100 1/8th note strums at 100 BPM on each string and recording ambient sound levels. The success criteria for this test will be determined by calculating the percentage of correctly identified strums, directly addressing the accuracy requirements outlined in the design specifications. This quantitative measure clearly indicates the system's performance in identifying and responding to strumming actions.

In addition to strum identification, a finger placement test will be conducted to evaluate the system's accuracy in detecting the placement of fingers on different string and fret positions. This involves systematically placing a finger on each combination and monitoring the serial port. Repeating this process multiple times and calculating the percentage accuracy will quantify the system's precision in detecting finger placement. This test directly validates the accuracy requirements for finger placement detection as specified in the design specifications. Overall, these tests are crucial in ensuring that the SuperFret system not only meets theoretical design trade-offs but also demonstrates robust performance aligned with the specific use-case requirements for the project.

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 will use a lab bench ammeter capable of measuring down to 0.1µA to verify this. Under normal conditions, participants will contact the 3.3V guitar string with 1 hand and

a ground signal with the other. A 10kΩ potentiometer will be between the 3.3V source and the string, and the potentiometer will initially start at 10kΩ. While monitoring the current, the potentiometer's resistance will be turned to 0Ω, and the current will be recorded. If the current ever reaches 1mA while lowering the potentiometer resistance, the test will be stopped. To test the maximum current the strings carry, we will use the ammeter to connect the string to ground and verify that no more than 1mA flows. We will also use a lab bench voltmeter capable of measuring down to 1μV to verify that the fretboard has no exposed 3.3V or 5V contacts other than the 3.3kΩ current limited contact.

D. User Experience

For user experience evaluation, subjective tests will be conducted to gather feedback on the web application and hardware components. The tests are designed to assess the users' perception of the system's usability and effectiveness.

Users will be asked to interact with the web application and provide ratings on a scale of 1 to 5 for categories such as the intuitiveness of the interface, readability of statistics, and ease of uploading songs. These subjective evaluations will be averaged to create a quantitative metric for the overall user experience with the web application. For instance, a user-friendly interface is crucial to the system's success, as it directly impacts the accessibility and satisfaction of the users.

Similarly, users will be 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 5 for each category will be 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.

These user experience evaluations are essential for obtaining qualitative insights into the effectiveness and user-friendliness of the SuperFret system. The system's success in meeting the user-centric design goals will be quantified by aggregating user ratings. The feedback gathered from users will be invaluable in making iterative improvements to enhance the overall user experience, ensuring that the SuperFret system fulfills technical specifications and is well-received by its target audience of beginner guitar players.

VIII. Project Management

A. Schedule

The Gantt chart in Appendix Table III 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 is 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.

B. Team Member Responsibilities

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

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

Owen designs the electronic hardware, which involves the PCBs on the fretboard, the strum detection circuitry, and the interface board that allows signals to pass between the Teensy and RPi.

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

C. Bill of Materials and Budget

So far, we have spent $207.22. The ordered parts include the fretboard PCBs, their components, and the guitar. Appendix Table II shows the full breakdown of these orders. We have also acquired an RPi from the ECE department and parts such as the Teensy and hookup wire from Roboclub, of which Owen is a member. Appendix Table II also indicates the projected future costs, primarily consisting of two more PCB orders. The total expected cost of the project is currently $417.22, leaving $182.78 for additional components that are needed or for expedited shipping.

D. Risk Mitigation Plans

Several critical risks have been identified, each requiring careful consideration and mitigation strategies to ensure a smooth design implementation.

One risk involves detecting open string strums when there is no direct contact between the fret and string. Mitigation strategies include removing this scenario from the use case by transposing all open strings up one semi-tone, making open strings impossible, or just trusting the user and assuming the right string was played when a strum is detected.

The ambiguity in fret-string contact due to multiple ways to play the same note poses another risk. To address this, we may develop an algorithm to determine which alternative of the same note is most appropriate to play. The algorithm will take in recently played notes to determine which fret is physically closer.

IX. Related Work

Fret Zealot [13] 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 scales, 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 if notes were played correctly as 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 greater interactability.

X. Summary

The SuperFret project aims to develop a system to assist beginner guitar players in improving their skills and playing basic songs. Learning new songs and chords, practicing tempo, and drilling finger exercises are made simple through our interactive design. Our product comprises a web application hosted on an RPi, a Teensy microcontroller as the embedded system's brain, and electronic hardware on the guitar, including LEDs on the fretboard and a microphone for strum detection. Users interact with the system through the web app, uploading MIDI files for practice. The LEDs on the fretboard guide users on finger placement and strumming based on the uploaded files.

