Spring 2015

Electronic Learning Guitar

Justin Fiser
University of Akron Main Campus, jwf17@zips.uakron.edu

The senior design project in which I was involved was through the Department of Electrical and Computer Engineering. I was a member of a 4-person design team that was given the task of creating a working project from an original idea. My design team had chosen to create an electronic learning guitar, an instrument that would be able to display guitar chords on the guitar neck through an LED grid. It would also be able to display chords regardless of how the guitar was tuned and determine if a chord was played correctly or not.

Before the design phase of the project began, it was my responsibility to perform background research on existing products available on the market and explore patents that were similar to our proposed project. This research provided a basis for how we could create a unique project.

I was designated as the team leader among my design team. This title meant I would be in charge of deciding meeting times and locations. I was also in charge of managing the efforts of the rest of the team in order to make sure deadlines were met.

In the fall of 2014, the team started the design phase. I was predominately in charge of the design of the LED Module, the portion of the project that decoded the signals from the main microcontroller and displayed the guitar chords on the lighting array. This design process began with a discussion as to how the LEDs could be illuminated. The solution was to use a series of decoders into a commonly used common-cathode, common-anode LED grid circuit. It was also determined that the best way to mounted the circuit board onto the guitar would be to use a flexible circuit board (FCB), which would have a low enough profile to fit below the guitar strings.

I took all of the measurements on the guitar neck onto which the flexible circuit board would be mounted. These measurements included the length of each fret, the width of the guitar neck along each fret, and the spacing of the guitar strings in a given fret. The measurements were necessary for the proper placement of the LEDs under the strings as well as for overall mounting on the guitar neck. I was responsible for placing the order for the FCB and soldering the LEDs on the board.

The flexible circuit board needed a way to connect to the main guitar mounted circuit board. I designed a PCB that attached the FCB to a ribbon cable. The ribbon cable then connected to the main board. This was a very inexpensive solution to join the two modules of our design.

I also designed the power circuit for the project. The divided guitar pickup required +/- 7 volts and the LED module and microcontroller required 5 volts. I was able to successfully implement the design of the power circuit using 2 fixed voltage regulators for +5V and +7V and an inverter for -7V.

Being the most familiar with guitar playing and music theory of the group, I checked the software development of my peers to ensure that the program was working as intended. This involved reviewing test outputs of code for their accuracy and providing recommendations regarding how to improve or embellish the software.

In the attached report, I was responsible for the background section and the engineering/marketing requirements for our design. I wrote the sections for the design of the Power Module and LED Module as well as the operating instructions and testing procedures.

Following the success of our design project over the past year, I will be taking the lead in submitting a form for disclosure of invention. With the aid of the University, I am hopeful that there are aspects to my team's design that would be patentable.

Electronic Learning Guitar

Final Design Report

Design Team #03

Jacob Barb
Mike Bolin
Justin Fiser
Kellen Reusser

Faculty Advisor: Dr. De Abreu-Garcia

April 22, 2015

Table of Contents

List of Figures

List of Tables

Abstract: *The goal of this project is to create an electric guitar that detects and displays chords in different tunings for learning and exploration purposes. A hexaphonic pickup will generate the required signals and a processor will detect the pitch. The computer software will apply the pitch information to determine the tuning and the voicings of chords. LEDs mounted on the fretboard of the guitar will indicate a note to be played.*

1. Problem Statement

1.1 Project Need

Learning to play chords on a guitar can be difficult for beginner and intermediate players. An understanding of the notes on the guitar neck and knowledge of music theory is needed to play a chord. A guitar that is capable of displaying where to place one's fingers on the guitar neck could be a useful tool for players of all skill levels.

1.2 Project Objective

The goal is to design a guitar that is capable of instructing a user in finger placement of guitar chords. The guitar must be able to process six notes at a time, sensing the unique tuning of the neck, and calculating the coinciding LEDs associated with the notes. The guitar will recognize when it is improperly tuned, assist the user in the tuning process, and indicate when the user has successfully played the desired chord.

1.3 Background

Learning how to play the guitar takes a profound amount of patience and motivation. The structure of the notes on the guitar can be difficult to comprehend. As Eli Harrison explains in his article *Challenges Facing Guitar Education*, “The guitar does not have white frets and black frets the way a piano has white keys and black keys. Beginning guitarists must immediately interpret the differences between half steps and whole steps, unlike beginning pianists, who can rely on the consecutive white keys of C major. Second, the six strings of the guitar create a condition where one pitch can have several fingerings. [1]” Musicians new to the guitar can easily become discouraged, as the learning curve for the guitar is steep. Thanks to modern technology, however, great strides are being made in flattening this learning curve to encourage more people to try learning the instrument.

A large number of guitar tablature and digital sheet music services are readily available through the internet. Since 2000, automated transcription of music through the use of digital signal processing has grown considerably and spurned the International Symposium on Music Information Retrieval [2]. Consequently, a vast library of music is available to musicians of all disciplines; however users must still apply their own personal judgment to determine whether or not a chord or melody was played correctly.

Different approaches have been made toward improving how an individual who is new to the guitar can learn the instrument with relative ease. One approach was through the use of an audio-visual system called Augmented Reality, which utilized web cameras to interpret cue cards and applied a laptop screen to display where to place one's fingers [3].

Other designs involve the use of an LED array placed directly on the fret-board. Patent US 5408914 applies an LED layout and uses comparator circuits to determine accuracy of the notes played on the guitar [4]. US 20110011241A1 contains a design for a guitar with compatibility with a gaming console and television [5]. US 20060249008A1 features an apparatus for the user's fret hand (the hand that is placed on the neck of the guitar) that color codes where the fingers are to be placed on the LED fret-board [6]. US 8395040B1 integrates capacitive sensing to determine the location of finger placement. US 20120192700A1 uses a visual animation of the lights on the guitar to indicate the timing of when the player is supposed to pluck or strum the guitar when synchronized with music [7].

On occasion, a guitar player may wish to retune a guitar to create a different sound, mood, or style. The typical or "standard" tuning of the guitar is EADGBE (from lowest to highest). Common types of retuning are open G (DGDGBD), open D (DADF#AD), and open C (CGCGBE). Once retuning is achieved, the layout of the notes on the neck of the guitar is completely different. Learning how to play chords across different tunings is tedious, and relearning the neck of the guitar could prove to be just as difficult as understanding the guitar in standard tuning. Having a system that can think for the user and calculate where the chords have "moved to" is the basis for this project, in addition to being a learning aid for beginner and intermediate players.

The concept of the guitar tuner has not changed since the late 1970's. Since sound involves the oscillation of waves, its signal may be compared with that of an adjustable voltage oscillator, wherein the voltage oscillator signal and pickup signal are compared to attain proper tuning [8]. If the two notes differ in any way, the state and degree of the instrument's sharpness or flatness is communicated to the user (typically though indicator lights), who then readjusts and assesses the status of the tuner following the adjustment.

The concept of the guitar in this design is to merge both the ideas of the lights on the guitar neck with the guitar tuner to provide a guitar that not only teaches the user how to play in one tuning, but also how to play in multiple tunings. While LED guitars are already available to consumers today, a guitar with this level of versatility still remains to be seen in the marketplace.

In the proposed guitar, the user decides how he or she would like to learn the guitar because the computer will be able to process any tuning, be it standard, open, or completely uncommon.

2. Design Requirements Specification

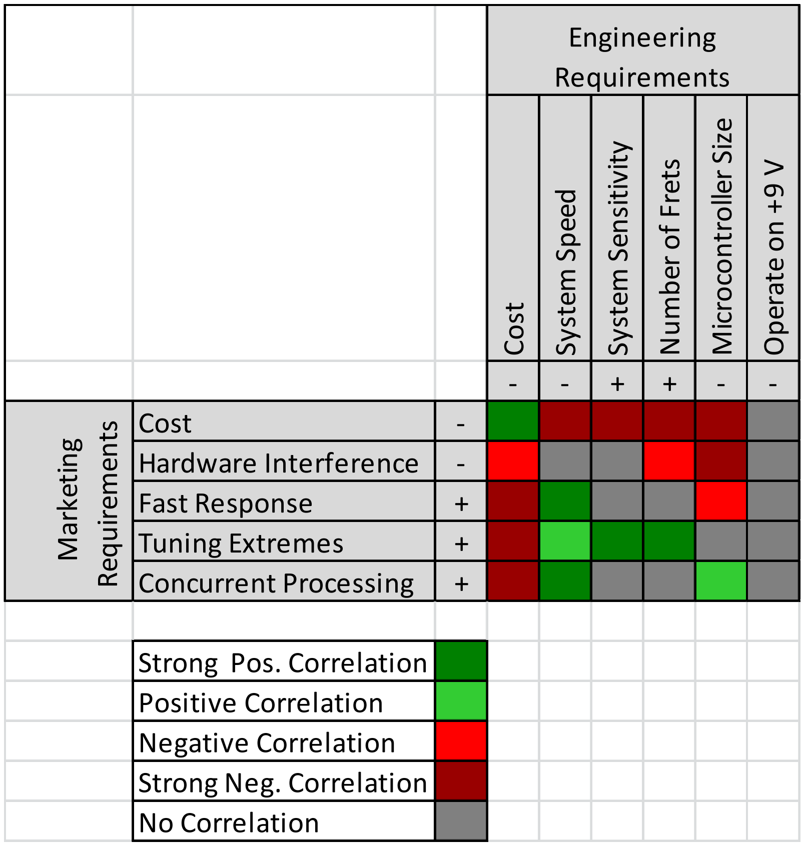
An LED smart guitar should satisfy the expectations of consumers in terms of the device capabilities and playability. Accordingly, the design requirements of the smart guitar project may include those listed in *Table 1*.

Table 1. Electronic Learning Guitar Design Requirements

Marketing Requirements	Engineering Requirements	Justification
3,4	1. *The pitch detection should be able to detect frequencies ranging over B1 (61.7 Hz) to G5 (784.0 Hz).*	These frequencies are roughly the physical tuning limitations of the highest (thinnest) and lowest (thickest) guitar strings.
3,4	2. *The electronics will allow for 10 cents sharp or flat as acceptable error range for notes considered in tune.*	Within the range of either 10 cents sharp or flat, most human ears won't be able to perceive that the note(s) is/are out of tune.
	3. *The microcontroller should meet the size requirements for I/O.*	The microcontroller selected for this project must be able to accommodate all necessary inputs and outputs.
3,4	4. *The process of pitch detection and feedback should execute in less than 2 seconds.*	The system should provide timely feedback to the user.
	5. *The system should be able to operate from a source of 9V.*	A cost-effective and commercially available +9 V wall adapter will provide power to the guitar-mounted equipment.
2,4	6. *Chords will be displayed over frets 0 through 12 (first octave).*	The inclusion of 12 frets covers all possible notes under a single octave on a given string.
Marketing Requirements 1. *The guitar should be a real, playable electric guitar.* 2. *The guitar should display each note via lit LEDs.* 3. *The guitar should be able to process multiple notes simultaneously.* 4. *The guitar should be able to display and detect chords.* 5. *The guitar should work regardless of tuning and will inform user if guitar is out of tune.* 6. *The guitar should have an easy-to-read user interface to interact with the player.*		

2.1 Engineering-Marketing Tradeoff Matrix

The below engineering-marketing tradeoff matrix illustrates the correlation among the two sets of requirements.



			Engineering Requirements					
			Cost	System Speed	System Sensitivity	Number of Frets	Microcontroller Size	Operate on +9 V
			-	-	+	+	-	-
Marketing Requirements	Cost	-	🟩	🟥	🟥	🟥	🟥	▨
	Hardware Interference	-	🔴	▨	▨	🔴	🟥	▨
	Fast Response	+	🟥	🟩	▨	▨	🔴	▨
	Tuning Extremes	+	🟥	🟢	🟩	🟩	▨	▨
	Concurrent Processing	+	🟥	🟩	▨	▨	🟢	▨

Strong Pos. Correlation	🟩
Positive Correlation	🟢
Negative Correlation	🔴
Strong Neg. Correlation	🟥
No Correlation	▨

2.2 Engineering Tradeoff Matrix

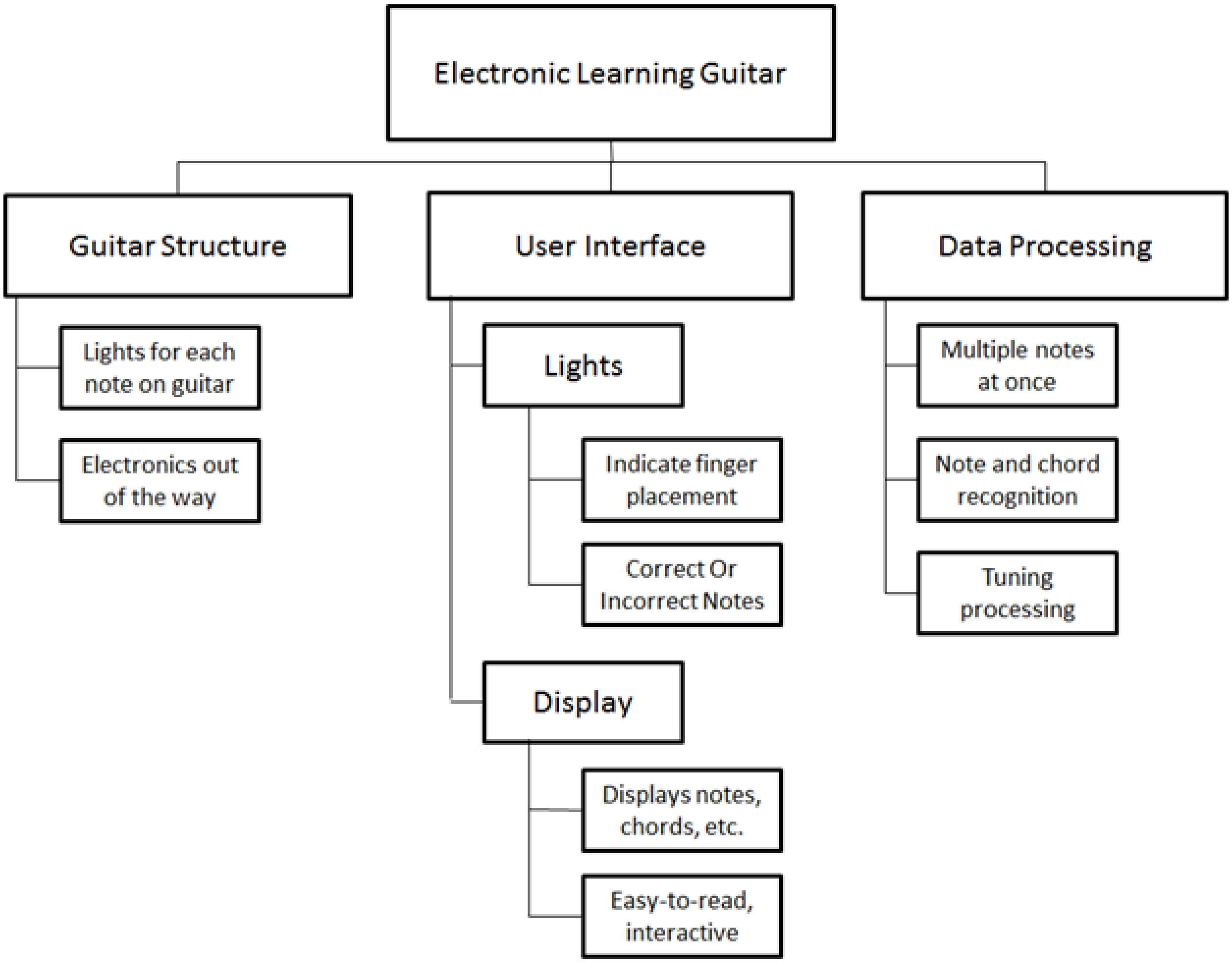
The chart below shows the correlation of the engineering requirements among themselves.

		Cost	System Speed	System Sensitivity	Number of Frets	Microcontroller Size	Operate on +9 V
		-	-	+	+	-	-
Cost	-		🟥	🟥	🟥	🟥	🟥
System Speed	-			▨	🟥	🟥	▨
System Sensitivity	+				🟥	🟥	▨
Number of Frets	+					▨	▨
Microcontroller Size	-						

Positive Correlation	🟩
Negative Correlation	🟥
No Correlation	▨

2.3 Objective Tree

The Objective Tree below is an alternative way to show how the marketing requirements can be broken down into different subcategories.


Electronic Learning Guitar
Guitar Structure
Lights for each note on guitar
Electronics out of the way
User Interface
Lights
Indicate finger placement
Correct Or Incorrect Notes
Display
Displays notes, chords, etc.
Easy-to-read, interactive
Data Processing
Multiple notes at once
Note and chord recognition
Tuning processing

3. Technical Design

Figure 1 shows the Level 0 block diagram of the LED Learning Guitar Design. Given a user input, the LED Learning Guitar will be able to display where the fingers need to be placed in order to correctly play a chord, both on the LED array on the guitar fretboard and on the Graphical User Interface (GUI). Two sources of power are needed, one for the software, which is run on a PC, and the other for the guitar-mounted electronics.

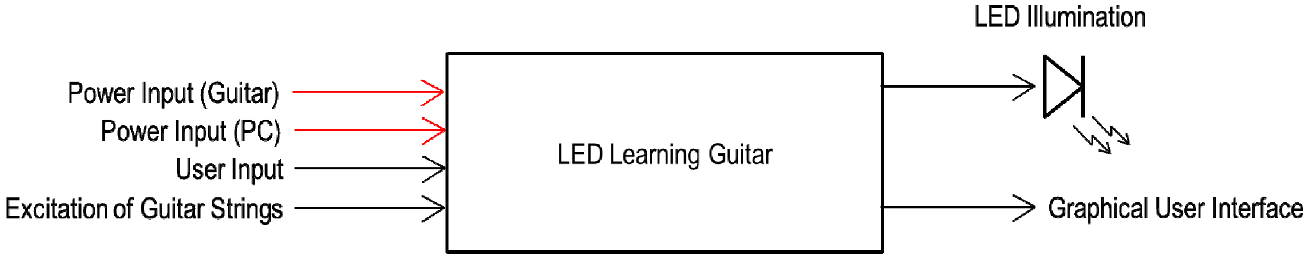


Figure 1. Level 0 block diagram of the LED Learning Guitar.

Figure 2 shows the Level 1 block diagram of the LED Learning Guitar design. Notice that the design contains five guitar-mounted modules and a PC. Power signals are shown in red. Data signals are shown in black. The functional requirements tables are listed below.

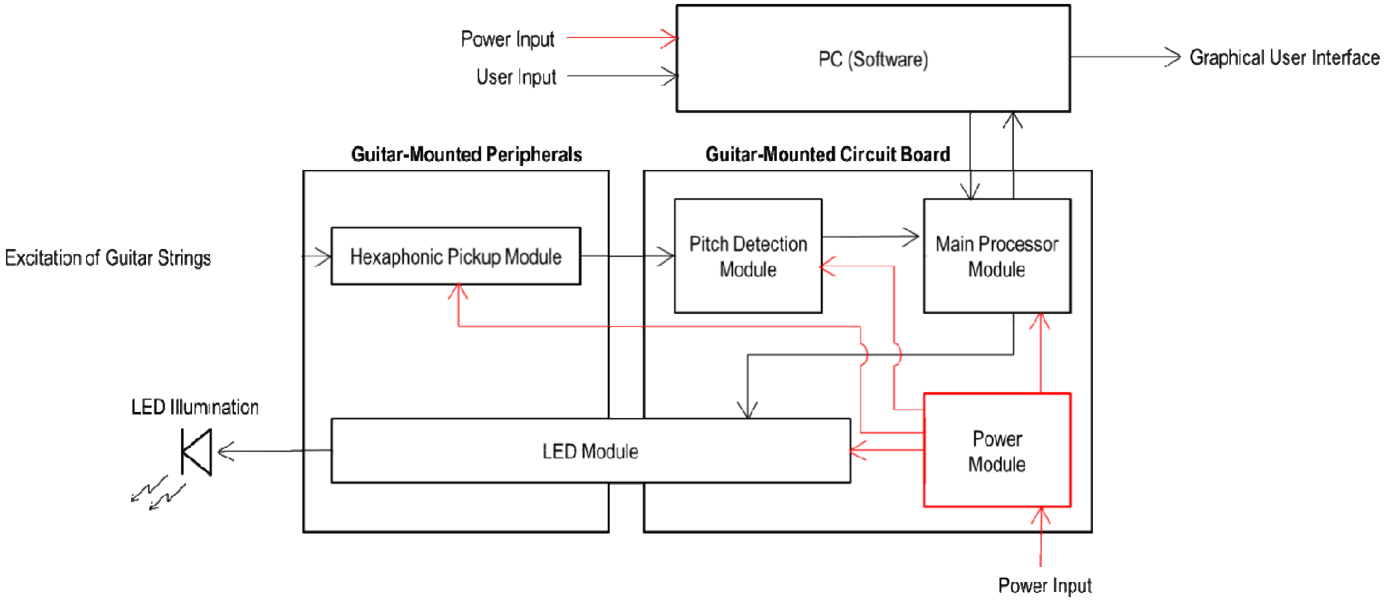


Figure 2. Level 1 block diagram of LED Learning Guitar.

Table 2. Functional Requirements for Power Module

Module	Power
Input	Wall Adapter: +9 V DC
Outputs	Voltage Signals: +5 V DC and ±7 V DC
Functionality	Take +9 V DC, from an AC/DC wall adapter, to produce +5 V DC for the Pitch Detector Module and ±7 V DC for the Hexaphonic Pickup Module.

Table 3. Functional Requirements for Main Processor Module

Module	Main Processor
Input	Power Module: +5 V DC Pitch Detection Module: Frequencies of guitar notes played PC (Software): Coordinates of LEDs to be turned on
Outputs	LED Module: Addresses for decoding and illuminations
Functionality	Relay note information to PC and receive chord information from PC to export addresses that control which LEDs turn on.

Table 4. Functional Requirements for Pitch Detection Module

Module	Pitch Detection
Input	Power Module: +5 V DC Hexaphonic Pickup Module: Analog guitar signals
Outputs	Main Processor Module: Frequencies of guitar notes played
Functionality	Take up to 6 analog signals produced by the Hexaphonic Pickup Module, measure the frequency of each signal, and export that frequency to the Main Processor Module.

Table 5. Functional Requirements for Hexaphonic Pickup Module

Module	Hexaphonic Pickup
Input	Power Module: ±7 V DC Excitation of guitar strings
Outputs	Pitch Detection Module: Up to 6 analog signals, one from each string
Functionality	Actuate when strings are strummed and produce up to 6 analog signals.

Table 6. Functional Requirements for LED Module

Module	LED
Input	Power Module: +5 V DC Main Processor Module: Addresses for decoding and illumination
Outputs	LED illumination
Functionality	Decode the addresses from the Main Processor Module and light the specified LEDs on the fretboard.

Table 7. Functional Requirements for PC (Software)

Module	PC (Software)
Input	Power input from separate wall adapter User input Main Processor Module: Note information
Outputs	Main Processor Module: Coordinates of LEDs to be turned on.
Functionality	Determine guitar tuning and chord placement on guitar neck.

In the discussions that follow, each module in **Figure 2** above will be examined in detail.

3.1 Power (JF)

3.1.1 Voltage Requirements

The Level 1 functionality for the Power Module is shown in **Figure 3**, which will receive a +9 V input from a barrel jack connector and convert it to three voltage outputs: +7 V, +5 V, and -7 V. The Power Module is responsible for supplying the voltage needs of all other design modules and equipment. The following list describes the voltage requirements of each module:

1) The Hexaphonic Pickup Module requires ±7 V, and a ground connection.
2) The microcontroller, Pitch Detection Module, and LED Control Module require +5 V and a ground connection, each.
3) The LED Lighting Module will operate on +5 V but is driven by the LED control module, not the Power Module.
4) Software will be run on a PC that is powered separately from all other design modules.

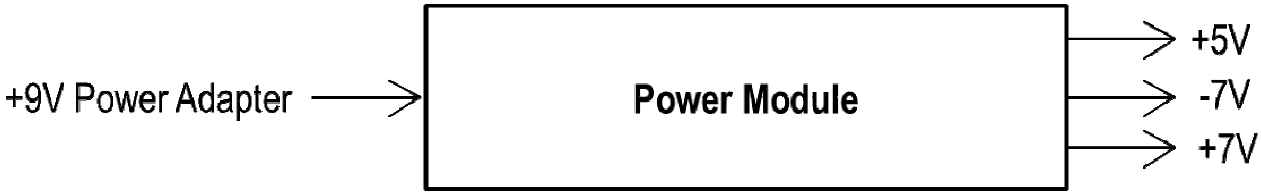


Figure 3. Power Module Level 1 Block Diagram

Table 8. Functional Requirements for Power Module

Module	Power
Input	Wall Adapter: +9 V DC
Outputs	Voltage Signals: +5 V DC and ±7 V DC
Functionality	Take +9 V DC, from an AC/DC wall adapter, to produce +5 V DC for the Pitch Detector Module and ±7 V DC for the Hexaphonic Pickup Module.

Thus, the Power Module must provide three voltage outputs: +7 V, +5 V, and -7 V. To realize these voltages, a +9 V signal from an AC/DC wall adapter will be stepped down to +7 V and +5 V. A DC/DC inverter can be used to invert +7 V to -7 V. A wall adapter was selected with the following rationale:

1) It can be purchased at a relatively low cost.
2) Unlike batteries, it will not need to be replaced (except if it malfunctions or gets damaged).
3) It plugs directly into the wall, providing continuous power so long as the electrical outlet is energized.
4) Of reduced weight and bulk on the body of the guitar by selecting a wall adapter over a battery.

3.1.2 Step-Down Voltage Regulators

The Level 1 functionality for the Power Module is shown in **Figure 4**. Notice that the Power Module contains a +9 V rail, two voltage regulators, and a DC/DC inverter.

The functional requirements of these subsystems are detailed next in that which follows.

Table 9. Functional Requirements for +9 V DC Voltage Rail

Module	+9 V DC Voltage Rail
Input	Wall Adapter: +9 V DC
Outputs	Voltage Regulator LDO-1: +9 V DC Voltage Regulator LDO-2: +9 V DC
Functionality	Take +9 V DC, from an AC/DC wall adapter, to provide +9V DC to the voltage regulators.

Table 10. Functional Requirements for LDO-1

Module	Fixed Voltage LDO-1 (A linear voltage regulator)
Input	+9 V DC Voltage Rail: +9 V DC
Outputs	LDO-1 Regulated Voltage: + 5 V DC
Functionality	Provide +5 V DC to the Main Processor and Pitch Detection Modules.

Table 11. Functional Requirements for LDO-2

Module	Fixed Voltage LDO-2 (A linear voltage regulator)
Input	+9 V DC Voltage Rail: +9 V DC
Outputs	LDO-1 Regulated Voltage: + 7 V DC
Functionality	Provide +7 V DC to the Hexaphonic Pickup and DC/DC Inverter Modules.

Table 12. Functional Requirements for DC/DC Inverter

Module	DC/DC Inverter
Input	LDO-2 Regulated Voltage: +7 V DC
Outputs	DC/DC Inverter Voltage: -7 V DC
Functionality	Provide -7 V DC to the Hexaphonic Pickup Module.

The +9 V DC signal from the wall adapters feeds a voltage rail that serves both linear voltage regulators. The first step in obtaining the required voltages is to step down +9 V. In deciding how to accomplish this, the current output is an important attribute that helps select the type of buck converter and provides a constraint with the maximum current output of the component. The following list outlines the circuit demands of the components that will be used in the design:

1) *LED Module*: While the LED module contains 78 LEDs, only one LED is lit at a time (refer to Section D for details). In other words, the load current of the LED Module is equal to the amount of current drawn by one illuminated LED, that is, 10 mA. The decoders used for lighting control require an insignificant amount of current (less than 1 μA).
2) *Atmega328p*: When active at 16 MHz and +5 V, the operational current draw is at most 10 mA. However, an additional 15 mA is needed to drive the power indicator LED.
3) *Atmega2650*: In the worse-case scenario, this device will draw 25 mA at 16 MHz and +5 V. As in 2) above, an additional 15 mA will be needed for the power indicator LED.
4) *FT232RL*: I_{CC}, the supply current for the chip, requires 15 mA. The outputs will draw 25 mA.

The total current draw for the entire system is 240 mA. Because the current and power demands of the design are very low, a cost-effective and efficient way to obtain the desired positive voltages is to use two linear, low dropout voltage regulators (LDO), as illustrated in **Figure 4**.

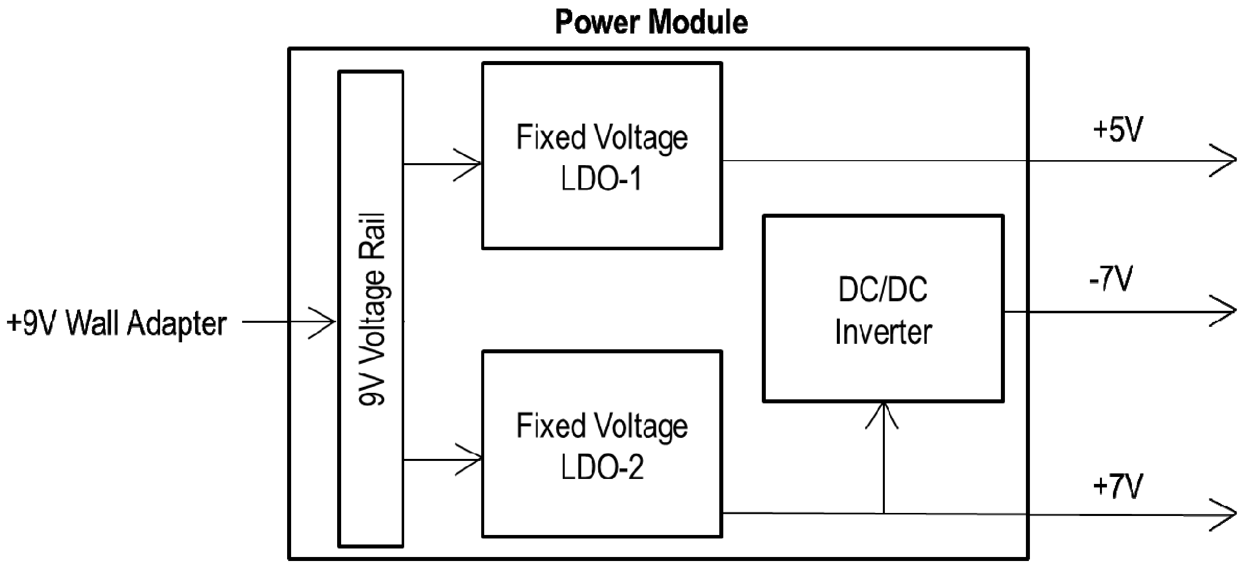


Figure 4. Power Module Level 2 block diagram.

3.1.3 LDO-1 Selection

To reduce both cost and conserve space on the circuit board, LDO-1 will be a fixed-output LDO. The Texas Instruments LM1117-N-5.0 (Refer to Appendix A) can convert up to +20 V to +5 V. With a maximum current rating of 800 mA, this chip satisfies the constraint imposed by the maximum currents calculated in *3.1.2*. **Figure 5** displays the schematic for the LM1117. The chip has three pins, V_{IN}, V_{OUT}, and GND. The V_{IN} pin will be tied to the +9 V voltage rail. V_{OUT}, at +5 V, will be tied to the LED and Pitch Detection modules as well as the Main Processor Module. The GND pin will be connected to the ground plane of the guitar mounted circuit board. On both V_{IN} and V_{OUT}, the data sheet requires a minimum of a 10 µF shunt-to-ground tantalum capacitor to handle low frequency ripple of the input and output voltages.

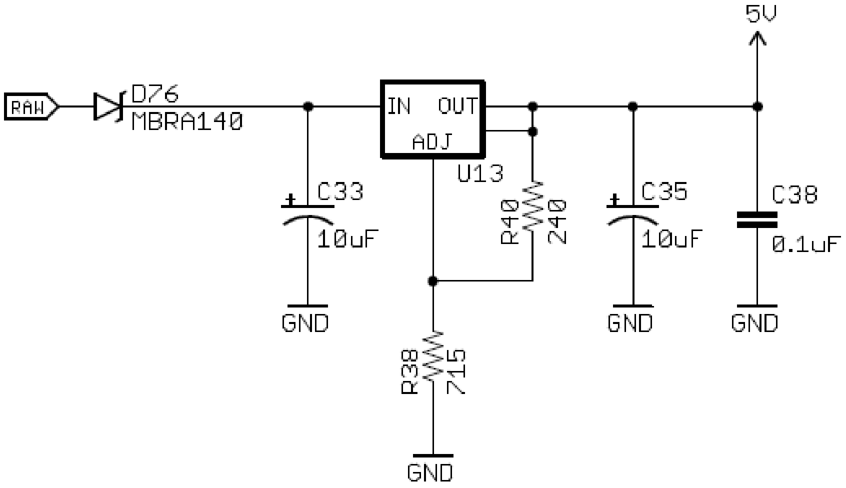


Figure 5. LM1117-N-5.0 schematic.

3.1.4 LDO-2 Selection

Similar to LDO-1, LDO-2 will also be a fixed-output LDO, the only difference being that LDO-2 will provide a fixed output of +7 V. The chip that was selected to realize this voltage is the Rohm Semiconductor BD70GA3WEFJ-E2 (Refer to Appendix A), shown in **Figure 6**. This IC has 8 pins, 2 of which are NC. +9 V will be connected to the Vcc and EN (enable) pins. Output pin V_o will yield the desired +7 V. V_o connects to V_{o_s}, the voltage monitor pin. GND and FIN are tied to the ground plane of the guitar-mounted circuit board. Just like LDO-1, input and output capacitors are used across Vcc and V_o, respectively.

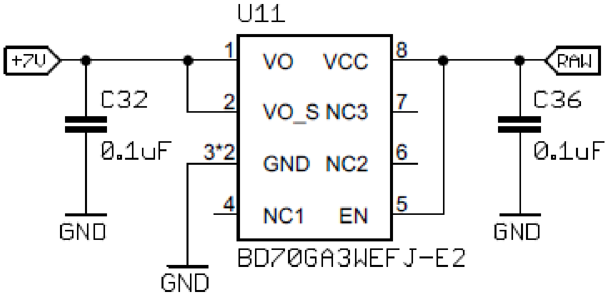


Figure 6. BD70GA3WEFJ-E2 schematic.

3.1.5 DC/DC Inverter Selection

The IC chosen for this task is the Texas Instruments TPS6755 (refer to Appendix A). The general schematic of this chip can be seen in **Figure 7** below. The data sheet for this inverting DC/DC converter provides recommended operating conditions and values for many of the circuit components. The only component values not provided are R39 and R36, the resistors responsible for the voltage divider that dictates what the output voltage will be. The equation below shows the relationship between the two resistors.

$$R39 = -V_o \frac{R36}{1.22}$$

Selecting R36 to be 10.2 kΩ and the desired output voltage at -7V, R39 is determined to be 58.54 kΩ. A standard resistor value close to this is 59.0 kΩ. The enable input, pin 1, determines whether or not the inverter is on or off, and it requires a minimum +2V signal in order to be on. To achieve an always-on state, the enable will be tied to the input voltage, as shown in **Figure 7**.

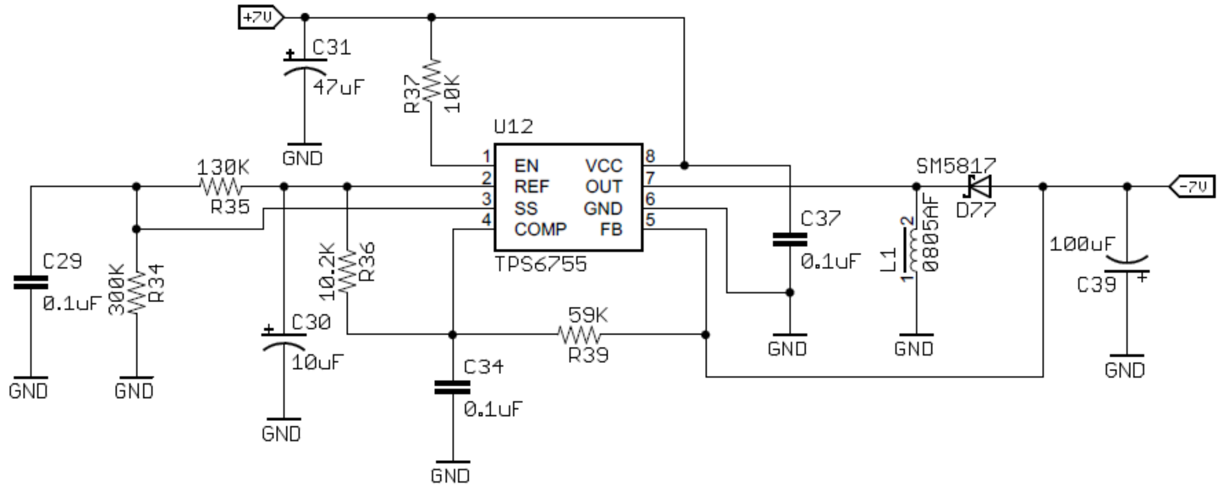


Figure 7. TPS6755 schematic.

3.2 Pitch Detection Module (KR)

3.2.1 Level 1 Design

The pitch detection module is responsible for determining the chord being played by the user. To do this the module must be able to:

1.) Identify or quantify the chords being played.
2.) Convey the chord information to the user, or at least pass this information on to another module for the same purpose.

These requirements are illustrated in the Pitch Detection Module Level 1 Block Diagram shown in **Figure 8**.

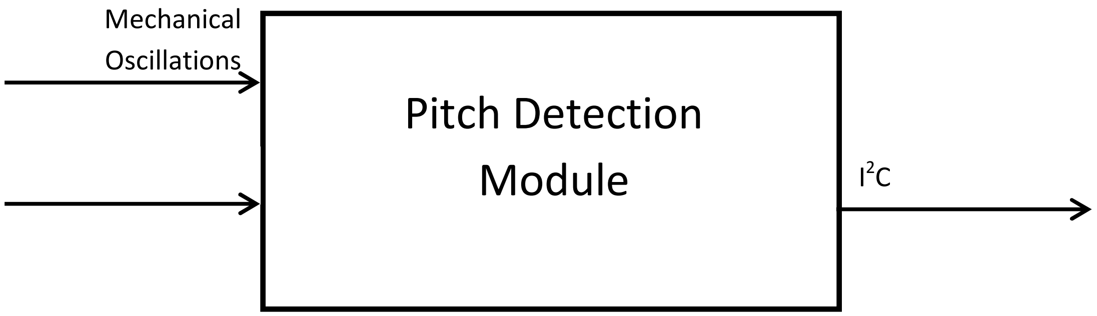


Figure 8. Pitch Detection Module Level 1 block diagram.

Table 13. Functional Requirements for Pitch Detection Module

Module	Pitch Detection
Input	Guitar: Mechanical Oscillations Power Module: Power

Outputs	I^2C: I^2C Communication
Functionality	Use the mechanical oscillations of the guitar strings to calculate the frequency of each of the six strings and output them by I^2C.

3.2.2 Level 2 Design

The original approach considered for detecting the chord being played was to process the output waveform of the audio jack of an electric guitar. After further research and consideration, it was determined that the signal processing of chords would be difficult and either hardware or software intensive, or both. Rather than considering the audio output, a superposition of six waveforms from the individual strings, it was decided that analyzing the contribution of each string individually would be a more worthwhile pursuit.

To separate the contributions of each string, a divided pickup was sourced. A traditional electric guitar pickup uses six magnets, one under each string, and one large coil. Each magnet contributes to the magnetic field experienced by the coil. When no strings are moving, the magnetic field is relatively constant, so there is no current induced on the coil. When a string is strummed, the string, a conductor, oscillates over the magnet, which causes a change in the magnetic field experienced by the coil. It turns out that this change in magnetic field and the oscillation of the string have the same frequency. Therefore, the current induced by this changing magnetic field is a waveform with the same frequency. The waveforms induced by all six of these strings oscillating are superimposed onto one another. This jumble of waveforms is the audio signal output of the electric guitar. A divided pickup, on the other hand, uses six smaller, separate coils, one under each string. Using this construction, each string generates its own separate output signal. The contribution from the neighboring string and magnet are negligible, meaning a clean and reliable signal can be measured from the output of each of these pickups. For the purposes of this design, a Roland GK-3 divided pickup was chosen as it was affordable and from a reliable guitar peripheral company.

From this divided pickup, six clean, analog, electrical signals can be measured. The designed pitch detector then must be able to measure these signals to deduce a frequency. It was decided that using a multiple of simple pitch detectors would yield a simpler design in a hardware sense. For this reason, one pitch detector is present on each string output from the divided pickup. Furthermore, the type of communication used to pass information from these pitch detectors to the next control block (the Main Processor Module) must minimize board real estate while still being able to pass the necessary information. The Level 2 Block Diagram in **Figure 9** illustrates this aforementioned set of requirements.

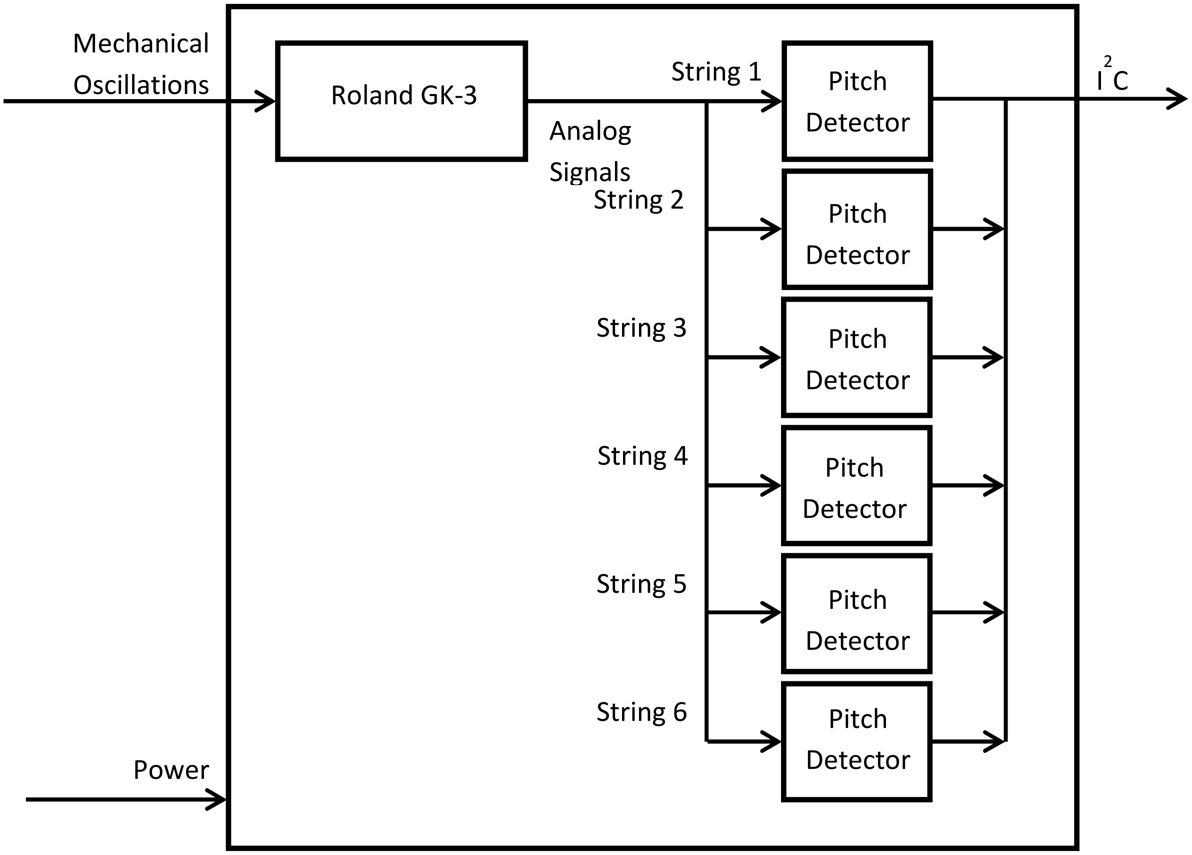


Figure 9. Pitch Detection Module Level 2 block diagram.

Table 14. Functional Requirements for Pitch Detector Sub-Module

Module	Pitch Detector
Input	Roland GK-3: Analog string signal
Outputs	I^2C: I^2C Communication
Functionality	Calculate the pitch of the note on string *n* and send this information out using I^2C.

3.2.3 Level 3 Design

The communication protocol chosen is inter-integrated circuits, or I^2C. This was chosen because it allows for a multitude of slaves to send information to a single master using only two conductors. This was ideal for this design as the information from these six pitch detectors needed to be passed to a single point while using limited board space.

It was clear that our pitch detector needed to be able to read an analog signal, process the signal to determine the frequency, and then use I^2C to pass the frequency information to the next module. Therefore, the pitch detector required:

1.) An analog to digital converter (ADC) to capture the analog waveform.
2.) A programmable processor to analyze the waveform and do calculations.
3.) A hardware I^2C module to simplify communication with the master.

The most logical choice to fulfill these requirements is a microcontroller. It was also decided that the chosen processors should be Arduino compatible in order to make firmware design a bit simpler. The Level 3 Block Diagram shown in **Figure 10** illustrates these requirements.

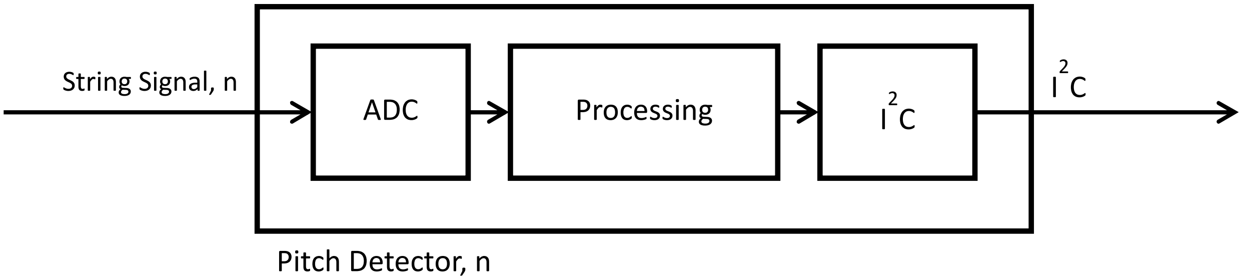


Figure 10. Pitch Detection Module Level 3 block diagram, Pitch Detector Sub-Module.

3.2.4 Components and Schematics

The microcontroller chosen for the pitch detectors was the Atmega328P. This was chosen for a number of reasons. First, it meets the above specifications: It has a 10-bit ADC, it is Arduino compatible, and it has a hardware I^2C interface. Second, the processor is relatively inexpensive compared to other Arduino compatible microcontrollers. Third, it comes in a QFP package, a good balance between easy to solder and small footprint. All of these justifications are supported by the datasheet provided in Appendix A. Finally, the Arduino development board for this microcontroller is affordable and comes in multiple different shapes and sizes, allowing for easy prototyping and development before a finished printed circuit board is constructed.

It was decided that a programmer header should be included for each microcontroller to allow for easy debugging and reprogramming, should the need arise. Although this addition requires a few extra components near the processor that will take up more real estate on the final board, it adds invaluable functionality for the design process. **Figure 11** shows the schematic for one pitch detector. In this schematic, JP1 is the program header and U1 is the microcontroller. The microcontroller runs on 5V. A 16MHz resonator, Y1, was chosen for the crystal required by the microcontroller. Decoupling capacitors of 0.1µF are used on the Analog Reference pin and the V_{CC} net, represented as C3 and C5 respectively. A power indicator LED was also added to the design as a first level of debugging. R3, the current limiting resistor, was chosen to be 330Ω to limit the current to 15mA. A pull-up resistor of 10kΩ, represented as R1, was used to tie the active-low reset pin high. A capacitor between the *DTR_1* net and the *Reset* net was required by the programmer being used and therefore was included as the 0.1µF capacitor C1. The full schematic of the Pitch Detector Module is also included in Appendix B.

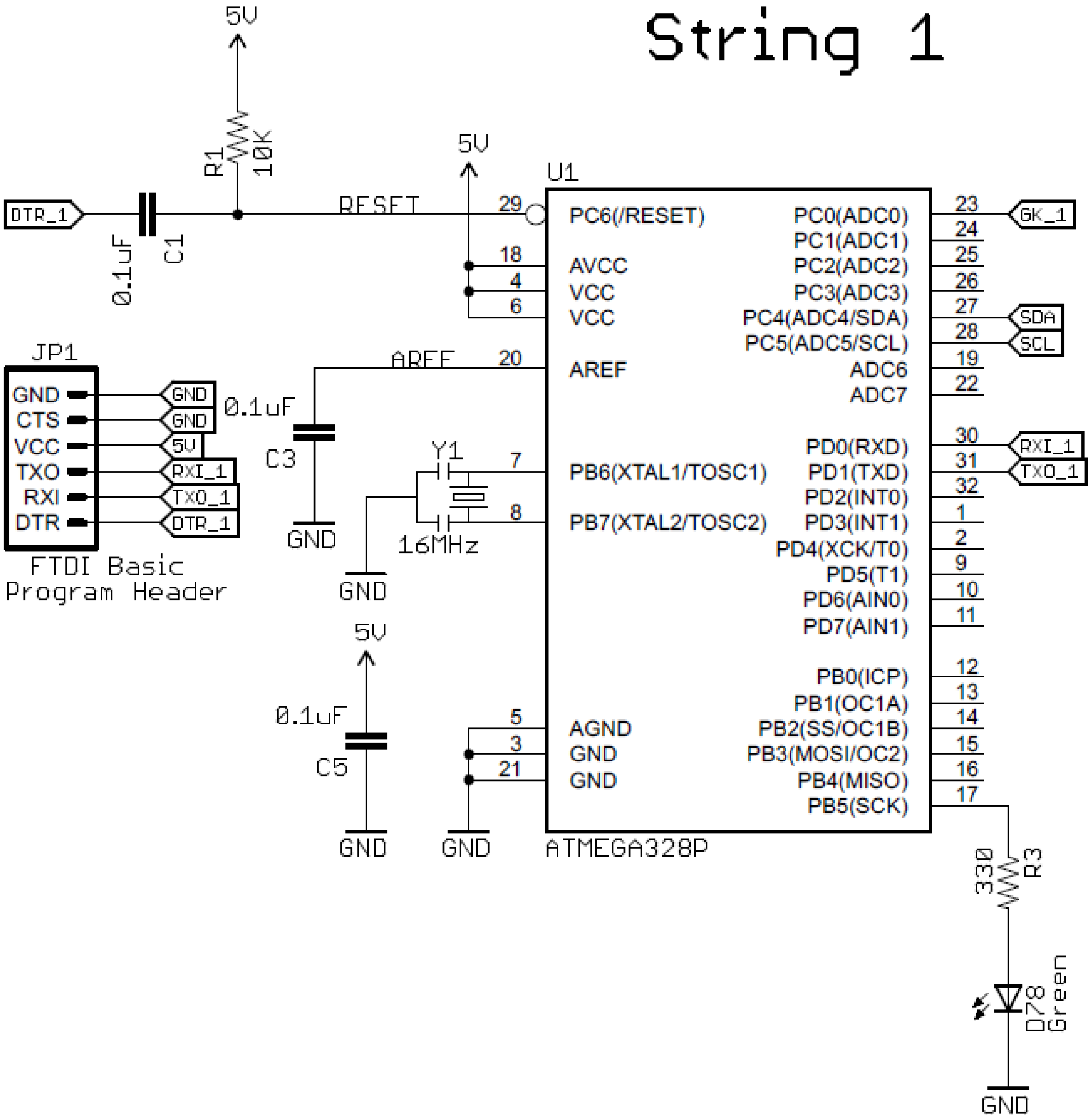


Figure 11. Schematic for one of six Pitch Detectors.