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.

Anticipated challenges in implementation and meeting requirements include detecting open strings without direct contact and addressing ambiguity in fret-string contact. These challenges require careful consideration and mitigation strategies to ensure the system's robustness and alignment with user expectations. Additionally, refining the algorithm for the most appropriate notes and maintaining optimal latency are ongoing challenges crucial to effectively meeting the system's use-case requirements. Overall, addressing these challenges will be key to the success of the SuperFret project and its positive impact on beginner guitar players.

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
TQPN - Ticks per Quarter Note

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 & fretboard radius explained," Guitar Gear Finder, https://guitargearfinder.com/guides/guitar-neck-shapes/ (accessed Oct. 12, 2023).
[8] M. Colli. "MIDI Beat Time Considerations." https://majicdesigns.github.io/MD_MIDIFile/page_timing.html (accessed Oct. 3, 2023)
[9] "The MIDI file format's Tempo Meta-Event" http://midi.teragonaudio.com/tech/midifile/ppqn.htm (accessed Oct. 3, 2023)
[10] "Twinkle Twinkle Little Star" MIDI Download. https://onlinesequencer.net/1815844# (accessed Oct. 3, 2023)
[11] "Standard MIDI-File Format Spec. 1.1, updated", https://www.cs.cmu.edu/~music/cmsip/readings/Standard-MIDI-file-for
[12] "Variable-length quantity" https://en.wikipedia.org/wiki/Variable-length_quantity (accessed Oct. 8, 2023)
[13] "Best way to learn guitar: How to learn guitar at home," Fret Zealot, https://www.fretzealot.com/ (accessed Oct. 13, 2023).