3.2.5 Layout

The Pitch Detection Module design was implemented as a section of the Guitar Mounted Circuit Board. Because of the symmetry of the design of this module, the layout structure for all six of the Pitch Detectors is near identical. The ADC0 pin of each microcontroller is connected directly to the GK-3 connector. The traces from the GK-3 connector were run with careful consideration. For example, the Pitch Detectors which handle strings of higher frequency have traces that minimize the length of parallel sections between each other. This technique was used in an attempt to minimize the amount of cross-talk due to these higher frequency signals. The Layout structure is shown in **Figure 12**. The photo in **Figure 13** shows the Pitch Detection Module section of the PCB populated with components. The full set of Gerber files is given in **Appendix D**.

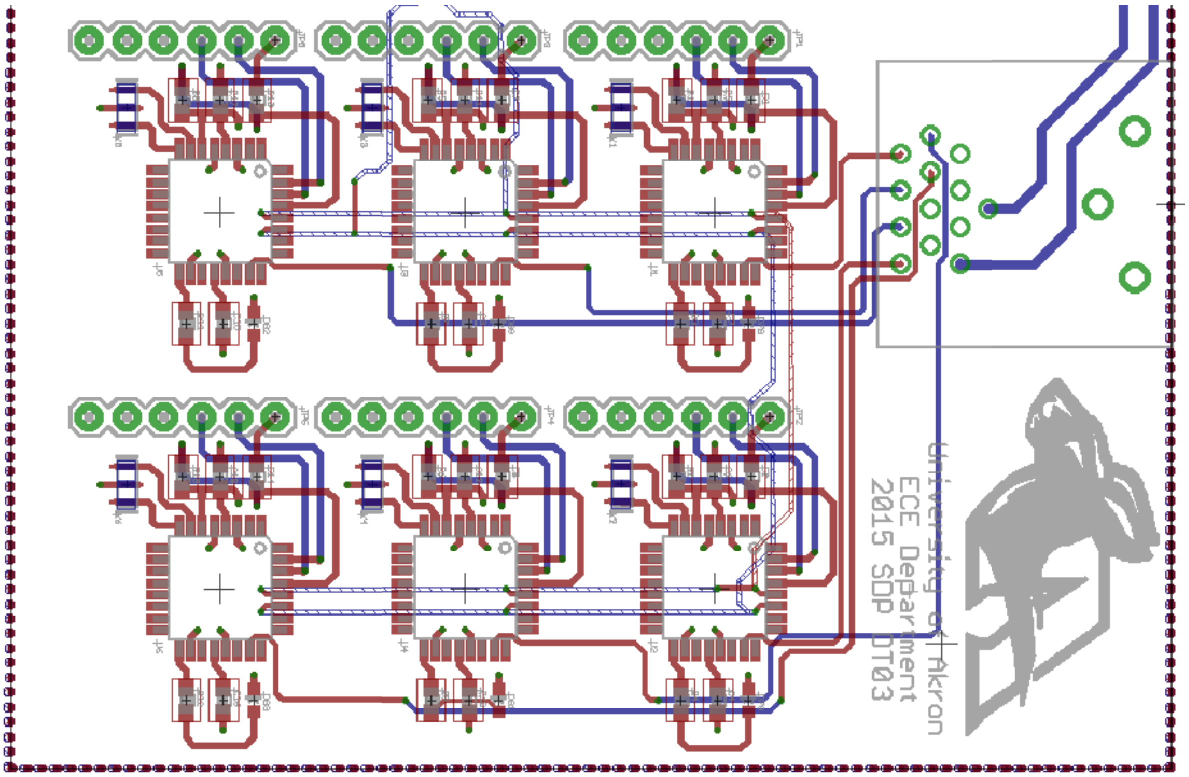


Figure 12. Pitch Detection Module layout.

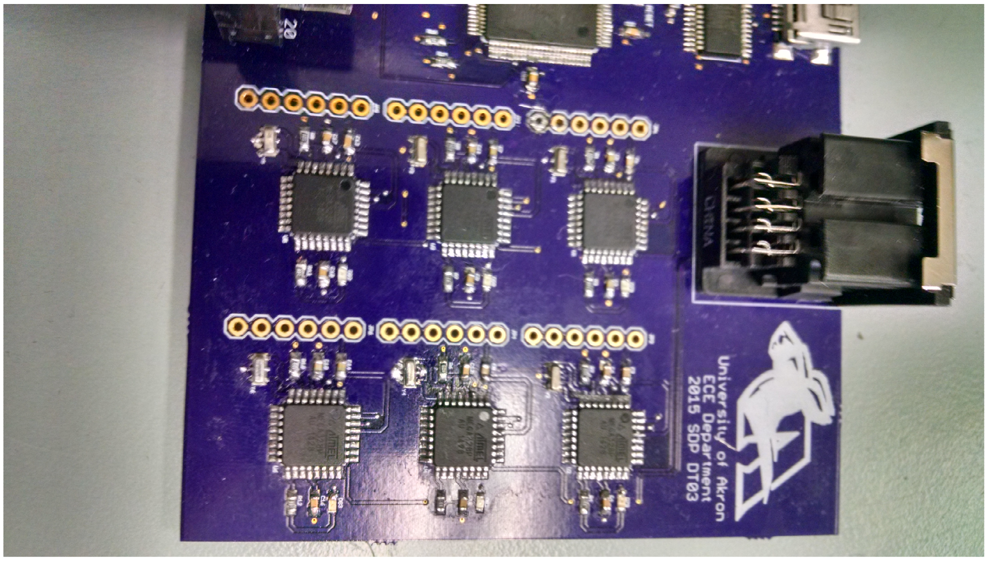


Figure 13. Pitch Detection Module populated PCB section.

3.2.6 Firmware Level 1 Design

The actual process of calculating the frequency being detected by the pitch detector is handled by the firmware with which the individual microcontrollers are programmed. As outlined in the Level 1 Firmware Block Diagram in **Figure 14**, the firmware is needed to take the analog signal provided by the Roland GK-3 as an input and pass a floating-point representation of the frequency out using I^2C. It should be noted that a floating-point representation of the frequency was chosen because the frequencies that represent different pitches are not whole numbers and the fractional component can be highly significant for lower pitch tones.

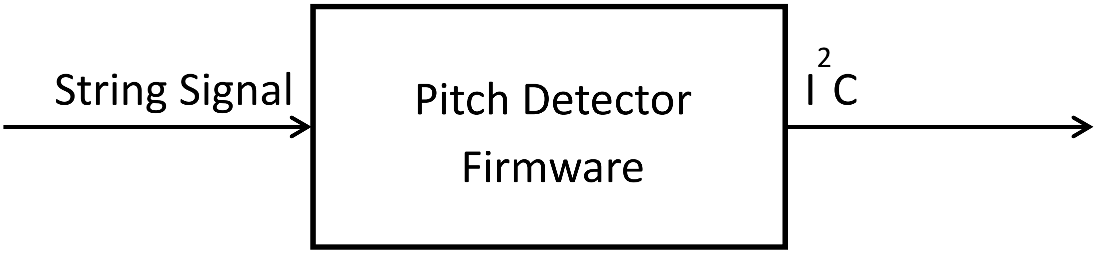


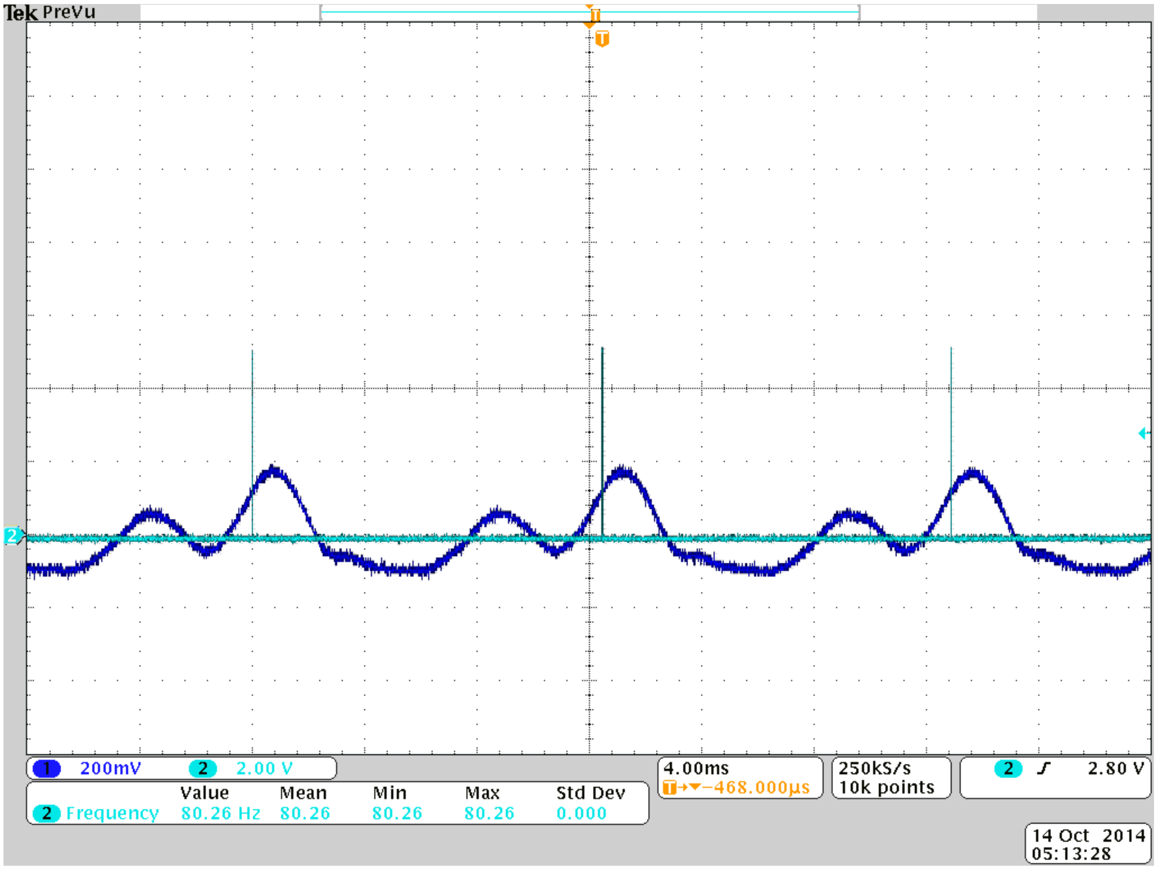
Figure 14. Pitch Detector Firmware Level 1 block diagram.

Table 15. Functional Requirements for Pitch Detector Firmware

Module	Pitch Detector Firmware
Input	Roland GK-3: Analog string signal
Outputs	I^2C: I^2C Communication
Functionality	Calculate the pitch of the note on string *n* and send this information out using I^2C.

3.2.7 Firmware Level 2 Design

Two different processes were considered to calculate the frequency from the analog signal captured by the ADC. One involved using a Fast Fourier Transform (FFT) to find the spectra component with the highest amplitude. After some research, however, it was determined that this approach was processor intensive, which would make the overall design less time and energy efficient compared to other methods. The method chosen for implementation was essentially a zero-crossing calculation. This process was given additional constraints, though. To ensure that the waveform's zero crossing is measured at the same place for each period, the slope of the signal is measured as well. Therefore, this process only measures the zero crossing when it crosses with a positive slope. The crossing was also not measured exactly at zero, but instead at a threshold slightly above zero. Doing this helps to eliminate the accumulation of error in the calculations due to noise. The value for the threshold was experimentally found to be 120mV. This threshold was close enough to zero to not be influenced by the odd shape of the waveform while still far enough from zero to not be affected by any underlying noise or contributions from adjacent strings. A sample waveform is shown in **Figure 15**.



	Value	Mean	Min	Max	Std Dev
2 Frequency	80.26 Hz	80.26	80.26	80.26	0.000

Figure 15. Sample waveform from one string on Roland GK-3.

This zero-crossing calculation also requires some measurement of time for it to have any relevance to the frequency. To do this, a timer module within the microcontroller was set to be roughly 40 kHz, thus defining the ADC sampling rate. This rate was chosen because it was much higher than the maximum frequency to be detected, 784Hz, while still sampling conservatively. It was also one of the timer rates easily obtained with a predefined pre-scaler. When a positive slope crossing of the threshold is detected, the microcontroller notes the time at which this occurs and stores that information. The next time it happens, it notes this time as well. These two times are then subtracted to define the period of oscillation. From this time difference, the frequency of oscillation is easily calculated. This frequency is the pitch of the note currently being played. All six microcontrollers are performing these calculations on their independent strings, gathering the pitch information for each. All six pass their calculated pitches to the next module, the Main Processor, using I^{2}C. From the pitches on each string, the chord being played can be determined. **Figure 16** shows the Level 2 Firmware Block Diagram for the overall pitch detector firmware design.

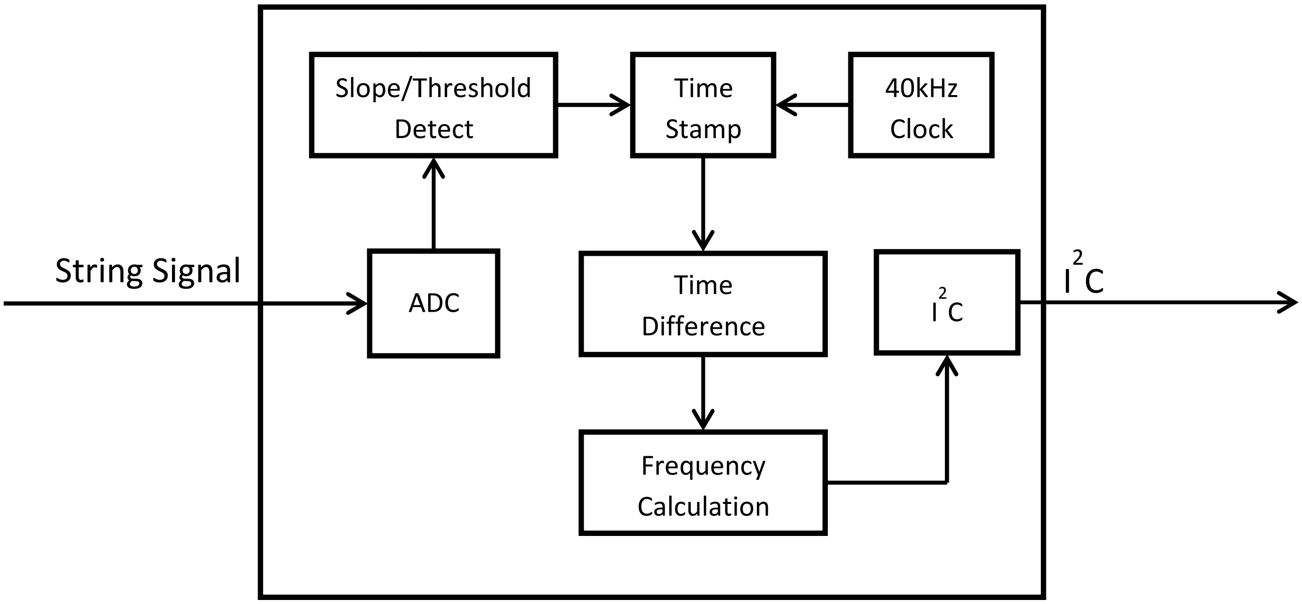


Figure 16. Pitch Detector Firmware Level 2 block diagram.

3.3 LED Module (JF)

The Level 2 block diagram of the LED Module can be seen in **Figure 17**. Notice that the LED Module contains two sub-modules, the LED Control Module and the LED Lighting Module.

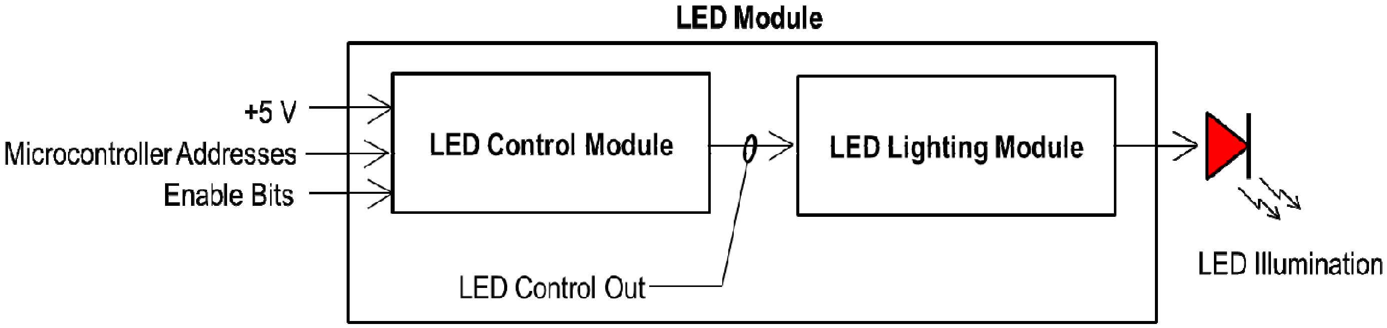


Figure 17. LED Module Level 2 block diagram.

The functional requirements are listed in the following tables:

Table 16. Functional Requirements for LED Control Module

Module	LED Control Module
Inputs	Power Module: +5 V DC

	Main Processor Module: Microcontroller Addresses 1-9 Main Processor Module:Enable Bits 1-3 from microcontroller
Output	Lighting Control Module: LED Control signal
Functionality	Receive Enable Bits and Microcontroller Addresses to decode which LED will be turned on.

Table 17. Functional Requirements for LED Lighting Module

Module	LED Lighting Module
Input	LED Control Module: LED Control signal
Output	Actuation of LEDs
Functionality	Illuminates a particular LED.

The LED Module, mounted on the neck of the guitar, is the means by which the guitar displays the notes that are required to complete the user-selected chord. Each illuminated LED represents a single note and indicates where the user needs to place his or her fingers to correctly play the chord. However, since for a given selection there is typically more than one way to play that particular chord (referred to as "voicings"), a GUI provides the user the option to cycle through different voicings. And so, for each toggle of the software interface, the appropriate LEDs illuminate to display the new voicing. Once the user is satisfied with his or her chord selection, the guitar strings are strummed and the software assesses if the chord was played correctly. The LED Control Module decodes the Microcontroller addresses and excites a particular LED in the LED Lighting Module.

3.3.1 Operation of the LED Control Module

Figure 18 shows the Level 3 block diagram of the LED Control Module, which consists of three 3-to-8 (also shown as "3-8") decoders. Two of the decoders are active-LOW (U1 and U2) while the third is active-HIGH (U3, details on the reasoning behind this selection is discussed below).

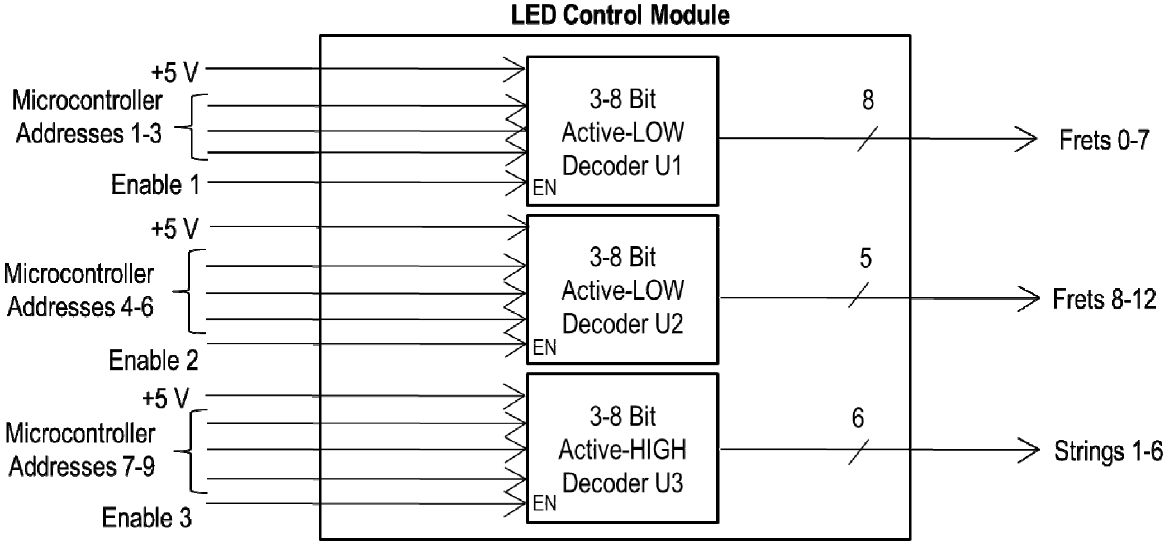

LED Control Module
+5 V
Microcontroller Addresses 1-3
Enable 1
3-8 Bit Active-LOW Decoder U1
EN
8
Frets 0-7
+5 V
Microcontroller Addresses 4-6
Enable 2
3-8 Bit Active-LOW Decoder U2
EN
5
Frets 8-12
+5 V
Microcontroller Addresses 7-9
Enable 3
3-8 Bit Active-HIGH Decoder U3
EN
6
Strings 1-6

Figure 18. LED Control Module Level 3 block diagram.

The functional requirement tables for the components in **Figure 18** are listed below.

Table 18. Functional Requirements for Decoder U1

Module	3-8 Bit Active-LOW Decoder U1
Input	Power Module: +5 V DC Main Processor Module: Microcontroller Addresses 1-3 Main Processor Module: Enable 1 Bit ("EN1")
Outputs	Cathode control of LEDs in frets 0-8
Functionality	Changes the state of LED cathodes along frets 0 to 8, inclusive, from HIGH to LOW.

Table 19. Functional Requirements for Decoder U2

Module	3-8 Bit Active-LOW Decoder U2
Input	Power Module: +5 V DC Main Processor Module: Microcontroller Addresses 4-6 Main Processor Module: Enable 2 Bit ("EN2")
Outputs	Cathode control of LEDs in frets 9-12
Functionality	Changes the state of LED cathodes along frets 9 to 12, inclusive, from HIGH to LOW.

Table 20. Functional Requirements for Decoder U3

Module	3-8 Bit Active-HIGH Decoder U3
Input	Power Module: +5 V DC Main Processor Module: Microcontroller Addresses 7-9 Main Processor Module: Enable 3 Bit ("EN3")
Outputs	Anode control of LEDs on strings 1-6.
Functionality	Changes the state of LED anodes along a string from LOW to HIGH.

The LED Control Module governs which LED in the LED Lighting Module is lit at a particular moment in time. Because of the high clock speed, the appearance of simultaneous illumination can be achieved by cycling the microcontroller addresses that represent a selected chord. Each decoder receives three microcontroller address inputs for a total of nine address signals from the microcontroller. Three additional microcontroller signals dictate the enable input of the decoders (EN1 for U8, EN2 for U9, and EN3 for U10). All three decoders will operate using +5V from the Power Module.

In order to fully understand how the LEDs in the LED Lighting Module are illuminated, consider the simplified LED matrix shown in **Figure 19**. In this example, a 6 x 8 LED matrix is controlled by two decoders, one active-LOW ("Fret" decoder) and the other active-HIGH ("String" decoder). The six outputs of the String decoder are connected to the anodes of the LEDs along a particular string. The eight outputs of the Fret decoder are connected to the cathodes of the LEDs along a particular fret. In an inactive state, wherein all LEDs are off, the outputs of the string decoder are all LOW and the outputs of the fret decoder are all HIGH.

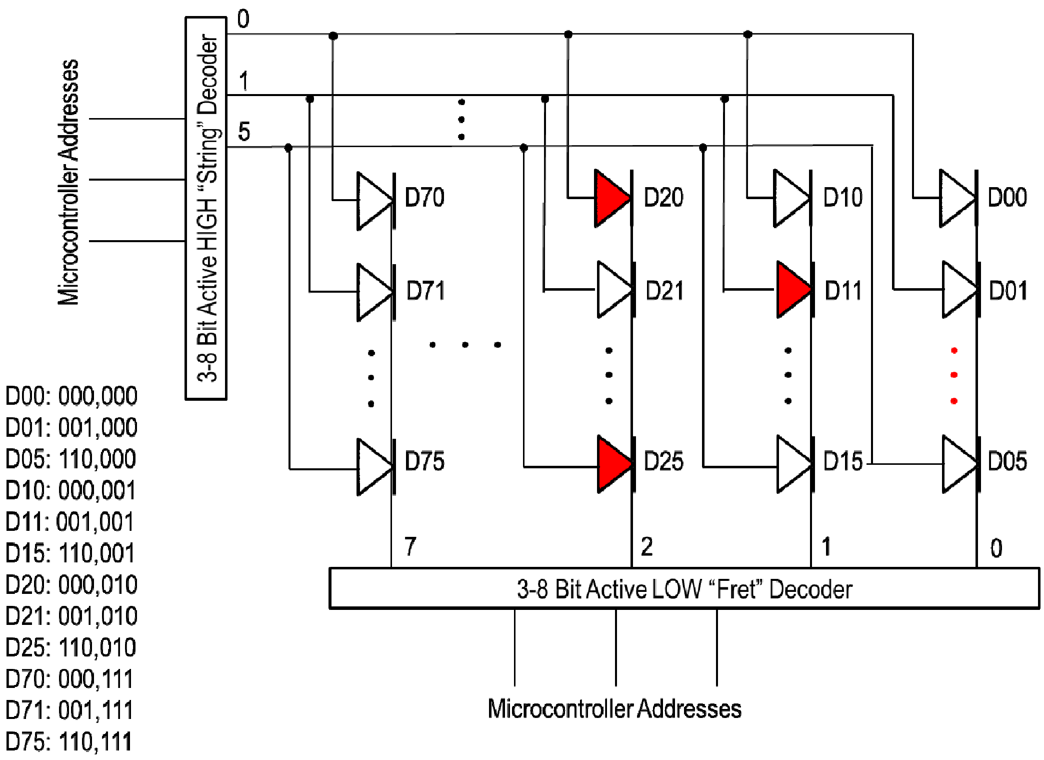


Figure 19. Sample LED matrix.

To excite a single LED, for instance, LED D00, the String decoder output must change from LOW to HIGH and the Fret decoder output must change from HIGH to LOW. These changes are enacted by passing a 3-bit address into each decoder, specifying the activation of String 0 in the String decoder and Fret 0 of the Fret decoder. Each LED in the matrix has a unique pair of addresses, as shown in the bottom left of **Figure 19**. The left set of three bits refers to the microcontroller address for the String decoder and the right set the address for the Fret decoder. For LED D00, the inputs to the String and Fret decoders must be 000 in order to light the LED.

Because guitar chords contain at least 3 notes, more than one LED must be illuminated at a time to fully represent the chord. While it is impossible to achieve true simultaneous illumination with this multi-decoder design, the appearance of multiple-lit LEDs can be achieved by rapidly switching the microcontroller address inputs, an action dictated by the microcontroller in the Main Processor Module. Consider the LEDs colored in red in **Figure 19** (D20, D11, D02, D03, D04, and D25). If these LEDs were selected to display a chord, the input addresses of the decoders would have to cycle through each of the unique address combinations for a given LED to display the whole chord. For instance, to change from lighting D20 to D11, the String decoder input will need to change from 000 to 001 and the Fret decoder input will need to change from 010 to 001.

To accommodate thirteen frets, two fret decoders must be used; namely U8 and U9. The enable inputs, EN1, EN2, and EN3, serve two purposes. First, EN1 and EN2 control which fret decoder is active for chords that span across frets 8 and 9. Because both U8 and U9 cannot be turned on at the same time (which would wrongfully display chords), the enable inputs allow for proper illumination of the LEDs. Second, all three enables will be programmed to turn off the decoders when switching between addresses.

3.3.2 LED Control Module Design

Figure 20 shows the schematic for the Lighting Control Module. The CD74HC138 decoders (seen as the top two decoders in the figure) will be responsible for controlling the states of the cathodes along the frets. The CD74HC238 decoder will manage the states of the anodes along the strings. In series with each string output of the HC238 is a single 301-ohm resistor. These resistors are used to control the amount of current flowing through the LEDs. This is further discussed in the next section, *3.3.3*, the LED Lighting Module Design.

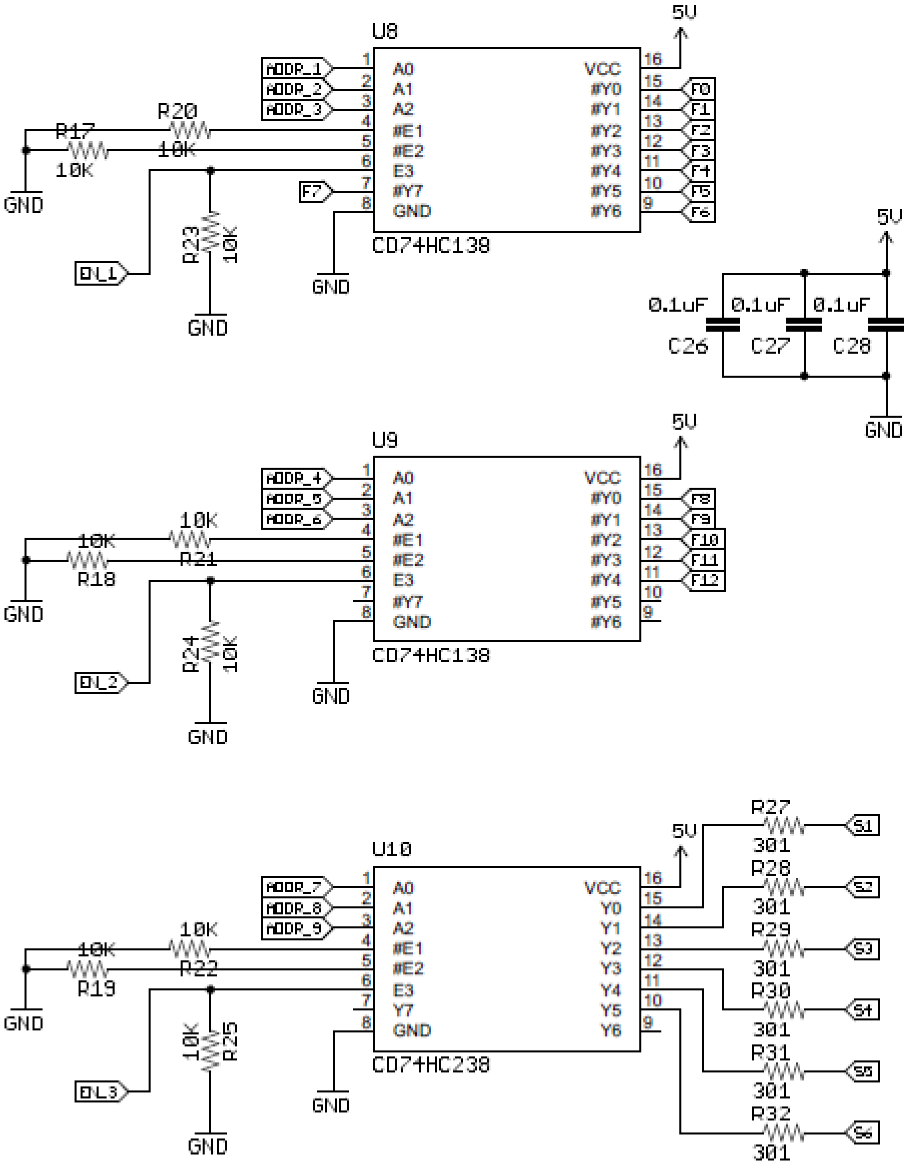


Figure 20. Lighting Control Module schematic.

3.3.3 LED Lighting Module Design

Figure 21 shows the full schematic for the LED Lighting Module. Every column of LEDs represents a fret on the guitar, as indicated by the numbers below each fret. Fret 0 refers to the fret closest to the headstock of the guitar.

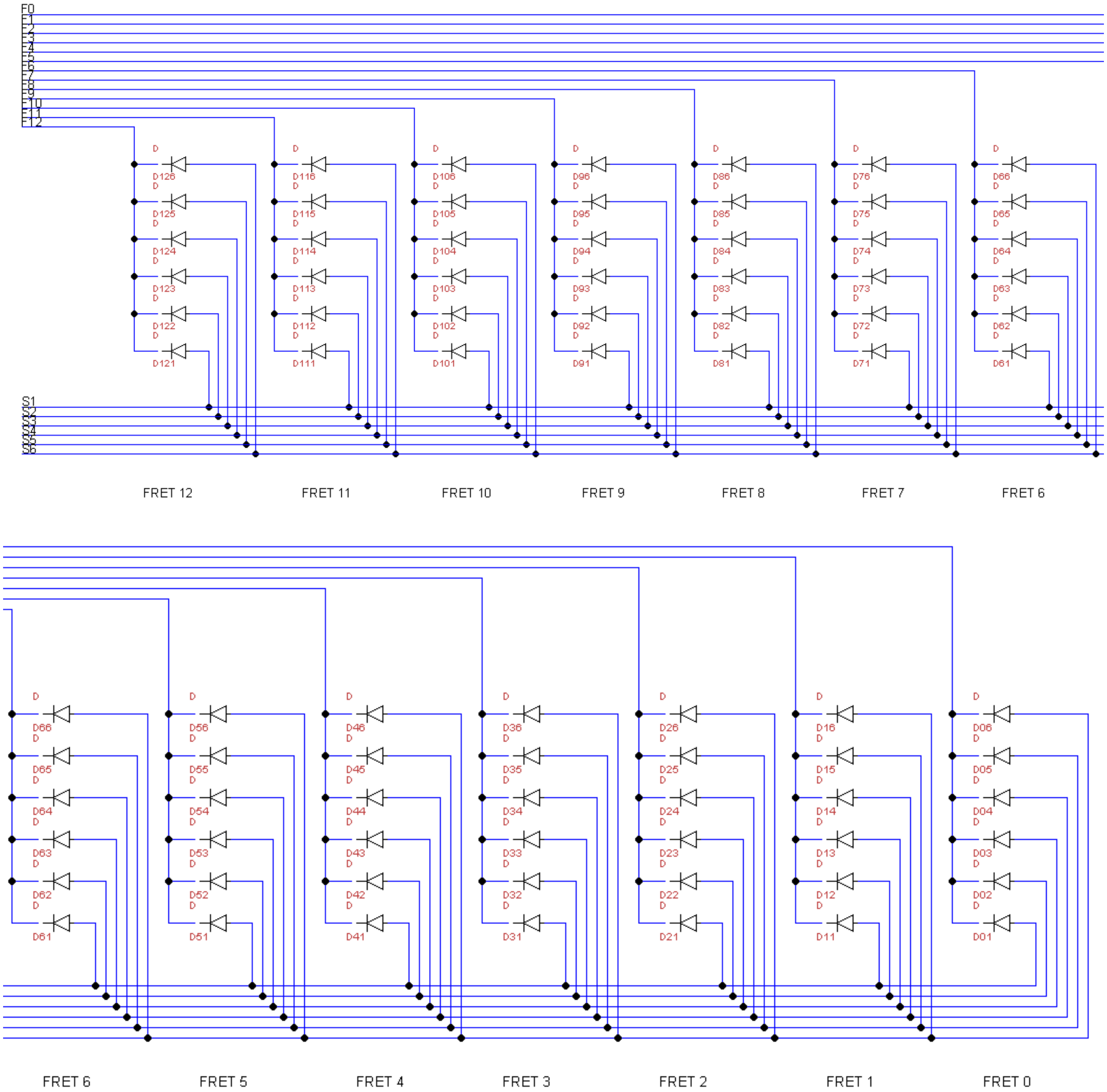


Figure 21. LED Lighting Module Schematic.

The LED Lighting Module was implemented as a flexible PCB (FCB), the board outline shown in **Figure 22**. Dimensions shown in **Figure 22** are in inches. The LED portion of the FCB will be placed on the "fretboard", the fretted fingerboard on the guitar neck. (Note that the FCB are unique to the guitar being used in the project and thus measurements of the frets will vary from guitar to guitar. Likewise, the string spacing also varies among guitars.) Note that at the top of the drawing of **Figure 22** there is the main spine that will be affixed to the back center of the guitar neck using an adhesive. The odd numbered frets are shown along the spine, starting from the bone nut on the left and proceeding toward the body of the guitar on the right. Stemming from the spine there are thirteen fret taps that will wrap around the neck to the fretboard and under the strings. Each tap will contain six LEDs, one for each note along that fret. Fret 0 will be placed before the first fret, however, the tap is alongside the bone nut (as opposed to the fret) of the guitar. The horizontal lines along each fret indicate the spacing of the strings. Each mark represents the approximate location of the string. These markings were used to determine the placement of the pads for the LEDs.

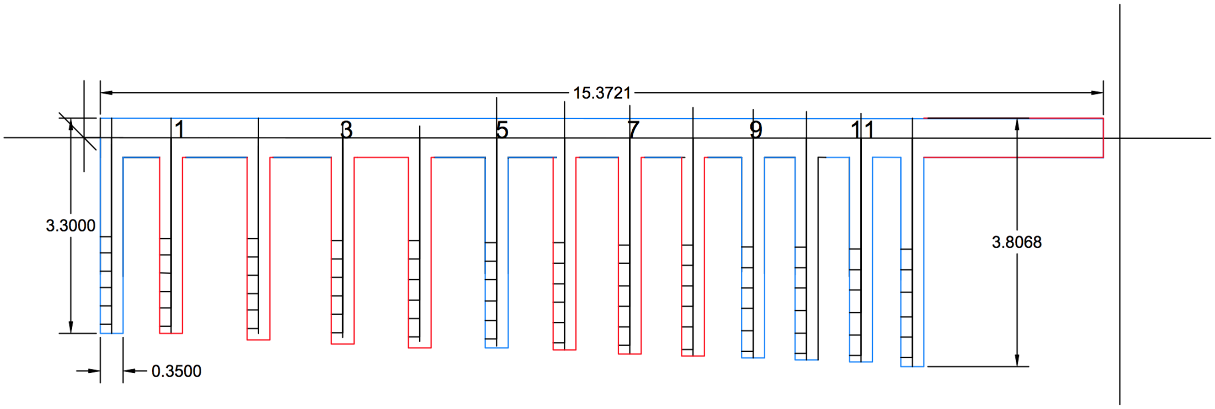


Figure 22. LED Lighting Module circuit board outline (Dimensions are in inches).

The diodes that are to be used for this project are 0603 Panasonic LNJ237W82RA High Bright Surface Mounting Chip LEDs (Refer to Appendix A). In order to prevent string buzz or dampening, the LEDs were required to be shorter than the height of the fret's bead, the portion of the fret that extends beyond the surface of the fingerboard. The standard size of the bead in a Stratocaster guitar is approximately 1.15 mm. With an overall height of 0.2 mm, these LEDs have a low enough profile to not interfere with normal guitar playing.

From the data sheet, the LEDs are rated at a forward voltage of 2.3V and a forward current of 20mA. Based on the chart for luminous intensity, the upper range lies between 10mA and 11mA. For the purposes of this design, a single current-limiting resistor will be placed along each anode (string) branch such that it provides roughly 10mA of current. Assuming the LED is driven by 5V and the drop in potential across the LED is about 2V, the value of the resistance can be approximated by the following relationship:

Here, the resistance *R* was found to be 300Ω. The nearest standard size for a chip resistor is 301Ω. Since each output of the 74HC238 will require the same amount of current, all six chip resistors will be of the same size and value.

Figure 23 shows the FCB layout of the Lighting Control Module. The string spacing information that was measured and drafted in **Figure 22** was applied to the LED spacing in the layout. The green lines represent the traces along the top layer. The red lines represent the traces along the bottom layer. The trace width was uniformly set to 8mil. At the body end of the guitar (shown as the top of **Figure 23**) is the pad for the connector that was selected for connection between the LED Lighting Module and the Guitar-Mounted Circuit Board. This receptacle connector will be the Hirose DF40C(2.0)-20DS-0.4V(51). Rated for 300 mA, this is more than sufficient to accommodate the 10 mA that the LED Lighting Module will draw. Interconnectivity is addressed in the following section.

The manufactured board is shown in **Figure 24**. Fret 0 can be seen as the leftmost fret tap. The gold colored spots are the copper pads onto which the LEDs are soldered.

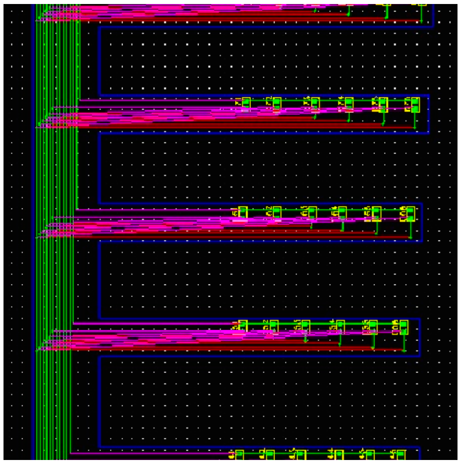


Figure 23. LED Lighting Module PCB layout.

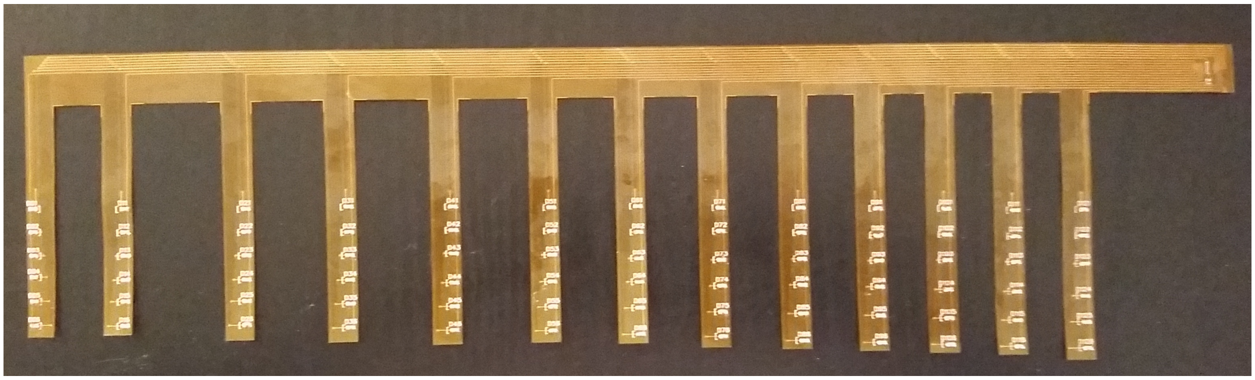


Figure 24. Post-production LED Lighting Module.

3.3.4 Interconnectivity with the Guitar-Mounted Circuit Board

A 20-conductor ribbon cable was the method chosen to connect the FCB to the Guitar-Mounted Circuit Board (GMBC). On the FCB end of this ribbon cable, the wires were split, stripped, and soldered to a small, two-layer printed circuit board. The PCB layout for this board is shown in **Figure 25**. Ten wires are soldered to the top layer of the board while the other ten wires are soldered to the bottom layer of the board. This was done as a means to reduce the size of the board. The traces on this board feed into the header, which in turn mates with the receptacle connector, X1, on the FCB. On the GMBC end of the ribbon cable, the wires were crimped to an insulation-displacement ribbon cable connector that attached to the 20-pin header on the main board. The post-production circuit board connector can be seen in **Figure 26**.

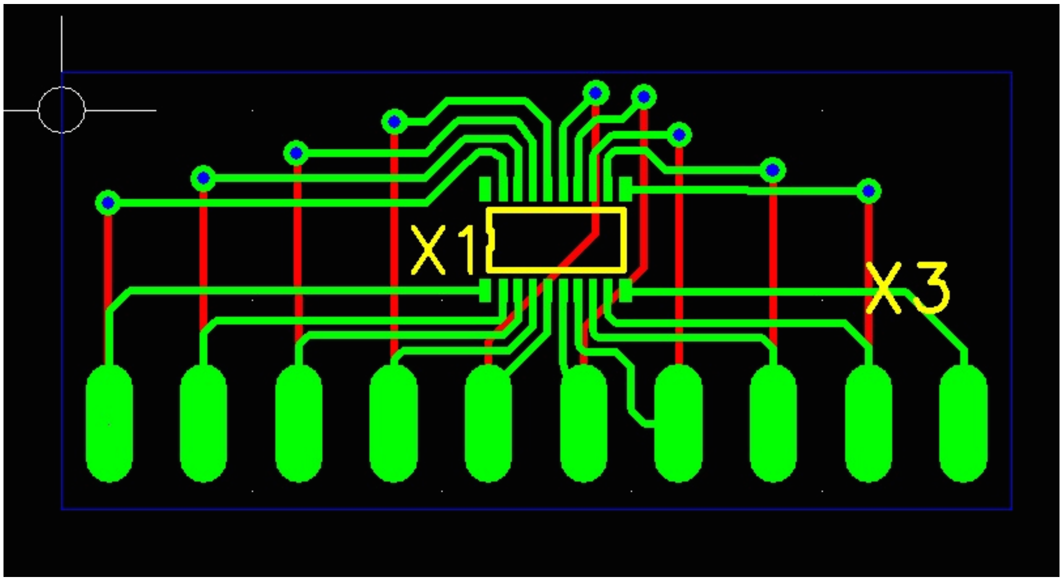


Figure 25. PCB layout for circuit board connector.

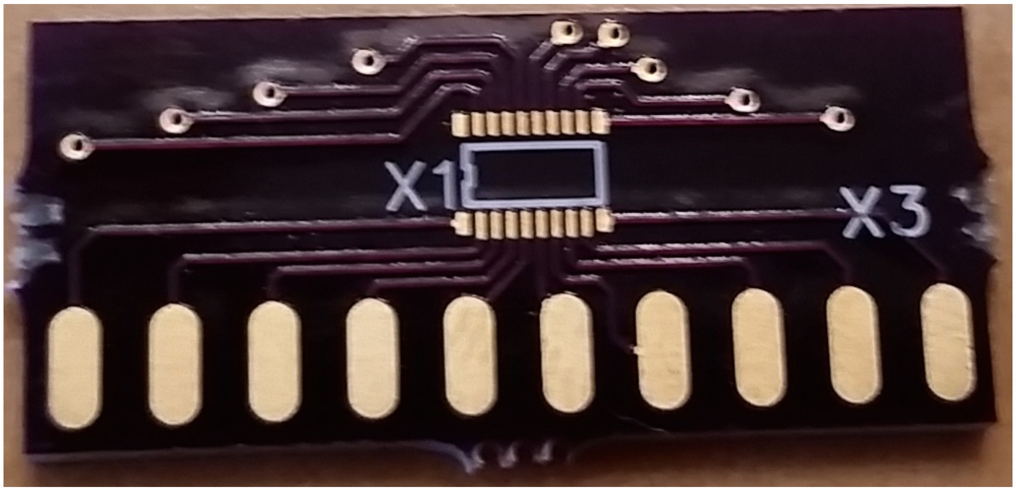


Figure 26. Post-production connector cable.

3.4 Main Processor Module (KR)

3.4.1 Level 1 Design

The Main Processor Module has three major responsibilities:

1.) It communicates via I^2C with the six microcontrollers that make up the Pitch Detection Module to gather chord information.
2.) It communicates this chord information to the software running on the PC, via USB.
3.) It controls the LED Module, where the finger placement of the chord is displayed on the neck of the guitar.

These responsibilities are shown in the Level 1 Block Diagram in **Figure 27**.

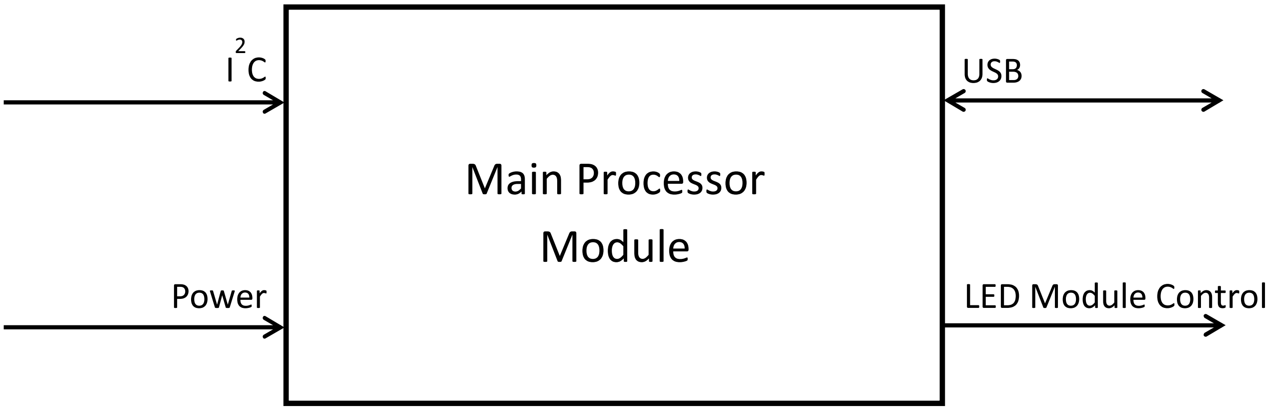


Figure 27. Main Processor Module Level 1 block diagram.

Table 21. Functional Requirements for Main Processor Module

Module	Main Processor
Input	I^2C: I^2C communication from Pitch Detection Module Power: Power from Power Module
Outputs	USB: USB communication to Software LED Module Control: Control signals to LED Module
Functionality	Communicates frequency information from Pitch Detection Module to Software. Receives LED lighting information from Software and controls the LED Module.

3.4.2 Level 2 Design

To fulfill the responsibilities outlined in the Level 1 Block Diagram, there are certain peripherals that the Main Processor Module must have. These peripherals are:

1.) A hardware I^2C module
2.) Means for USB communication
3.) GPIO to control the LED Module
4.) A processor to handle controlling all of these peripherals.

These requirements are shown in the Level 2 Block Diagram in **Figure 28**.

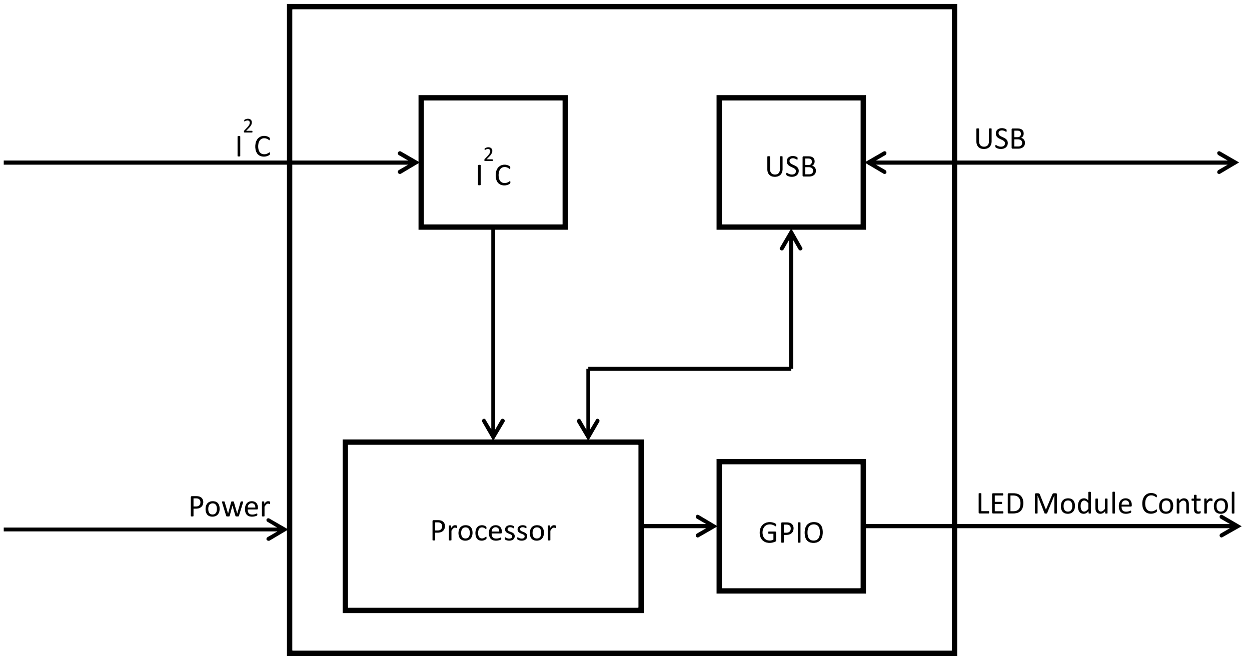


Figure 28. Main Processor Module Level 2 block diagram.