APPENDIX

Table I: System Block Diagram

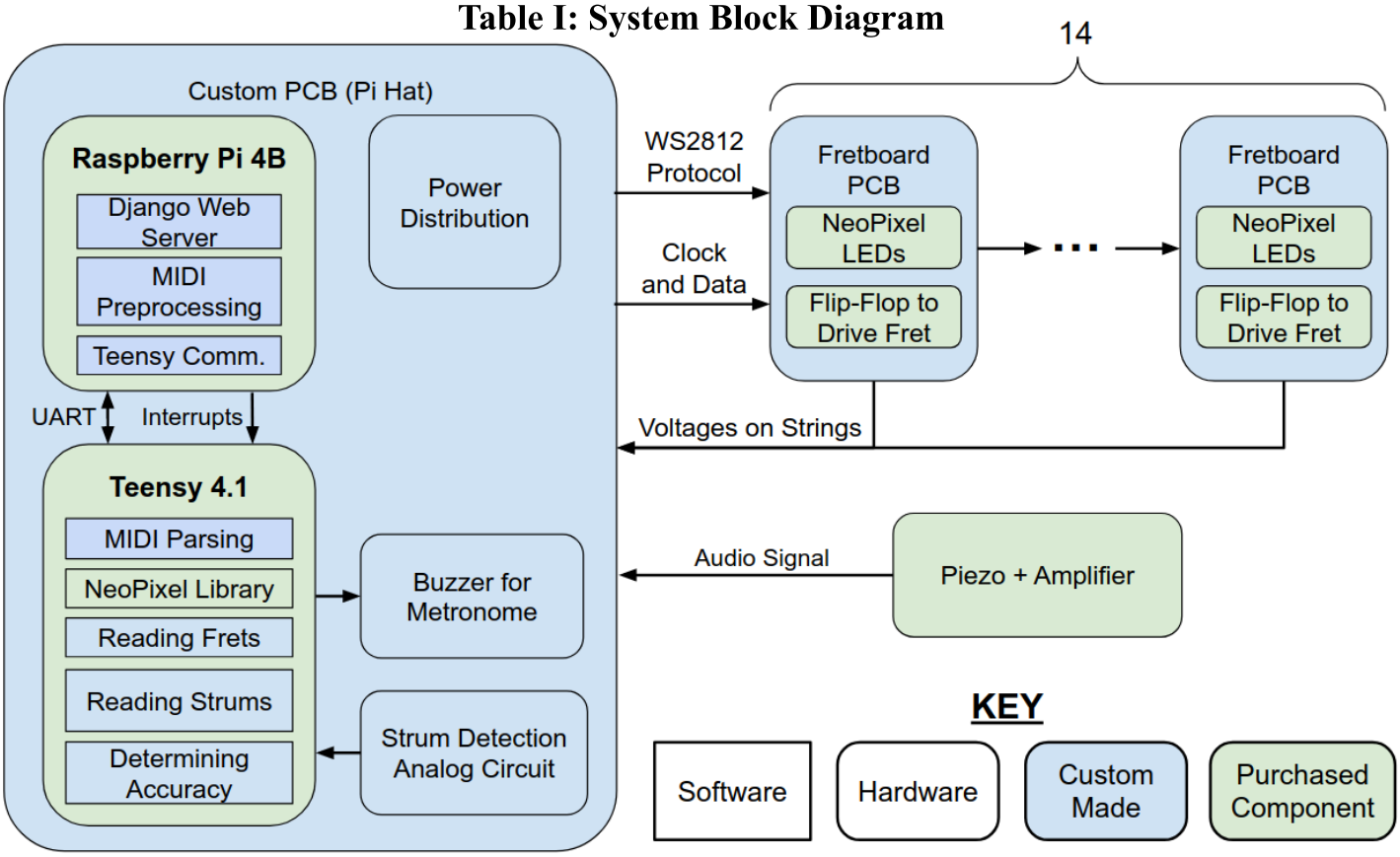

Table II: Bill of Materials and Budget

Item	Description	Model Number	Manufacturer	Quantity	Unit Cost	Tax & Shipping	Obtained
Fretboard PCB	Custom PCB with NeoPixels and a D-Flip-Flop	-	JLCPCB	10	$0.50	$25.20	☑
Fretboard Stencil	Solder paste stencil for fretboard PCBs	-	JLCPCB	1	$7.00		
NeoPixel LEDs	Addressable NeoPixel RGB LEDs	1655	Adafruit	3	$4.50	$10.04	☑
Op-Amps	8DIP CMOS rail to rail op-amps	MCP602-E/P	Microchip Technology	3	$0.82		
Barrel Jack	5.5x2.1mm barrel jack connector	54-00133	Tensility Intl.	2	$1.05		
5V/5A Wall Adapter	AC/DC wall adapter 5V/5A 5.5x2.1 barrel plug	PPL36U-050	Phihong USA Corp.	1	$17.15		
D-Flip-Flop	D-flip-flop SOT23-5 IC, 1 bit, rising edge, non-inverted	SN74LVC1G79DBVR	Texas Instruments	15	$0.33		
3.3K Resistor	3.3K Ohm 1% 1/8W 0805 resistor	RC0805FR-07120RL	YAGEO	50	$0.02		
0.1uF Capacitor	0.1uF ceramic capacitor 50V 0805	CL21B104KBFNNNE	Samsung	125	$0.02		
Acoustic Bass Guitar	Full size 4 string acoustic bass guitar	B003H8MXQQ	Best Choice Products	1	$109.99	$6.60	☑
Electret Microphone	Electret microphone with MAX4466 amplifier	B07DRGF8C2	HiLetgo	1	$0.00	$0.00	☑
Teensy 4.1 Microcontroller	ARM Cortex-M7 600MHz microcontroller	DEV-16771	PJRC	1	$0.00	$0.00	☑
Raspberry Pi 4B	Single board computer, 1.5GHz, 4 Core, 4GB RAM	SC0194(9)	RPi Foundation	1	$0.00	$0.00	☑
24 AWG Hookup Wire	Stranded 24AWG wire for connecting PCBs	-	-	-	$0.00	$0.00	☑
Fretboard PCB	Full order of multiple sizes of custom Fretboard PCBs	-	JLCPCB	20	$0.50	$30.00	☐
Fretboard Stencils	Solder paste stencils for fretboard PCBs	-	JLCPCB	2	$7.00		
Teensy and Pi Breakout PCB	Custom PCB to connect Teensy, Pi, and I/O	-	JLCPCB	5	$0.50	$30.00	☐
Breakout PCB Stencils	Solder paste stencils for breakout PCBs	-	JLCPCB	1	$7.00		
NeoPixel LEDs	Addressable NeoPixel RGB LEDs	1655	Adafruit	7	$4.50	$10.00	☐
D-Flip-Flop	D-flip-flop SOT23-5 IC, 1 bit, rising edge, non-inverted	SN74LVC1G79DBVR	Texas Instruments	15	$0.33		
Breakout PCB Components	Level shifters, buzzer, passives, diodes	-	-	1	$40		
Conformal Coating	Conformal coating to insulate the fretboard PCBs	-	-	1	$30	$0.00	☐
				Total	**$417.22**		

Table III: Schedule