3.4.3 Components and Schematics

From the requirements outlined in the Level 2 design, it is clear that a microcontroller is needed. To this end, an Arduino compatible microcontroller, the Atmega2560, was chosen. This microcontroller has multiple hardware I^2C modules, 256kB of memory, and an abundance of GPIO. And even though it exceeds the needs for this application, Arduino compatibility was an important factor in selecting this microcontroller. Also important was the availability of a development board featuring this microcontroller, the Arduino Mega board, which would allow for firmware development before the final printed circuit board is constructed. This microcontroller does not feature a USB communication module, however.

To compliment this microcontroller and add USB communication capability, a Serial to USB IC was sourced. The chosen component for this is the FT232RL, an integrated circuit interfaced using UART, which enables serial communication over USB. As the Atmega2560 has multiple free UART modules, this was the most convenient option.

The full schematic for the Main Processor Module is shown in **Figure 29**. The Atmega2560 is designated as U7 and the FT232RL IC is designated as U14. Both of these ICs require a few other components to work. The Atmega2560 uses a 16MHz resonator denoted as Y7. It also has two 10kΩ pull-up resistors, R14 and R16, on the I^2C lines. A power indicator LED and current limiting resistor R15 of 330Ω are also used. The reset pin, like the I^2C lines, has a 10kΩ pull-up resistor R13. There is also 0.1µF capacitor, C20, separating the Reset net from the DTR net associated with the FT232RL. Doing this allows the Atmega2560 to be reprogrammed using the USB connection, making debugging much easier. Multiple 0.1µF capacitors are used on the many V_{CC} pins scattered across the 100-pin QFP package of the Atmega2560. The FT232RL also has one of these 0.1µF coupling capacitors on its VCC pin, as well as a 47µF tantalum capacitor to facilitate higher current usage when needed. There are also LEDs on the TXLED and RXLED pins of the FT232RL with pull-up resistors of 1kΩ. This is another feature included for debugging purposes, making transmit and receive actions visible to help troubleshoot communication issues.

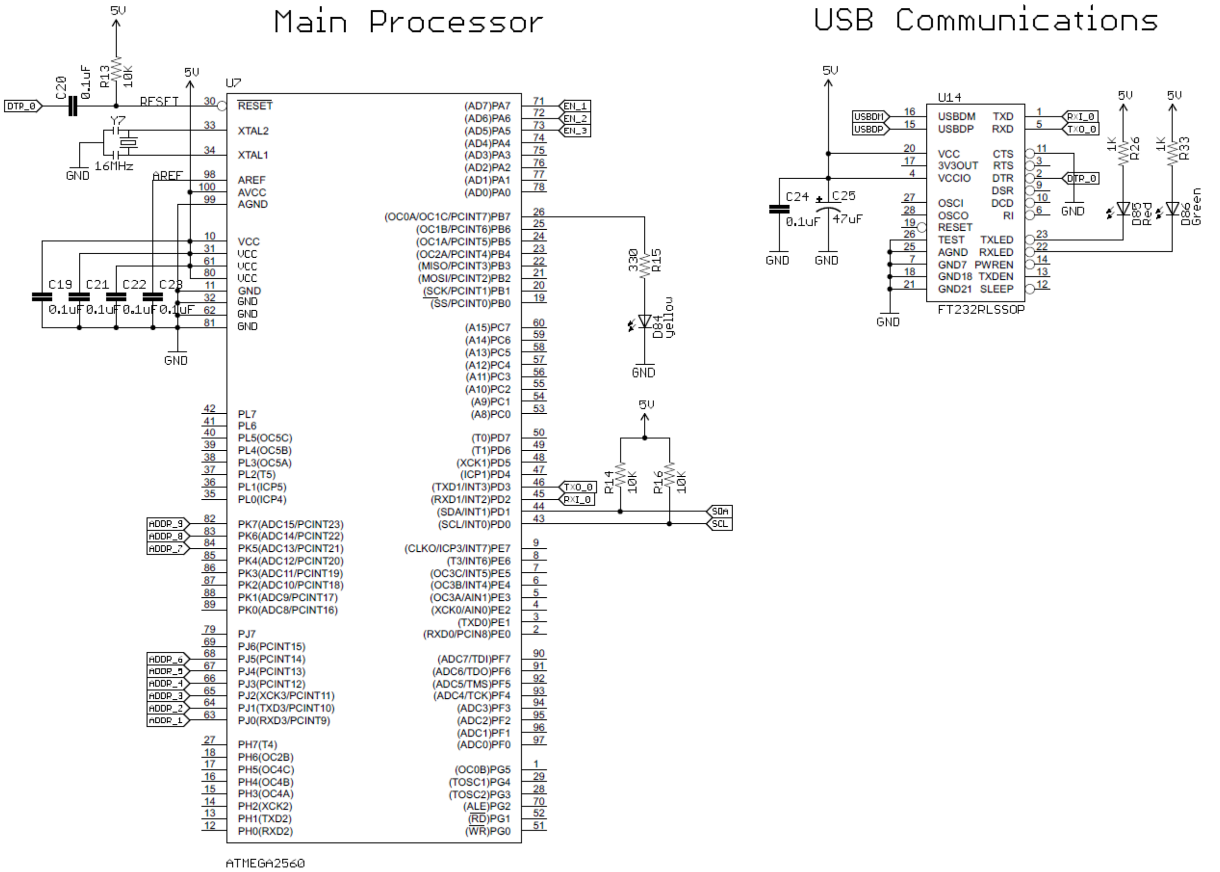


Figure 29. Schematic for Main Processor Module.

3.4.4 Layout

The layout for the Main Processor Module was straight forward. The FT232RL chip is placed near the Mini-USB connector and the Atmega256 is placed almost directly under it. The supporting passive components are placed near these components. This layout structure was chosen in an attempt to make the design as compact as possible. This also serves to reduce the trace length on the communication lines, making the design less susceptible to electromagnetic interference (EMI). The layout for the Main Processor Module is shown in **Figure 30**. A photo of the this section of the populated circuit board is shown in **Figure 31**. The full set of Gerber files is given in **Appendix D**.

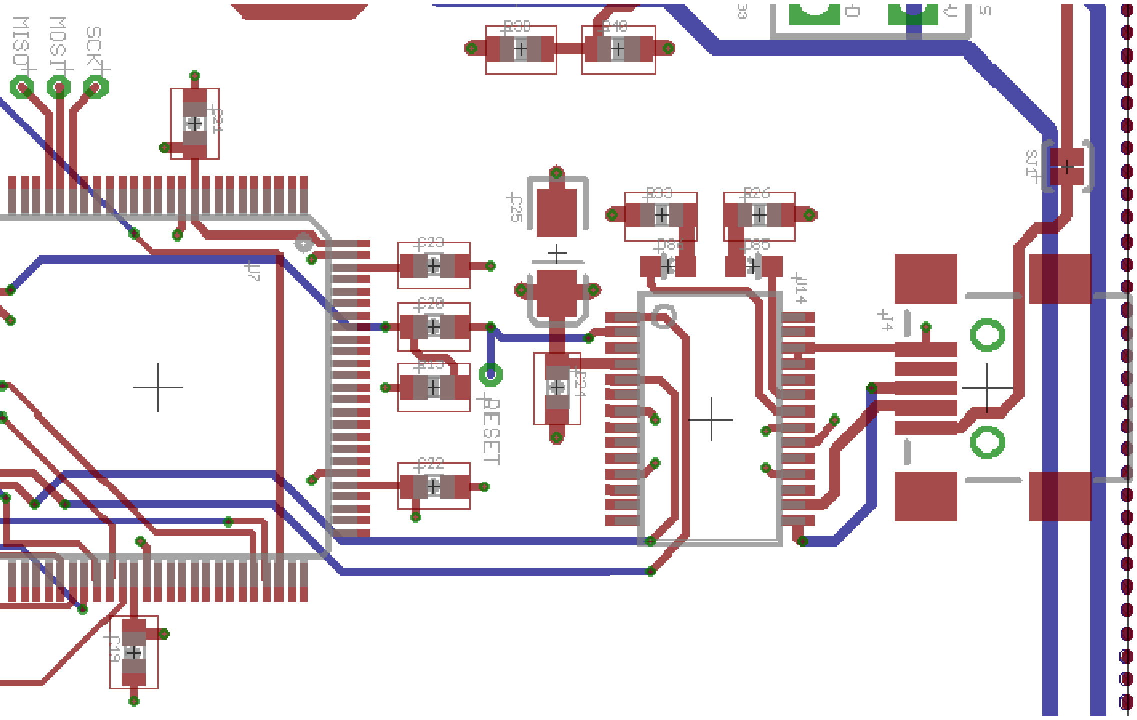


Figure 30. Main Processor Module Layout.

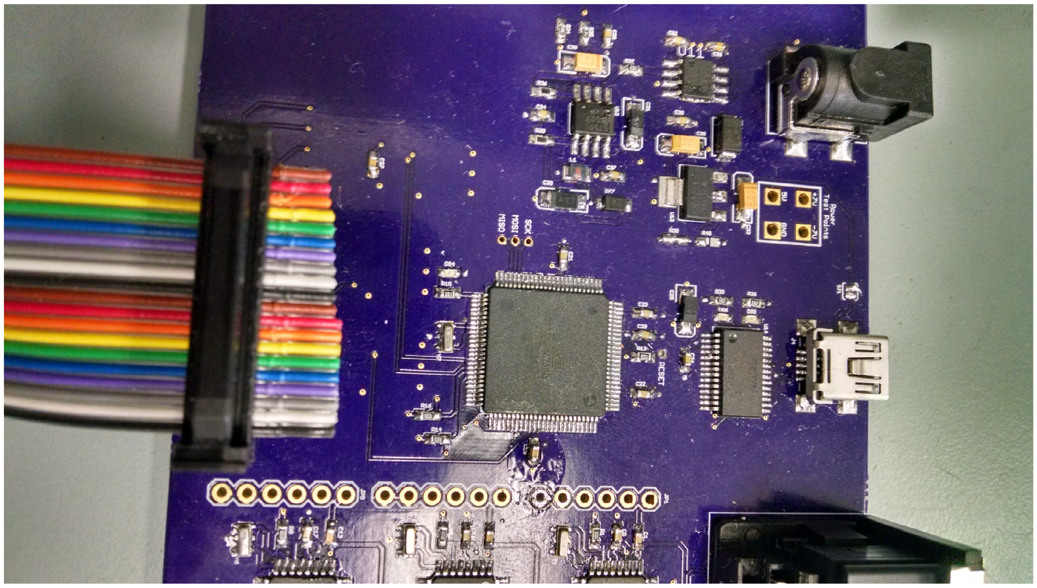


Figure 31. Main Processor Module populated PCB section.

3.4.5 Firmware Level 1 Design

As described in the hardware Level 1 Design, the Main Processor Module must be able to communicate by I^2C with the Pitch Detection Module, communicate with the software using USB, and control the LED Module using GPIO. While the hardware components to do this are already present in the design, they must be controlled by the programmed Firmware. For this reason, the Level 1 Firmware Block Diagram shown in **Figure 32** has these same input and output requirements.

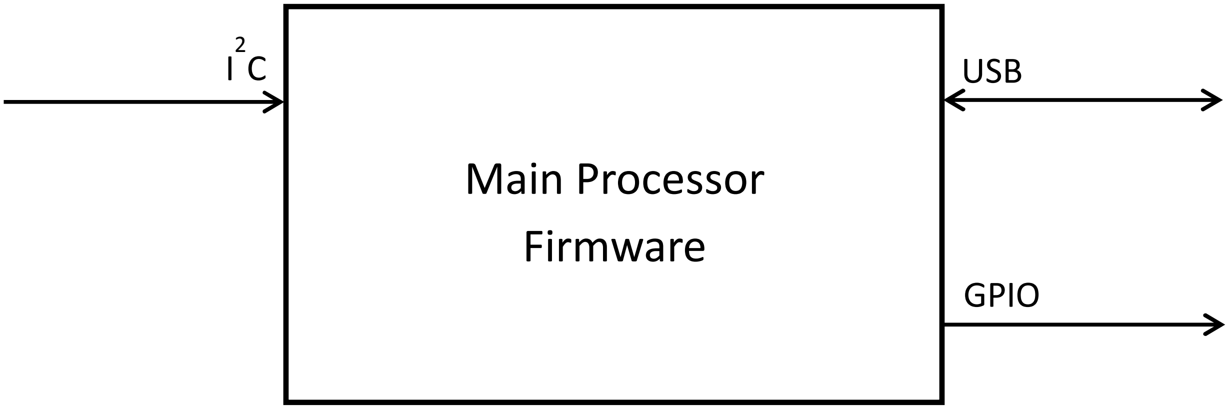


Figure 32. Main Processor Module Firmware Level 1 block diagram.

Table 22. Functional Requirements for Main Processor Module Firmware

Module	Main Processor
Input	I^2C: I^2C communication from Pitch Detection Module
Outputs	USB: USB communication to Software GPIO: Configured GPIO to control the LED Module.
Functionality	Communicates frequency information from Pitch Detection Module to Software. Receives LED lighting information from Software and controls the LED Module.

3.4.6 Firmware Level 2 Design

The Level 2 Firmware Block Diagram shown in **Figure 33** outlines how this is implemented in the firmware of this module. The *Frequency Communication* Sub-Module takes the I^2C communication from the Pitch Detection Module as an input and outputs the Software's required information using UART, which is translated to USB for the Software Module by the FT232RL hardware. The *Get LED Position* Sub-Module receives LED lighting information from the Software by USB to UART. It then outputs this lighting information to the *LED Position Control* Sub-Module. This module then manipulates the GPIO to control the LED Module.

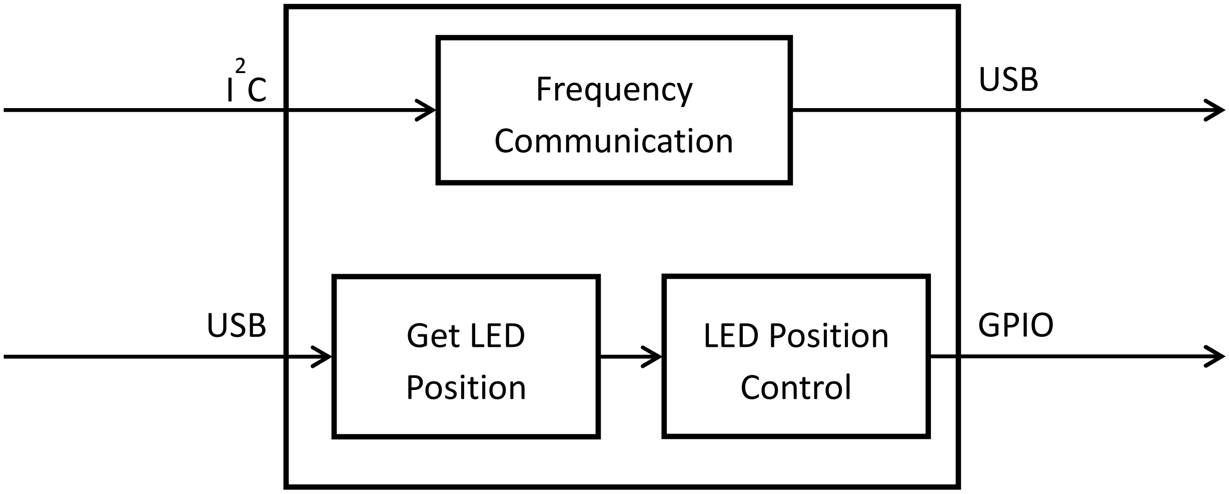


Figure 33. Main Processor Module Firmware Level 2 block diagram.

Table 23. Functional Requirements for Frequency Communication Sub-Module

Module	Frequency Communication
Input	I^2C: I^2C communication from Pitch Detection Module
Outputs	USB: USB communication to Software
Functionality	Communicates frequency information from Pitch Detection Module to Software in the form of an array of floating-point values.

Table 24. Functional Requirements for Get LED Position Sub-Module

Module	Get LED Position
Input	USB: USB communication from Software
Outputs	LED Position Control: Sends interpreted coordinates to LED Module.
Functionality	Receives LED lighting information from Software by USB, interprets the given coordinates, then passes the information to the LED Position Control Module for further use.

Table 25. Functional Requirements for LED Position Control Sub-Module

Module	LED Position Control
Input	Get LED Position: Receives interpreted coordinates from the previous module.
Outputs	GPIO: Sets the states of the GPIO.
Functionality	Receives coordinates from Get LED Position, then sets the GPIO accordingly—first the string pins, then the fret pins.

3.4.7 Firmware Level 3 Design

The functional requirements outlined in the Level 2 Firmware Design are implemented in the Level 3 Firmware Design shown in **Figure 34**. The Frequency Control Sub-Module is realized in a straight-forward manner. First, the Main Processor acquires all six floating-point frequencies from the Pitch Detection Module. It builds an array of the relevant information and waits for the Software to request this information. When prompted, it sends it to the Software using UART to USB starting with String 1, the lowest string on the guitar, and ending with

String 6, the highest. After the data has been acknowledged and processed by the software, the module discards the gathered information and waits for the next request for frequency information.

The Get LED Position Sub-Module is also a simple design. The Software communicates to the Main Processor Module that it will be sending the LED lighting position information and the Main Processor Module begins to read in this information. This information will contain only a fret number, but will be transmitted in a pre-defined order, from String 1 to String 6. This order helps to limit the amount of information that needs to be passed, as the string number is already known by the module. For example, if the third number read by the Main Processor Module is seven, then the Main Processor Module knows to turn on the LED corresponding to String 3, Fret 7. This position information is then passed to the LED Position Control Sub-Module for handling.

The LED Position Control Sub-Module is the simplest component of the firmware design. It receives the position information from the Get LED Position Sub-Module and then configures the GPIO accordingly by pulling the fret pin on the decoder HIGH and the string pin on the decoder LOW. To give the illusion that multiple LEDs are on at once, these states must be cycled, though. To do this, an intermediary state with all of the string pins pulled HIGH is used. This way, there is no potential difference that can cause current to flow through the LEDs as the GPIO that control the fret pins are being configured.

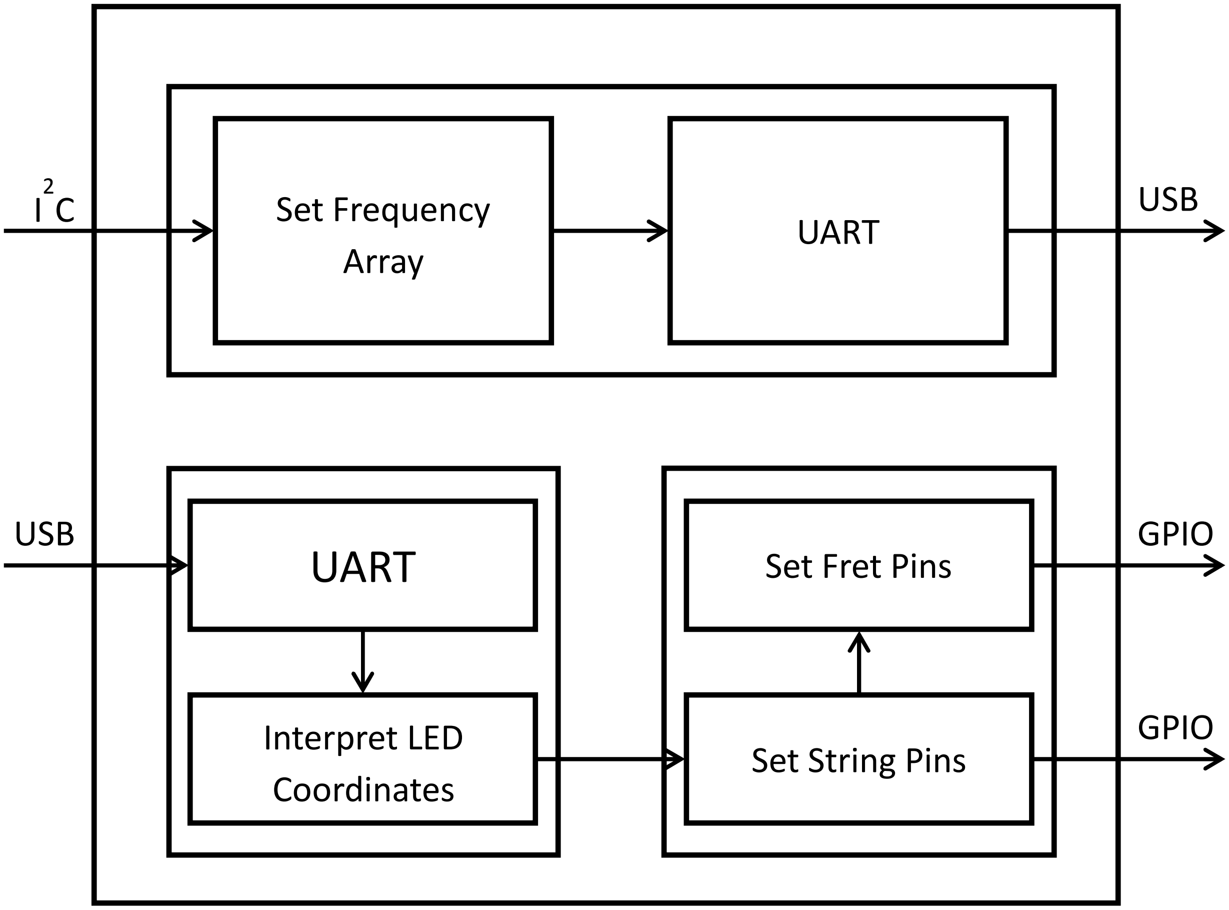


Figure 34. Main Processor Module Firmware Level 3 block diagram.

The Main Processor Module receives lighting information and frequency request information from the software using serial communication. This information is sent as six bytes in the form of ASCII characters. The order these bytes are received corresponds to string the byte carries information for. The lighting information is given using a hexadecimal numbering scheme where the character '0' is an open string, '1' is Fret 1, 'A' is Fret 10, and so on. To denote that a string should not be played, the character 'X' is sent for that string. For example, if the six bytes sent were '0230XX', the LEDs corresponding to open strings would illuminate on String 1 and String 4, Fret 2 would illuminate on String 2, Fret 3 would illuminate on String 3, and nothing would illuminate on String 5 and 6. To prompt the Main Processor unit to send frequency information, the software sends the six bytes 'JJJJJJ'. The main processor responds with six floating point numbers with a new-line character between each.

3.5 Software (JB)

The software is programmed in C++ and is represented by the block diagram shown in **Figure 35**. The overall functionality of the software is to determine what LEDs should be illuminated on the LED Module and pass this information to the Main Processor Module. The software also provides the user with a way to tune the guitar while giving feedback during the tuning process, allows the user to select a chord to be displayed by the LED Module, and determines if a chord has been played correctly while again providing feedback to the user.

There are three basic steps in the software flow. The first step is initialization, in which the classes and data structures needed to drive the system are instantiated and initialized. User input will be disabled until initialization has completed. The second step is tuning, in which the functionality is to provide the user with a way to select the desired tuning, tune the guitar to that tuning, and create a mapping of the notes on the guitar. The third step of operation is playing, in which the functionality is to allow the user to select a chord to play via a user interface, to determine where the notes that compose the selected chord are on the neck of the guitar, and to confirm if the user correctly played the selected chord via notifications on the user interface. These three steps or modes of operation will be referred to throughout section 3.5 to illustrate different roles the modules play during different stages in the application flow. The functionalities are summarized in **Table 26**.

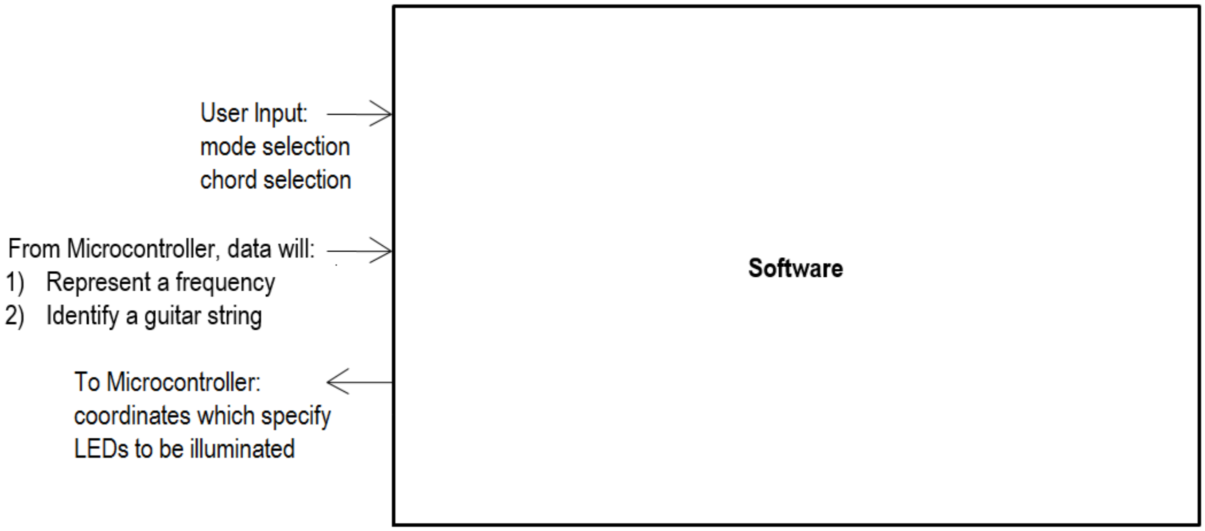


Figure 35. Level 1 block diagram of the software module.

Table 26. Functional Requirements for PC (Software)

Module	PC (Software)
Inputs	User input Main Processor Module: Note information
Outputs	Main Processor Module: Coordinates of LEDs to be turned on.
Functionality	Determine guitar tuning and chord placement on guitar neck.

3.5.1 Class Definitions (JB)

The functionality of **Table 26** will be achieved by five software classes: the Graphical User Interface (GUI), the Guitar class, the Note class, the Voicing class, and the Serial class. The User Interface and Guitar class are shown in **Figure 36**. Instances of the other three classes are used within the Guitar class but have been omitted from the diagram for simplicity.

The User Interface, designed using Qt Creator, provides the user with control of the software and feedback when playing. The Guitar class is responsible for establishing a connection to the Main Processor Module (via an instance of the Serial class) and for providing all the methods needed to drive the system (including those of the Voicing class). One instance of the Guitar class is created when the user starts the application. The Note class is simply a container for information of notes. Many instances of the Note class are created during run time, all of which are created and managed by the Guitar class.

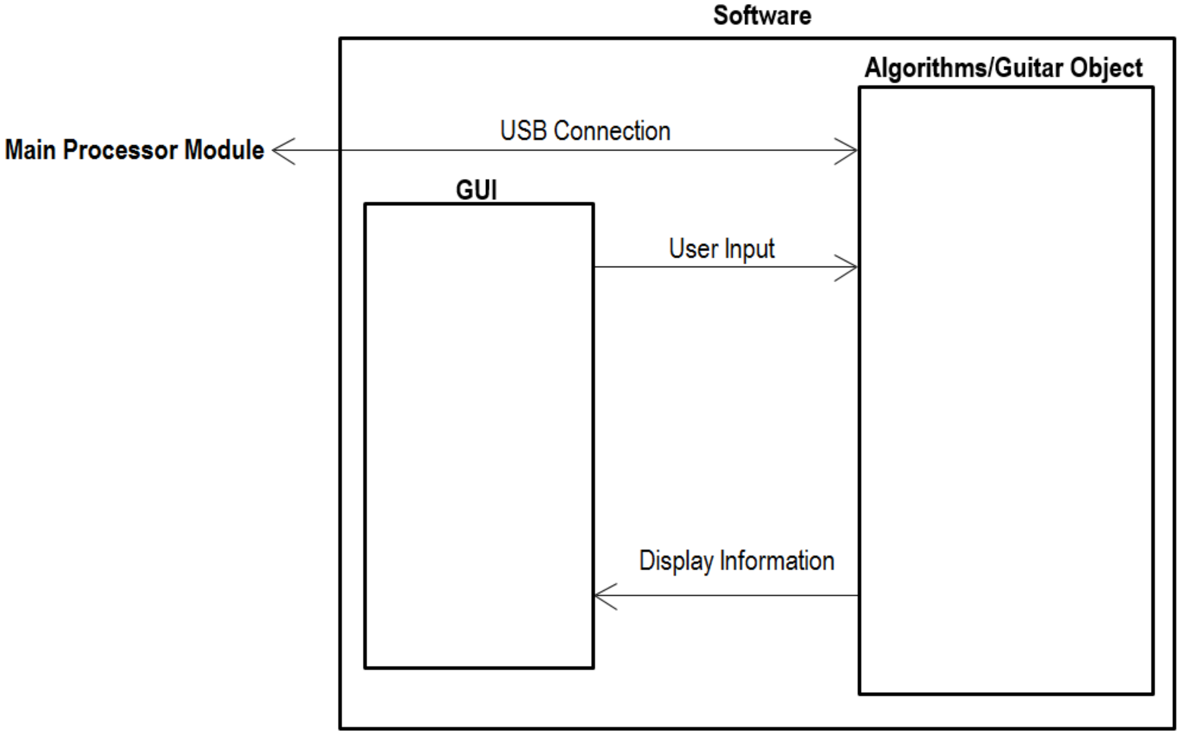


Figure 36. Level 2 block diagram showing classes within the software and their connections.

3.5.1.1 User Interface

The User Interface provides the user with control of the software and with feedback when the guitar is strummed. A screenshot of an example interface is shown in Appendix C. Control is provided by combo boxes (dropdown boxes) and buttons on the left of the interface. Feedback is provided by status messages on the right of the interface which display the expected and detected frequencies, along with an image of a guitar on the bottom of the interface which emulates the LED Module. The functionality of the User Interface is discussed in further detail in sections *3.5.2.1* and *3.5.2.8*.

Table 27. Functional Requirements for User Interface

Module	User Interface
Inputs	User Input: Mode selection and chord selection
Outputs	Guitar Object: Forward mode & chord selection information for processing User Display: Display emulation of LEDs
Functionality	Provide the user with control of the software.

3.5.1.2 Guitar Class

The Guitar class contains a majority of the data processing methods needed to control the system. On initialization of the software, a single Guitar object is instantiated (a single instance of the Guitar class is created in memory) on the PC which is responsible for establishing the USB connection to the Main Processor Module. This USB connection is a two way connection; note information (in the form of floating point frequency) comes from the Main Processor Module to the Guitar object, and coordinate information indicating what LEDs should be illuminated is sent back to the Main Processor Module from the Guitar object. To determine what lights should be illuminated, the Guitar object takes chord selection input from the User Interface and computes where the notes composing the selected chord are on the neck of the guitar.

Table 28. Functional Requirements for Guitar Class

Module	Guitar Class
Inputs	User Interface: Mode and chord selection Main Processor Module: Note information
Outputs	User Interface: Display information (coordinates) Main Processor Module: Coordinates of LEDs to be turned on
Functionality	Establish connection, process information, provide information to Main Processor Module and User Display

3.5.1.3 Note Class

The Note class is a container for information of each note relevant to the Guitar class. The Note class is used by the guitar class to determine if a note is in tune, map the notes that are available on the guitar, and determine if a chord was played correctly. On initialization of the Guitar object, an array of Note objects is created within the Guitar object by reading in the information for the notes from a text file. One line of the text file contains the information for a single note (separated by spaces), and every possible note is represented by exactly one Note object in the array. This exhaustive list of notes will be referred to in future sections as the *allNotes* array. The information in these Note objects includes:

- Note full name (e.g C#4)
- Note name (e.g. C#)
- Integer representation of the note (e.g. 1)
- Exact floating point frequency (e.g. 277.2)
- Minimum and maximum frequency for note to be considered in tune
- Minimum and maximum frequency that is considered within the threshold of this note (e.g. the frequency range that represents C#4, the cutoff frequencies between C#4 and C4 on the low end and between C#4 and D4 on the high end)

3.5.1.4 Voicing Class

The Voicing class is required to find all the different ways a single selected chord could be played given the tuning of the guitar. These permutations of a chord are known as voicings, and there can be as many as 30 or more for any given chord. A single instance of the Voicing class is used within the Guitar class to compute all the voicings for the currently selected chord. The Voicing class is discussed in more detail in section *3.5.2.5*.

3.5.1.5 Serial Class

The Serial class contains the code needed to read from and write to the Main Processor Module. One instance of this class is created within the Guitar class to provide the Guitar class with a communication channel to the microcontroller. The hardware is controlled by writing specific sequences of characters to the Main Processor Module which are detected to determine the correct action. For example, a specific control string is written to the Main Processor Module to update the sampled frequencies, and other control strings correspond to lighting patterns for the LED Module. The character sequences for controlling the LED Module are discussed in more detail in section *3.5.2.7*.

3.5.2 Algorithms and Methods (JB)

The primary software modules/algorithms required to provide the software functionality are shown in the Level 3 block diagram in **Figure 37**. The operation of each of these modules is discussed in the following subsections. Specific methods are described as needed to provide a clear understanding of exactly how each of the functionalities is achieved.

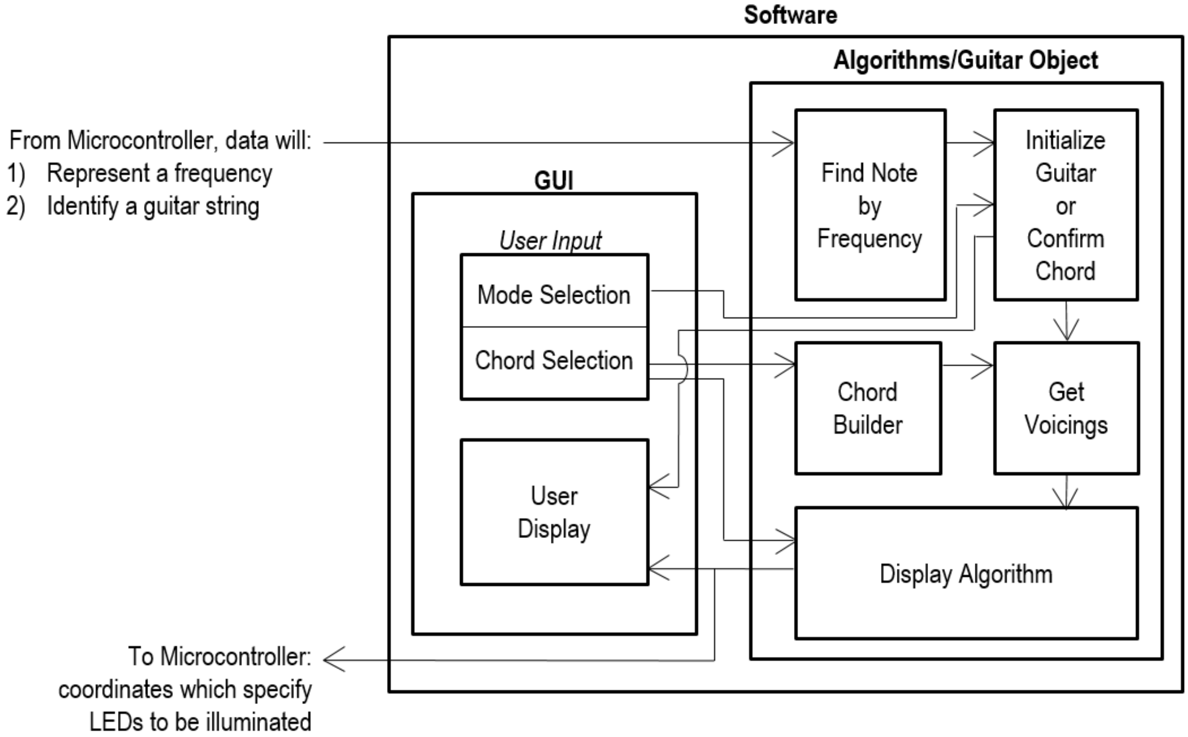


Figure 37. Level 3 block diagram showing primary functions within the classes.

3.5.2.1 User Input

There are three sets of controls which the User Interface provides. The first set of controls is a set of six combo boxes (one for each string) beside each of the string status text boxes on the right side of the interface. These combo boxes allow the user to select the desired tuning of the guitar. This information is passed to the Guitar object so that a mapping of notes can be created.

After the user selects their desired tuning, the second set of controls can be used for tuning the guitar and confirming if chords are play correctly. This second set of controls consists of a panel of four buttons at the top left of the interface. The tuning button is used to sample the frequencies detected and provide tuning feedback, and the clear tuning button simply clears the sampled frequencies. The confirmation button is used to sample frequencies detected and provide feedback as to whether or not the samples detected match those expected for the selected chord to be considered correct. The clear button is used to clear the sampled chord confirmation frequencies. The user can use these controls at any time to seamlessly tune/re-tune the guitar and confirm chords

The final set of controls provides the user with a way to select chords to be displayed and confirmed. These controls consist of dropdown boxes to specify a chord to display and buttons to cycle the previous/next voicing of the selected chord. When the voicing selection changes, the emulator at the bottom of the interface is adjusted to reflect the newly selected voicing and the information is also passed to the LED module to drive the display on the guitar.

3.5.2.2 Find Note by Frequency

The Find Note by Frequency Module identifies what notes are represented by the floating point data being passed in from the Main Processor Module. This is done by iterating through the allNotes array of the Guitar object. For each note, the frequency passed in is compared with the minimum and maximum frequency information stored in the Note class to determine if the frequency represents the current note. If the frequency passed in is not within the range of the note, the software iterates to check the next note. This information is then passed to the Initialize Guitar or Confirm Chord module so that the mapping of notes on the guitar can be initialized and statuses can be sent to the User Interface.

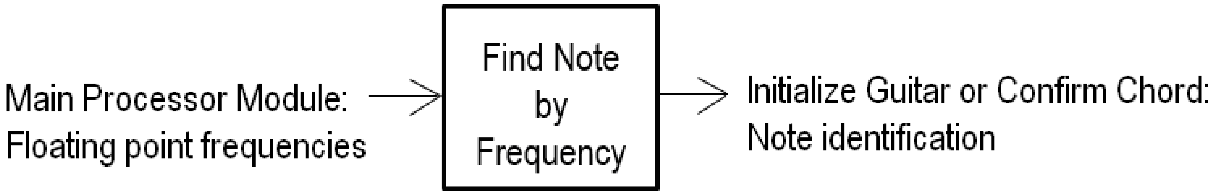

Main Processor Module:
Floating point frequencies
Find Note
by
Frequency
Initialize Guitar or Confirm Chord:
Note identification

Figure 38. Level 4 block diagram of Find Note by Frequency module

Table 29. Functional Requirements for Find Note by Frequency

Module	Find Note by Frequency
Inputs	Main Processor Module: Floating point representation of note frequency
Outputs	Initialize Guitar or Confirm Chord: Note identification
Functionality	Given a frequency from the Main Processor Module, identify the note and determine if it is in tune

3.5.2.3 Initialize Guitar or Confirm Chord

The functionality of this module depends on the mode of operation. When tuning, this module creates a mapping of the notes on the guitar. When the software is in playing mode, this module confirms whether or not the selected chord has been played correctly.

The mapping of notes on the guitar is achieved by the Tuning Table, a two dimensional array of Note objects with dimensions 6x13 (6 strings x 12 frets + open string). The tuning process starts with six floating point frequencies coming from the Main Processor Module, one frequency from each of the six strings. These frequencies come from the user strumming the guitar's open strings (i.e. not holding down any of the frets) and represent how the guitar is tuned. The notes represented by the frequencies are compared to what they should be given the selected tuning, and the information is used to set the first note in each of the six arrays representing the separate strings on the guitar. An example is illustrated in **Figure 40**, where the second string of the mapping is being initialized.

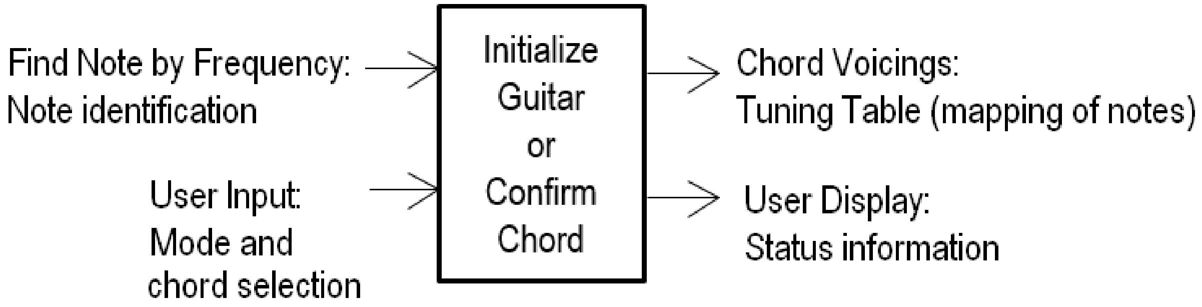


Figure 39. Level-4 block diagram of Initialize Guitar or Confirm Chord module

Table 30. Functional Requirements for Initialize Guitar or Confirm Chord

Module	Initialize Guitar or Confirm Chord
Input	Find Note by Frequency: Note identification User Input: Mode selection
Outputs	Chord Voicings: Mapping of notes on neck of guitar User Display: Tuning status or confirmation of chord
Functionality	In tuning mode, this module creates a mapping of notes on the guitar to be used by the Chord Voicings module. In playing mode, this module provides statuses to the User Interface indicating if the chord was played correctly.

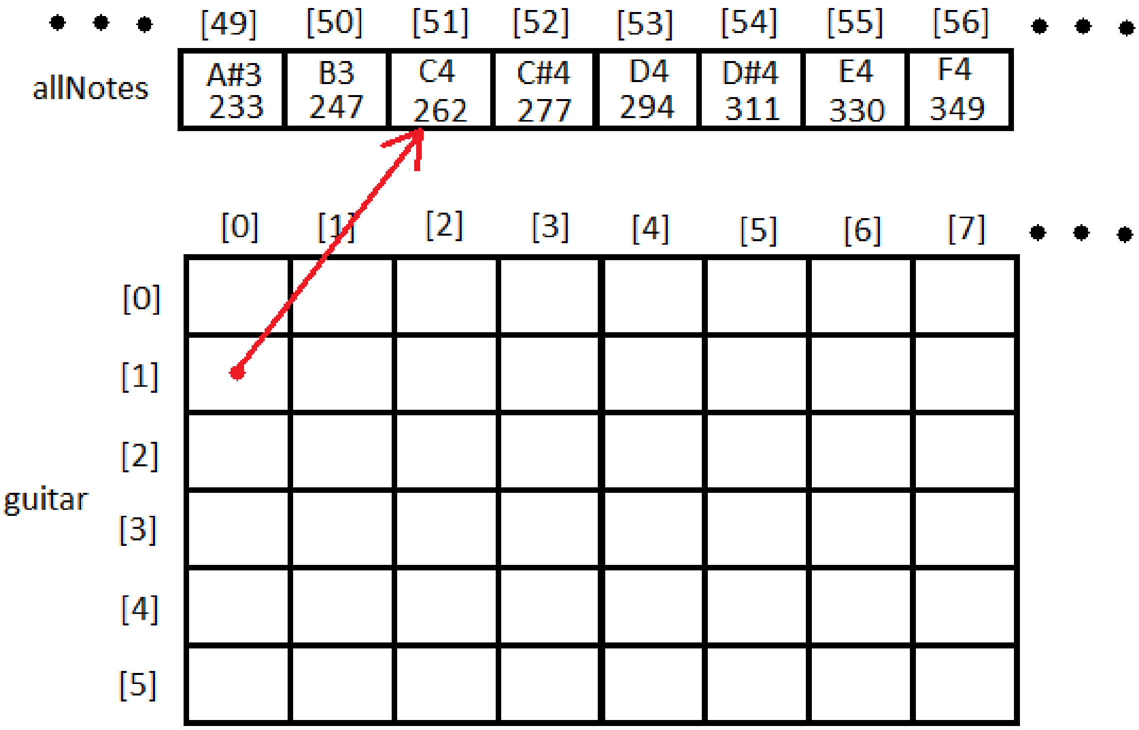


Figure 40. Mapping of notes in memory

Once the first note of the mapping has been determined for a given string, the rest of the mapping for that string can be initialized by iterating through the all notes array, setting each consecutive note of the Tuning Table to the next note in the allNotes array. This process is illustrated in **Figure 41**.

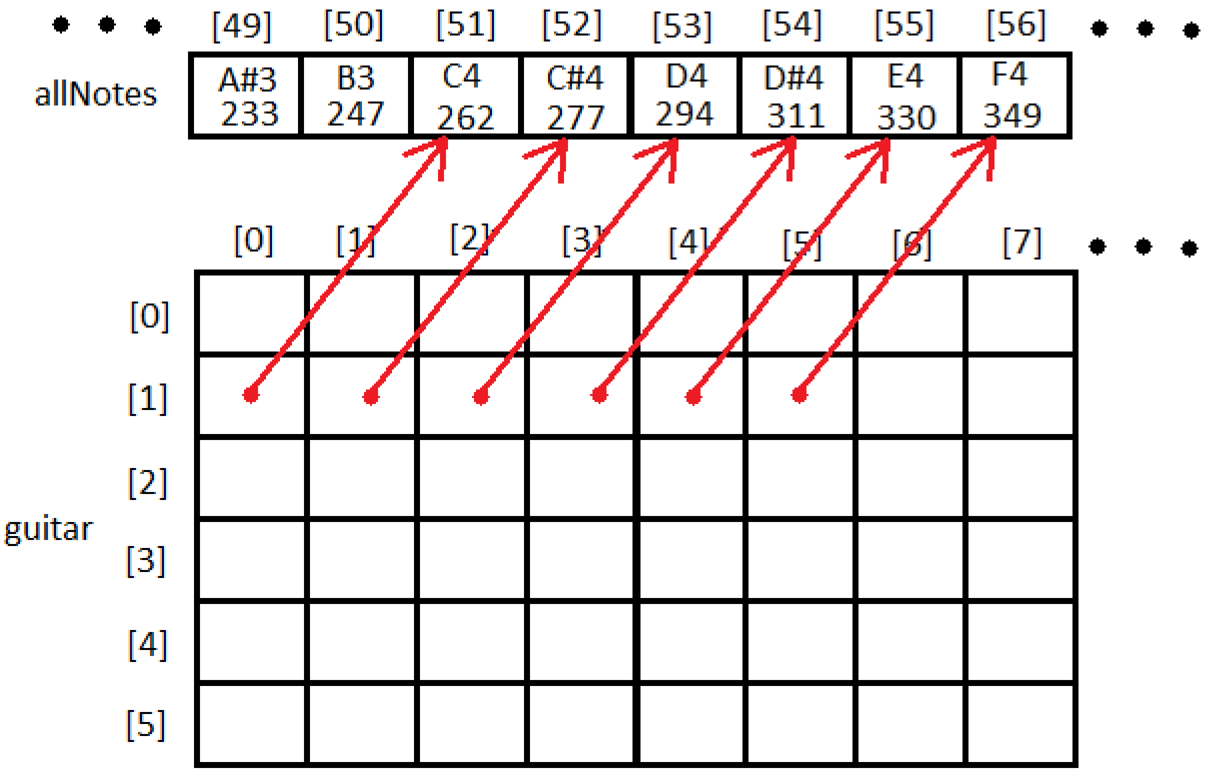


Figure 41. Mapping of notes in memory

Once the mapping of notes has been created successfully and the user switches to playing mode, this module simply determines if the selected chord has been played correctly. This is done using the same frequency information passed from the Main Processor Module through the Find Note by Frequency module. However, rather than using the information to initialize the mapping, the notes identified by the Find Note by Frequency module are matched to the notes which compose the selected chord to confirm if the chord was played correctly. The confirmation status is sent back to the User Interface to provide feedback. A status of chord correct is reported if the notes represented by these frequencies are an exact match to the notes of the selected voicing. A status of chord partially correct is reported if not all the notes match, but all there is at least detected note which corresponds to each of the notes that make up the selected chord. Examples of chords being confirmed as correct or partially correct are shown in Appendix C.

3.5.2.4 Chord Builder

The Chord Builder determines what notes compose the chord selected by the user. The chord selection information is translated into information that the Chord Voicing module can interpret and use. An array representing the selected chord is created; the array contains an integer representation of each note in the selected chord. (The integer representation of the notes is used for comparison efficiency since many comparisons are needed, and integers can be compared faster than strings.)

The combo boxes on the User Interface have an integer associated with them which represents what option is currently selected. The integer from the chord type selection dropdown box is passed to a master chord builder method, which then calls the correct chord builder method (e.g. chordBuilder_minor will be called if the user has selected a minor chord). These separate Chord Builder methods take the integer from the scale selection dropdown box to determine the first note of the chord and then determine the rest of the notes that compose the chord. The integer representation of the notes that compose the selected chord is stored in an array and passed to the Chord Voicings algorithm to determine where the notes are on the neck of the guitar.

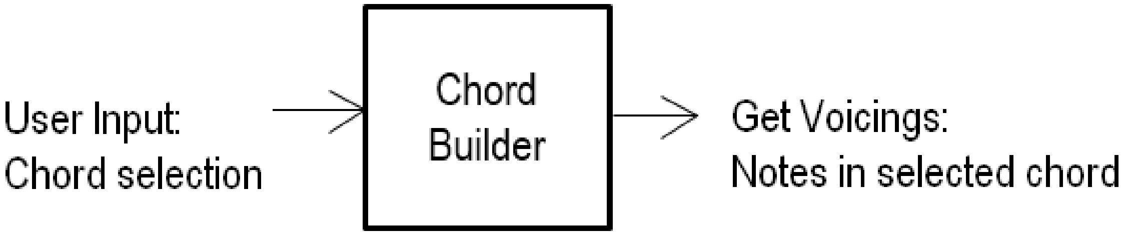


Figure 42. Level 4 block diagram for Chord Builder module

Table 31. Functional Requirements for Chord Builder

Module	Chord Builder
Inputs	User Input: Chord selection
Outputs	Chord Voicings: Integer representation of notes that compose the selected chord
Functionality	Turn the chord selection information from User Input into a representation that can be understood and used by the Chord Voicings algorithm

3.5.2.5 Chord Voicings (MB)

The Chord Voicings Module is comprised primarily of the chord voicings algorithm. This algorithm processes the many different tunings to which a guitar may be set, as well as the many different chords that can be voiced to that particular tuning. This algorithm also accounts for the fact that the human hand limits how far across the frets a chord may be played.
The Level 1 block diagram for the Chord Voicings Module is shown in **Figure 43**.

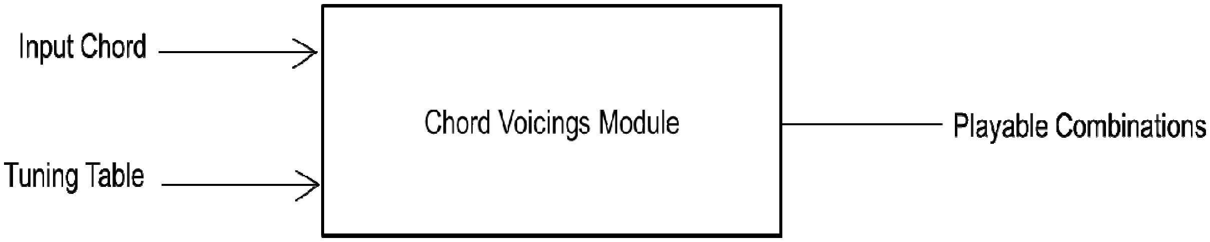


Figure 43. Chord Voicings Module Level 1 block diagram.

Table 32. Functional Requirements for Chord Voicings Module

Module	Chord Voicings
Inputs	Input Chord, Tuning Table
Outputs	Vector of coordinate pairs – Playable Combinations
Functionality	Processes all possible chord combinations, given an input chord and a tuning table. Each combination is sent through a validation process.

The goal of the chord voicings algorithm is to determine all possible "playable" combinations for an input chord of variable size, for a particular tuning. A state-space representation is used to process 4x6 sub states of the tuning table. 4x6 is used to remove any combinations farther than four frets because of the length of the human fingers. A sliding 4x6 window is iterated through the tuning table which is 13x6. For each 4x6 within the 13x6, the program generates a sub state of the tuning table. This allows the program to search for combinations within four frets. The Level 2 block diagram for the Chord Voicings Module is shown in **Figure 44**.

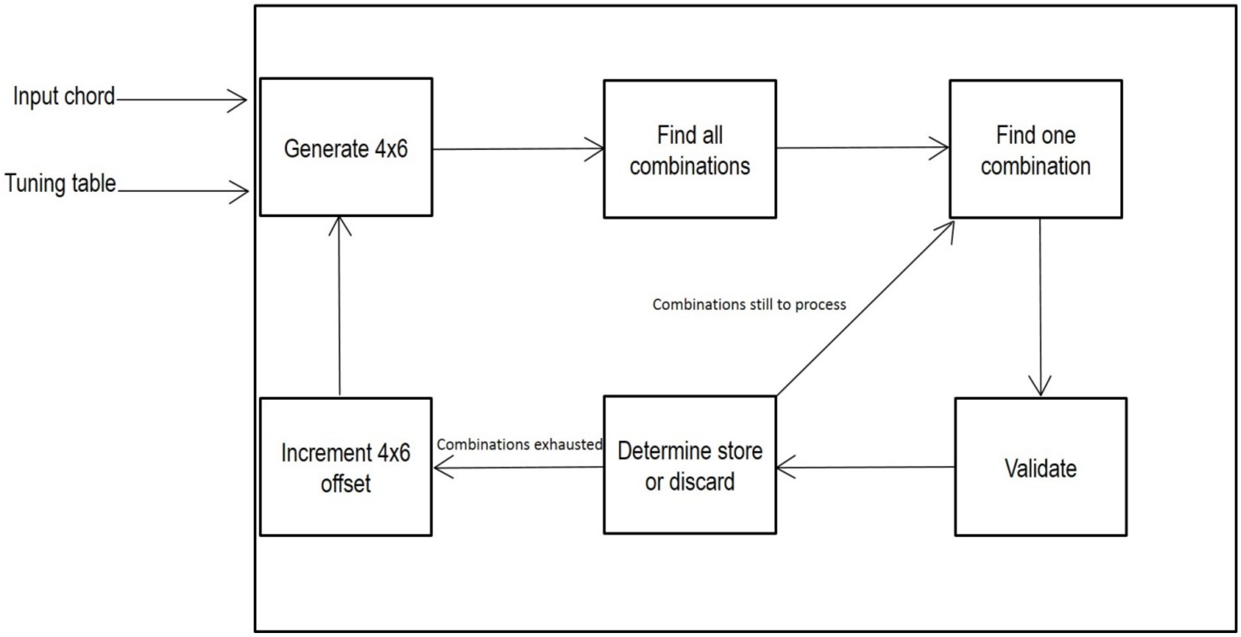


Figure 44. Chord Voicings Module Level 2 block Diagram.

Table 33. Functional Requirements for the “Generate 4x6” module

Module	Generate 4x6
Input	Tuning table, Input chord
Output	4x6 offset
Functionality	Generates a 4x6 offset of the 13x6 tuning table

Table 34. Functional Requirements for the “Find all combinations” module

Module	Find all combinations
Input	The current 4x6 state
Output	All chord combinations to be validated
Functionality	Nested loops to determine the sets of combinations

Table 35. Functional Requirements for the “Find one combination” module

Module	Find one combination
Input	Current combination being processed by the nested loops
Output	One combination that has not been validated
Functionality	Iteratively validate the combinations within the 4x6, one combination at a time.

Table 36. Functional Requirements for the “Validate” module

Module	Validate
Input	One combination
Output	True or False
Functionality	The output determines whether to store, or discard a combination.

Table 37. Functional Requirements for the “Determine store or discard” module

Module	Determine Store or Discard
Input	True or False
Output	A stored or discarded combination
Functionality	The execution of the validation module. This decides whether or not to place the combination in question into the vector.

Table 38. Functional Requirements for the “Increment 4x6 offset” module

Module	Increment 4x6 offset
Input	There really isn’t an input.
Output	Offset variable +1
Functionality	Increases the offset by one as long as there still is a 4x6 sub state to process. Otherwise it the processing algorithm for that tuning and input is completed.

This module will use the information of the notes held within the current 4x6 offset, which will be used to process the data. A six-nested for-loop is used to process every single combination of 4x6. Each time a combination is found, it is sent to the validation module.

The nested six-loop must be re-arranged so that combinations are not missed. Each of the six loops corresponds to one of the six strings. Counters for each row are used to store how many notes are in that row. Each of the loops is rearranged in ascending order so that the largest loop is on top. The program stores which loop corresponds to what string so that if the chord is validated, it can be stored in order. This is done so that when the Main Processor Module processes the coordinates, it only has to recognize the X coordinate. Since they are sent in order, the Y coordinate would be known.

The "Generate 4x6" Module of **Figure 44** generates a 4x6 portion of the tuning table. Only the positions of the tuning table that match the input chord are used to process the combinations. If a position does not match the tuning table, a -1 is stored to indicate that position did not match the input chord.

The "Find all combinations" Module of **Figure 44** uses six nested for loops to process the entire set of combinations for the current 4x6 offset. Each of the six loops corresponds to a string on the guitar. The rows, or strings, are arranged in descending order. A counter stores the number of notes that match the input chord on each row. The row with the largest counter is arranged on top.

The "Find one combination" Module of **Figure 44** cycles through the combinations found one at a time. This single combination is then passed to the "Validate" Module to verify that it is a desired combination.

The "Validate" Module of **Figure 44** produces the validation checks that determine whether one combination is considered to be a playable combination. Its output is a true or false variable that is taken into the "Store combinations" Module of **Figure 45.** The true or false will determine whether the one combination is stored or discarded. The criteria for storing and discarding are explained in **Figure 45.**

From the "Determine store or discard" Module of **Figure 44,** the validation checks for the one combination chord are true or false. True combination values are stored into the combinations vector, which is an exhaustive list of all the playable combinations. False combination values are discarded and simply not placed on the vector. If there is another combination, the program continues to execute for each combination, otherwise it increments the 4x6 offset. The process is then repeated so that all possible combinations for every 4x6 state have been processed and validated.

The "Increment 4x6 offset" Module of **Figure 44** is a logical incremental increase in the 4x6. The next 4x6 portion will contain the same three out of the four columns from the previous 4x6. This is done for every 4x6 subset of the 13x6 tuning table. This is the "sliding window" analogy used to describe the algorithm. Each time the offset is increased, the window has moved by one fret. **Figure 45** shows the Level 3 block diagram for the Chord Voicings Module. This diagram shows the "Validate" Module seen in **Figure 44**. These are the validation checks that determine if a chord is playable.

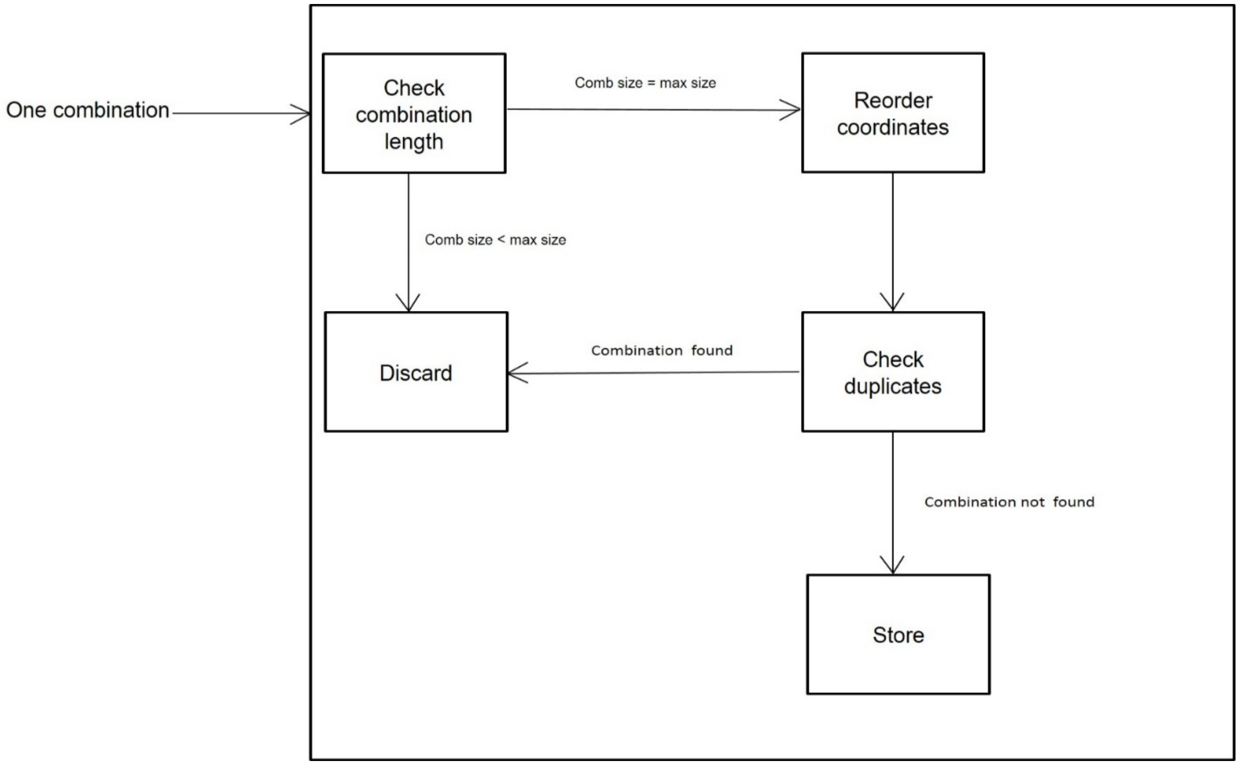


Figure 45. Chord Voicings Module Level 3 validation checks block Diagram.

Table 39. Functional Requirements for the "check combination length" module

Module	Check combination length
Input	One combination
Output	Discard or reorder
Functionality	Determines whether the combination to be validated is the superset size. If the combination contains values that are less than maximum size chord within that 4x6, then immediately they are discarded.

Table 40. Functional Requirements for the "Reorder coordinates" module

Module	Reorder coordinates
Input	One combination
Output	Ordered combination starting at fret 0
Functionality	Place the current combination in order. The nested for loops that are used to find all of the combinations are placed out of order, and also the combinations in the array are placed out of order. The functionality of this module is to reorder them, so that when they are placed in the final vector, they appear in order.

Table 41. Functional Requirements for "Check duplicates" module

Module	Check duplicates
Input	One combination
Output	True or False

Functionality	Determines if the chord being validated has already been found. If the chord has already been found by another 4x6, then the vector should only store one copy, so the duplicate is discarded.

Table 42. Functional Requirements for the “Discard” module

Module	Discard
Input	One combination
Output	Discard
Functionality	Discards a chord based upon the checks from “Check duplicates” and “Check combination length” modules. After the chord is discarded, the 6-nested for loop will continue to find new combinations, or the 4x6 will be incremented.

Table 43. Functional Requirements for the “Store” module

Module	Increment 4x6
Input	One combination
Output	Store
Functionality	Stores a chord that has passed all validation checks. This chord is the maximum size, guaranteeing that is a superset combination. Also the chord has not already been placed upon the playable combinations vector.

The “Check combination length” Module in **Figure 45** is the first of two validating checks. This one determines if the combination length matches the desired length. Each 4x6 offset has a maximum combination size. A counter is used to implement this. If a row contains a note that matches the input chord, then the maximum size is increased. It is important to note that the maximum feasible size in each row contains a matching note, so six is the maximum possible size. For example, if one row did not contain any notes, then the maximum size would be five. If the chord, or one combination that is being validated, matches the maximum size, then it will continue to be ordered and checked for further validation. If the chord size is less than the maximum combination then this is considered a subset of another combination and must be discarded.

The “Reorder coordinates” Module in **Figure 45** rearranges the coordinates into descending order based on row size. This module simply looks at the set and places them so that the first fret is the first coordinate and the last fret is the last coordinate pair. It should be noted that the Y-coordinate will always tell which row or string the coordinate corresponds to, but for ease of use they always follow a specific order. This will also allow the Y-coordinate to be ignored when sending coordinates to the display Module. Only the x coordinate will matter, the Y coordinate will be known based upon the order.

The “Check duplicates” Module in **Figure 45** is the second of two validating checks. Since 4x6 states are processed, duplicate combinations can appear within multiple stages. The “Check duplicates” Module examines the vector, and determines if that combination has already been added. If it does not appear, then the combination corresponding to the chord is stored. If the combination already appears, there is no point in storing the value twice, so the information is discarded.

The “Discard” Module in **Figure 45** discards the chord information if it has failed either of the two validity checks. The program then returns to where it was executing and continues. Usually this will be at some point within the nested six loops which are processing the combinations. If it happens to be the last combination, the 4x6 offset will be increased and another 4x6 set will be processed. If it was the last combination then this becomes the end of the module, as all data has been processed.

If the chord passes both of the validation checks, then it is sent to the “Store” Module in **Figure 45**, which writes the combination onto the playable combinations vector. The program will follow the same flow described in the “Discard” module description shown above. The algorithm will find a new combination to process, end if all of the 4x6 states have been processed, or, provided another state exists, increase the 4x6 offset and process the new state.

3.5.2.6 Display Algorithm (MB)

The display algorithm is a visual representation of the “Chord Voicings” Module seen in section *3.5.2.5.* From the exhaustive list of the playable combinations found, one is displayed to the user through the GUI. The 13x6 array, which corresponds to the guitar’s fretboard has a visual representation for the corresponding lighting arrangement. The primary goal of the Display Algorithm is to allow a beginning guitar player to view which permutation of the chord they are trying to master. The Level 1 block diagram for the Display Algorithm Module is shown in **Figure 46**.

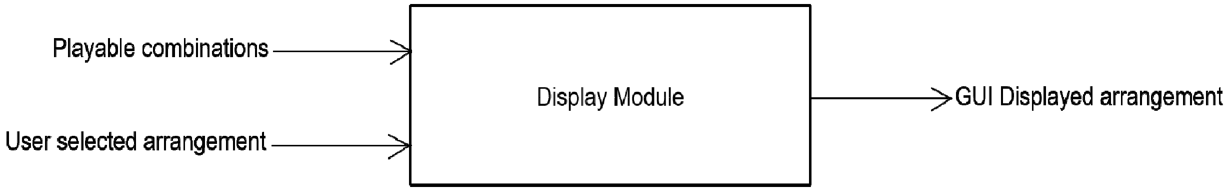


Figure 46. Display Module Level 1 block Diagram.

Table 44. Functional Requirements for the "Display" module

Module	Display
Input	Playable combinations, Selected arrangement
Output	One combination displayed inside of the GUI.
Functionality	Determines the user selected combination, and displays where on the guitar that would be within the GUI. A nice visual representation of where the coordinate locations are.

The purpose of the "Display" Module is to provide the user the ability to visually see one combination of string depressions that make the desired chord and cycle through all of these possible combinations to choose the permutation they would like to use. The playable combinations are found using the "Chord Voicings" Module explained in section *3.5.2.5*.

3.5.2.7 Software LED Control

Once the guitar has been tuned, chords with known positions can be built by the software, as shown in **Figure 47**. The user is then able to select the desired chord from a drop-down menu. The different voicings of the selected chord are displayed in the lighting sequence. The lighting sequence is a six byte serial communication between the software and the microcontroller which contains encoded information about which fret and string combination corresponds to a finger placement. These finger placements are then illuminated on the LED matrix.

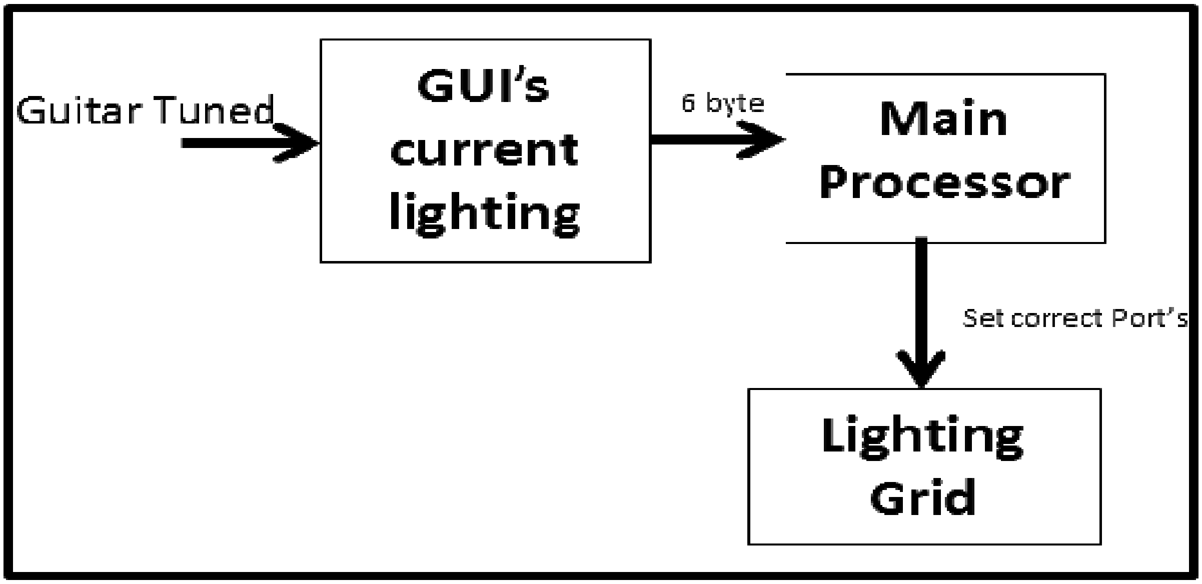


Figure 47. Software LED control.

Figure 48 shows the encoding scheme used to control the LED matrix. The positions of the coordinate to be illuminated are sent as six ASCII bytes in string order. The ASCII bytes represent the fret to be illuminated while the actual order the bytes are received corresponds to the string this fret should be illuminated (e.g. the first byte in the sequence corresponds to the number of the fret on the first string that should be illuminated). This information is sent to the Main Processor module from the GUI software. For example, the lighting sequence shown in **Figure 48** would be represented by a serial communication of "012345".

	Fret1	**Fret 2**	**Fret 3**	**Fret 4**	**Fret 5**	**Fret 6**	**Fret 7**	**Fret 8**	**Fret 9**	**Fret 10**	**Fret 11**	**Fret 12**	**Fret 13**
String 1	0	1	2	3	4	5	6	7	8	9	A	B	C
String 2	0	1	2	3	4	5	6	7	8	9	A	B	C
String 3	0	1	2	3	4	5	6	7	8	9	A	B	C
String 4	0	1	2	3	4	5	6	7	8	9	A	B	C
String 5	0	1	2	3	4	5	6	7	8	9	A	B	C
String 6	0	1	2	3	4	5	6	7	8	9	A	B	C

Figure 48. Encoding for LED matrix for software control.

If nothing is to be displayed on a string, the ASCII character "X" is sent for that string's fret information. **Figure 49**, for example, shows almost every position in Fret 1 illuminated. The lighting sequence that would be sent to the Main Processor module is "000X00". This "X" byte is visible in **Figure 49** as the un-highlighted fret position on String 4.

	Fret1	**Fret 2**	**Fret 3**	**Fret 4**	**Fret 5**	**Fret 6**
String	0	1	2	3	4	5

1	🟩					
String 2	🟩 0	1	2	3	4	5
String 3	🟩 0	1	2	3	4	5
String 4	0	1	2	3	4	5
String 5	🟩 0	1	2	3	4	5
String 6	🟩 0	1	2	3	4	5

Figure 49. An example of Encoding for LED matrix skipping a position.

Once the user presses the button to display the next chord in the GUI, the proper lighting sequence for that particular chord is determined and the six bytes corresponding to this lighting sequence are sent to the Main Processor module.

3.5.2.8 User Display

The User Display consists of a two dimensional array of images which represent the strings and frets on the guitar. The display emulates the LED module on the screen of the PC, showing which LEDs should light up based on the coordinate information passed in from the display algorithm. A screenshot of this display is shown in Appendix C. The User Display also has several status messages. These statuses include a tuning/confirmation status for each of the six strings which shows the expected ideal frequency and the detected frequency for each string, as well as a master status which shows the overall state of the software (including overall tuning/confirmation statuses when appropriate).

Table 45. Functional Requirements for User Display

Module	User Display
Inputs	Initialize Guitar or Confirm Chord: tuning or chord confirmation status Display Algorithm: Coordinates specifying which LEDs to turn on
Outputs	User Interface: Status information and emulation of LED Module
Functionality	The User Display emulates the LED Module on the screen of the PC, using the same coordinate information that is passed to the Main Processor Module to drive the LEDs.

4. Operation Instructions (JF)

4.1 Guitar Setup

In order to get started using the LED Learning Guitar, the user must follow the steps outlined below.

Step 1: Open the RockYa program on the PC.

Step 2: Connect the guitar mounted circuit board (GMCB), located on the back of the guitar, to the USB port on the computer with the provided cable, as shown in **Figure 50**.

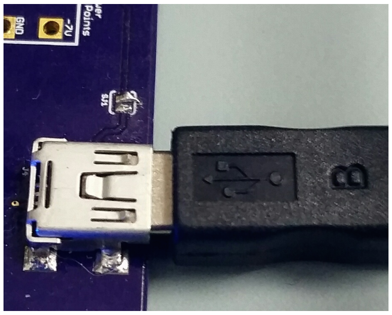


Figure 50. USB connection.

Step 3: Connect the 13-pin GK connector cable must be from the Roland GK-3 hexaphonic pickup (shown in **Figure 51**) to the port on the GMBC (shown in **Figure 52**).

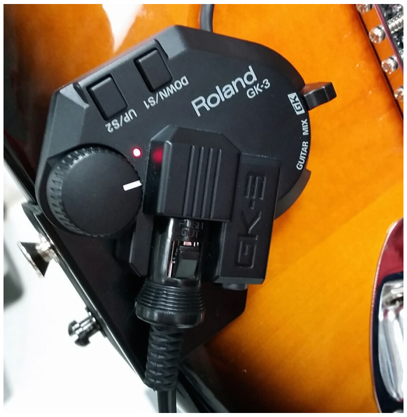


Figure 51. GK cable connection at the Roland GK-3 pickup.

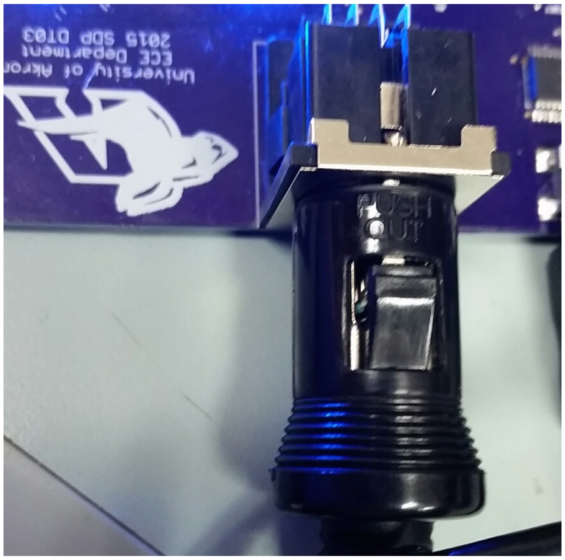


Figure 52. GK cable connection at the GMCB.

Step 4: Plug the power cable into the barrel jack, which is located on the GMCB (see **Figure 53** below). Blue LEDs on the GMBC should illuminate to indicate power. If there is no illumination, check the barrel jack connection and examine the board for any potential shorts. The user should also see the power indicator light on the Roland GK-3 illuminate. If the indicator light is out, reexamine the GK cable connection.

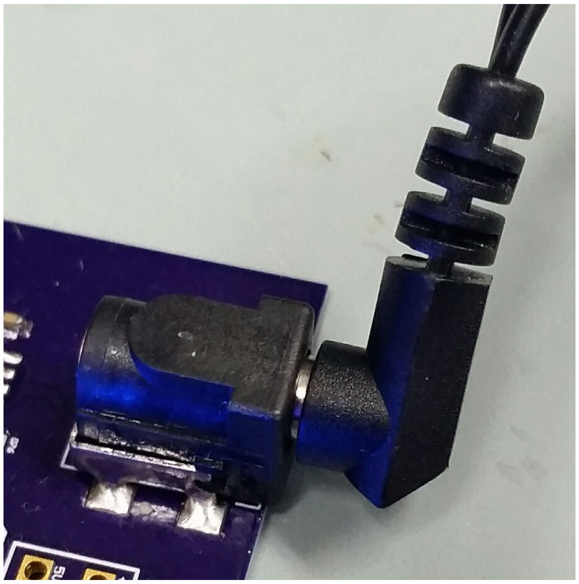
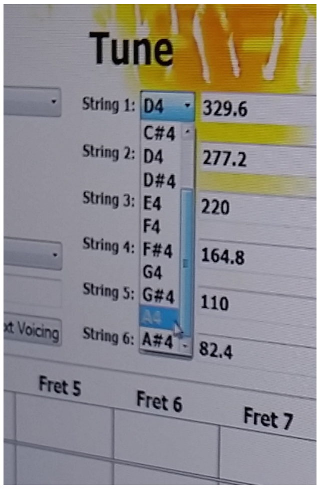


Figure 53. Barrel jack power connection.

4.2 Initialization and Tuning of Guitar

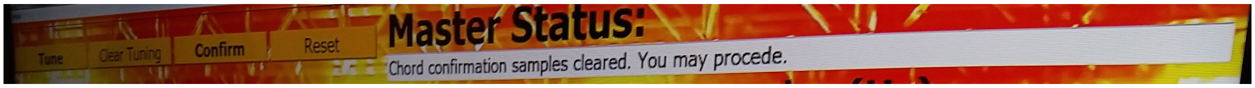
Step 5: Use the graphical user interface (GUI) to select the desired tuning from the drop-down menus on the screen, as shown in **Figure 54** below.



String 1: D4 329.6
String 2: 277.2
String 3: 220
String 4: 164.8
String 5: 110
String 6: 82.4

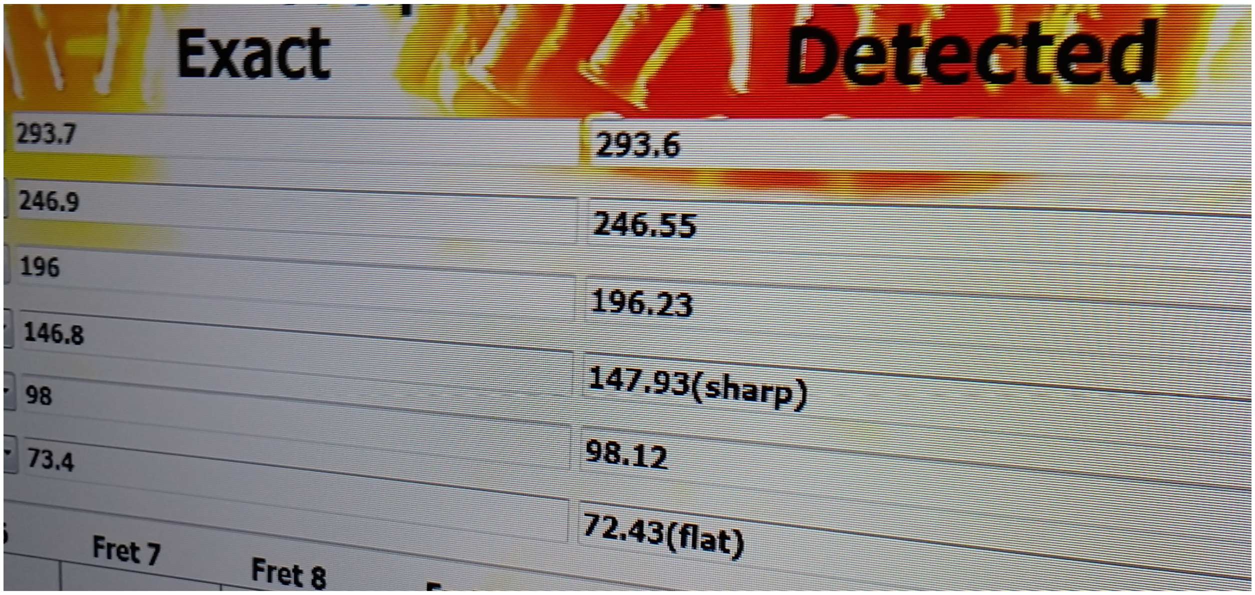
Figure 54. Tuning selection from drop-down menus.

Step 6: Tune the guitar. Start the tuning process by strumming the strings and then clicking the "Tune" button on the GUI. Strumming the guitar and clicking the "Tune Guitar" button several times between strums increases the accuracy of the frequencies read. The software will display when a note is flat or sharp on the right portion of the screen, as shown in **Figure 56**.



Master Status:
Chord confirmation samples cleared. You may procede.

Figure 55. User selection buttons and Master Status indicator.



Exact	Detected
293.7	293.6
246.9	246.55
196	196.23
146.8	147.93(sharp)
98	98.12
73.4	72.43(flat)

Figure 56. Guitar frequency display for tuning purposes.

4.3 Chord Selection

Step 7: Select a chord using the two left drop-down menus on the GUI. The first menu is for the selection of the root note of the chord, as shown in **Figure 57**. The second menu is for the type of chord, as shown in **Figure 58**.

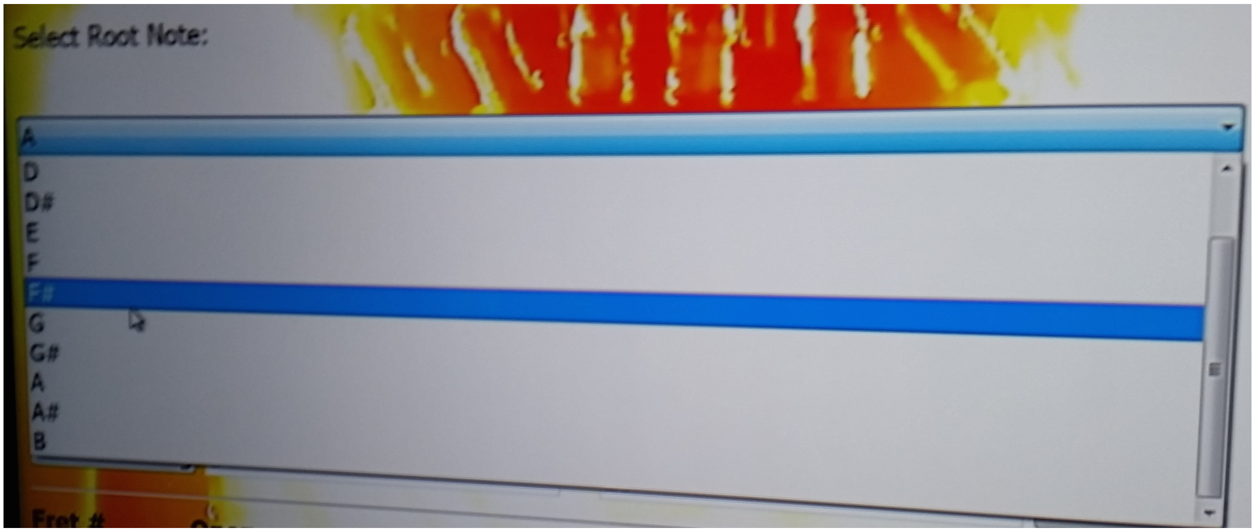


Figure 57. Root note of guitar chord selection drop-down menu.

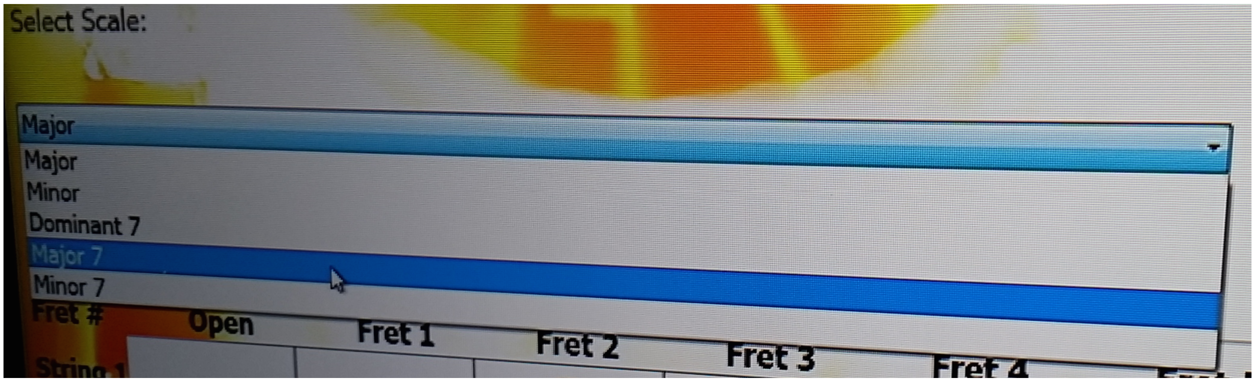
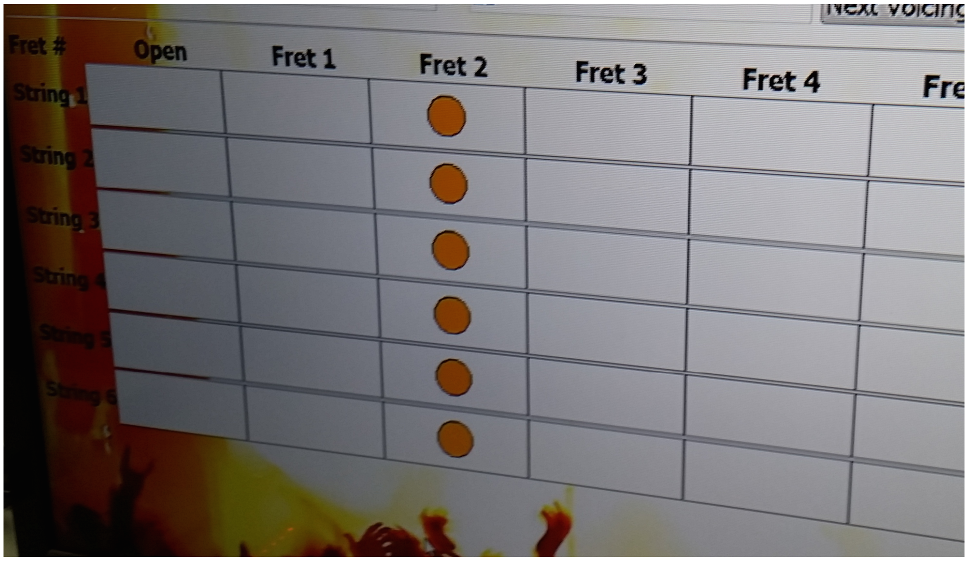


Figure 58. Chord type selection drop-down menu.

Step 8: Select a voicing. At the bottom of the screen, there is a virtual display that shows a voicing of the guitar chord like that shown in **Figure 59**. In addition, the LED Lighting Module on the guitar neck will also display the chord voicing, as shown in **Figure 60**. If the LED Lighting Module is not illuminated when the chord is displayed on screen, make sure that the connection points of the ribbon cable at the GMBC and the Lighting Module are tight and secured. The user can click "Previous Voicing" and "Next Voicing" buttons to change the voicing of the guitar chord. **Figure 61** shows a different voicing for the same chord as the one shown in **Figure 59**.



Fret #	Open	Fret 1	Fret 2	Fret 3	Fret 4	Fre
String 1			●			
String 2			●			
String 3			●			
String 4			●			
String 5			●			
String 6			●			

Figure 59. Virtual guitar neck display showing an A major chord in open G tuning.

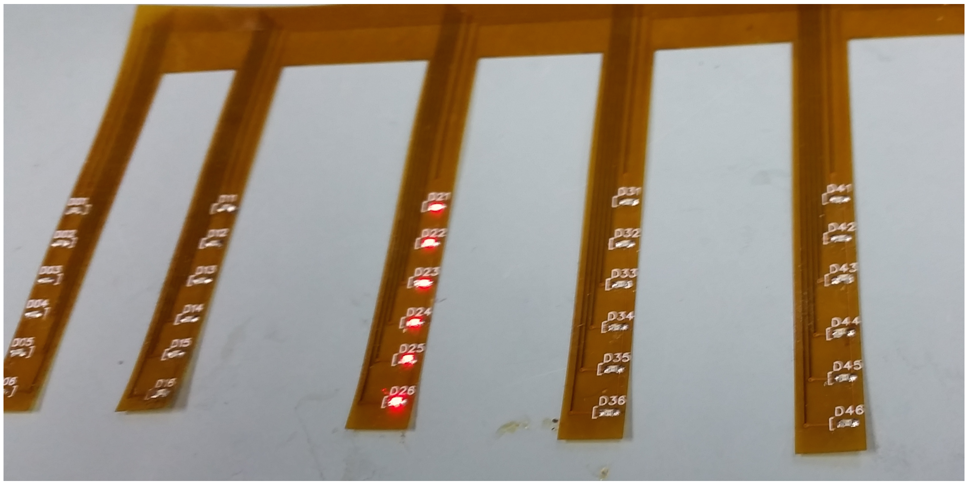


Figure 60. LED Lighting Module guitar chord display.

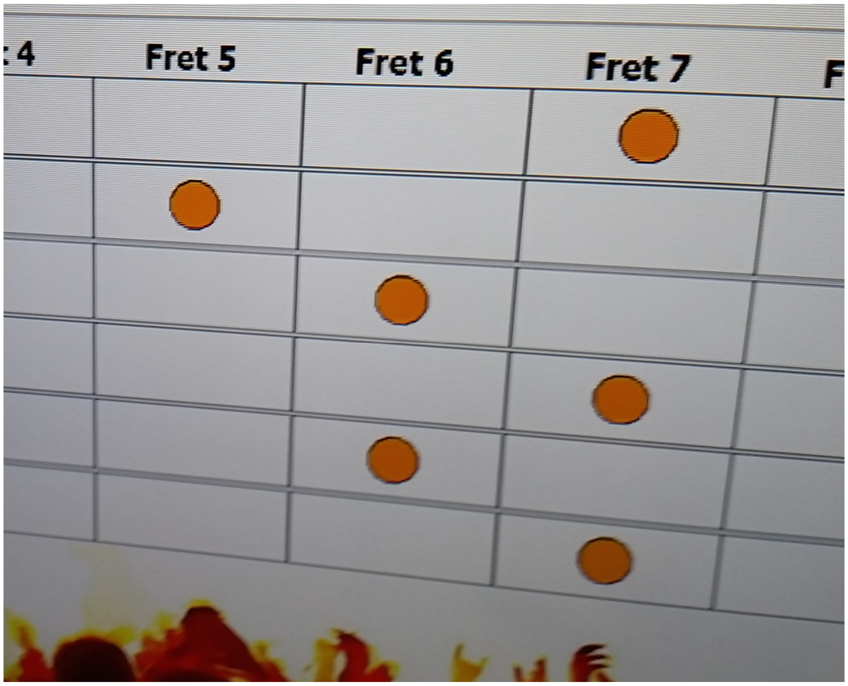
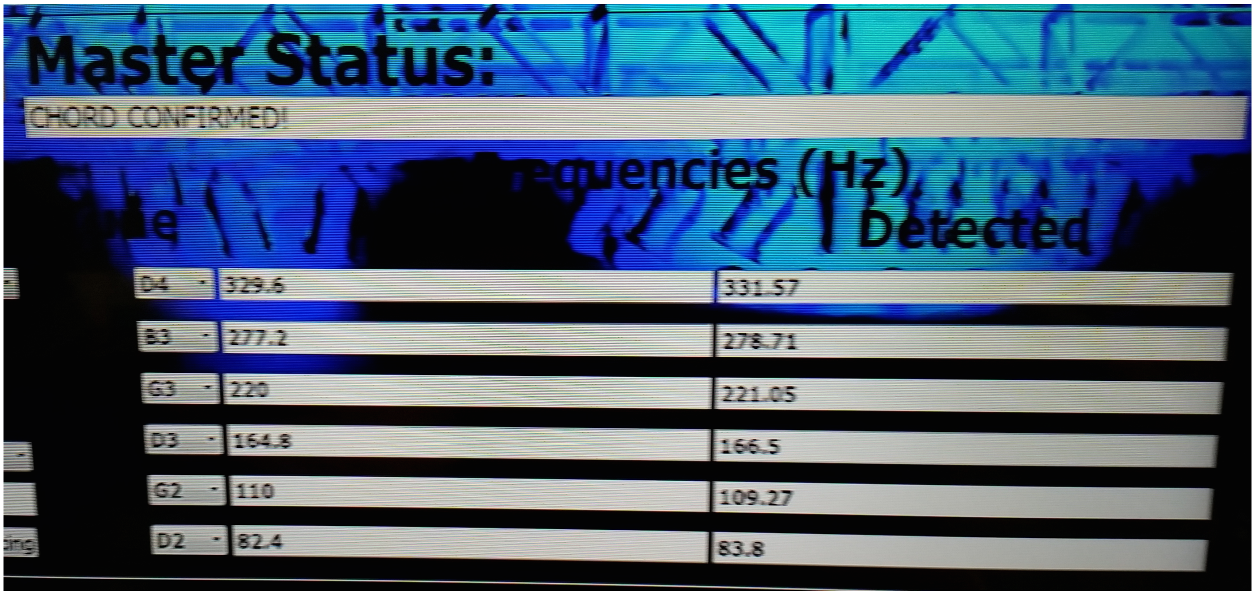


Figure 61. Virtual guitar neck display showing a new voicing for an A major chord in open G tuning.

4.4 Playing Chords and Feedback

Step 9: Play the chord. Once the user has settled on the desired chord, the fingers are to be placed on the notes where indicated by the virtual guitar display and the LED Lighting Module. In some cases, the user might find that a voicing is uncomfortable or nearly impossible. For these cases, the use of a capo on the lowest illuminated fret is encouraged. After the user strums the guitar and clicks the “Confirm” button, the GUI will provide feedback to inform the user if the chord was played correctly or if the notes played were out of tune. A guitar chord that is played correctly either partially (meaning the minimum notes required to make the desired chord are detected) or fully (all the right notes displayed on the guitar neck were detected) results in a color change of the GUI backdrop and a “Chord partially correct” or “CHORD CONFIRMED” message in the Master Status box at the top of the screen. **Figure 62** shows a “CHORD CONFIRMED” status, for example. If the user wishes to try a different chord, a simple click of the “Reset” button and a toggle of chord selection through the drop down menus will accomplish this.



Master Status:

CHORD CONFIRMED!

	Frequencies (Hz)	Detected
D4	329.6	331.57
B3	277.2	278.71
G3	220	221.05
D3	164.8	166.5
G2	110	109.27
D2	82.4	83.8

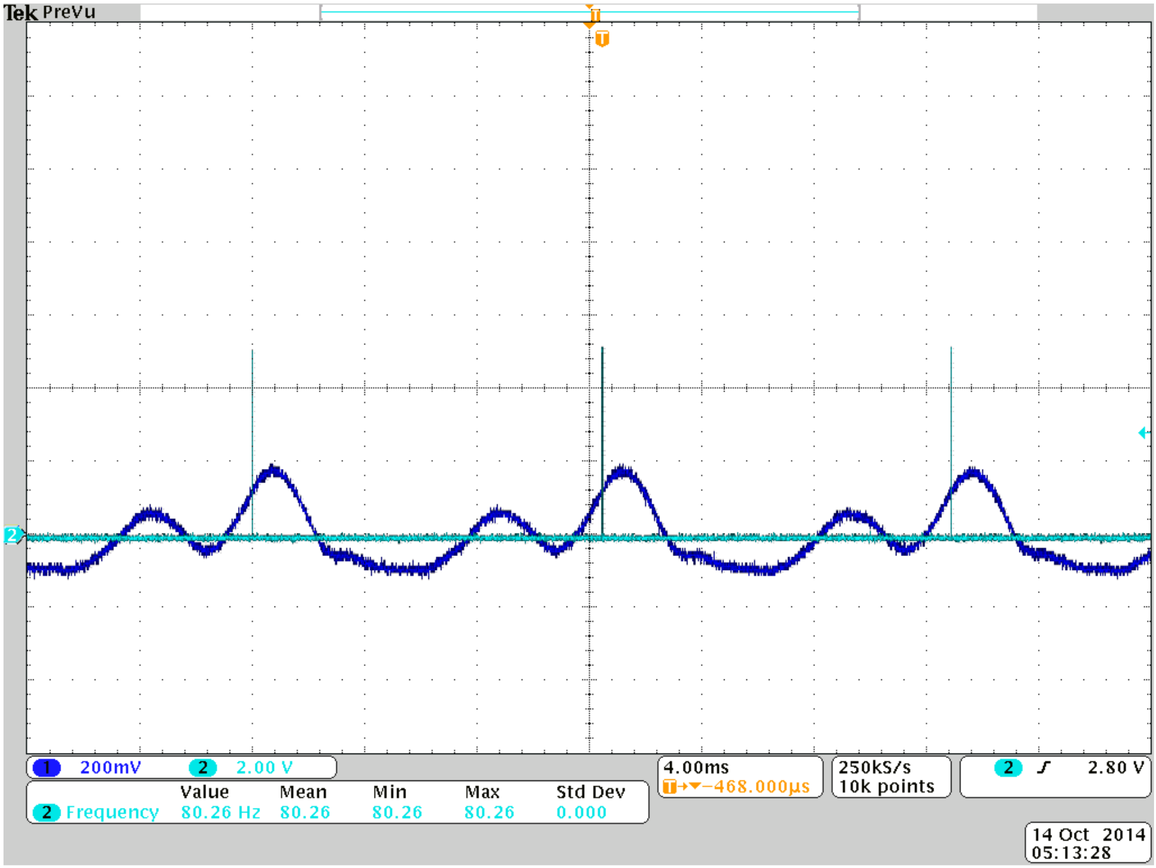
Figure 62. Guitar chord feedback display for correctly played chord.

5. Testing Procedures (JF)

Testing procedures for the various hardware and software modules that make up the system are given below.

5.1 Pickup Module

The pickup module was tested to measure the strength and frequency of the signals created by the excitation of the guitar strings. A test cable was constructed using a 13-pin male connector onto which wires were soldered to each pin. These wires were connected to a breadboard. Two power supplies provided the necessary +/- 7 volts to the power pins. The six analog audio outputs for the guitar strings were measured using the oscilloscope, as shown in **Figure 63**. The amplitude of the signal, as expected, was dependent upon the intensity of the strum. The peak amplitudes of these waveforms were taken into consideration when choosing the thresholds for the pitch detection module.



	Value	Mean	Min	Max	Std Dev
2 Frequency	80.26 Hz	80.26	80.26	80.26	0.000

Figure 63. Measured analog string signal.

5.2 Pitch Detection Module

Initial testing for the pitch detection module started with the Atmega 328P Microcontroller Circuit Board Module. The 328P was placed on a breadboard. A single analog audio output of the hexaphonic pickup served as the signal to be analyzed, as can be seen in **Figure 64**. Thresholds for measuring the frequency of the audio signal were set according to the measurements taken with the oscilloscope. The Atmega, which was connected via USB, sent the determined frequencies to a serial monitor on the PC.

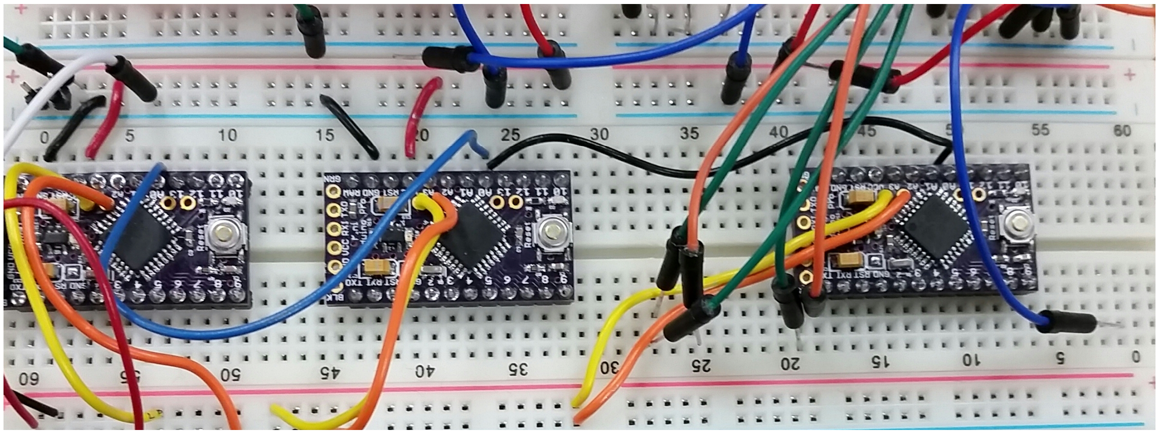


Figure 64. Pitch detection breadboard testing.

Once the Guitar-Mounted Circuit Board (GMCB) was completed, each of the six pitch detector chips on the board was programmed to detect frequencies on a given string. Each string was found to have slightly different voltage thresholds, given that the thicker strings were capable of producing higher amplitudes. The GK connector cable was run from the output of the hexaphonic pickup into the 13-pin port on the GMCB. Communicating once again via USB, the measured frequencies were displayed on the serial monitor for the six strings. The measured frequencies were compared with the ideal values for a particular note played to determine the accuracy of the pitch detection.

5.3 Software/LED Control Module Interfacing

The chord builder software was programmed to display chord voicings on the virtual guitar display as well as on the LED Lighting Module. Since the LED Control Module manages the lighting display for the guitar neck, testing was performed to check the interfacing of the software with DIP decoders on the breadboard. The outputs of the decoders were connected to a 4-by-6 array of breadboard-mounted LEDs, as shown in **Figure 65**. The software was connected via USB to the Atmega 2560 test board, out of which jumper wires connected to the decoders on the breadboard. Lighting test sequences and guitar chords were passed to the breadboard, decoded, and displayed. These tests would later evolve into full scale testing with the LED Lighting Module once the FCB was produced.

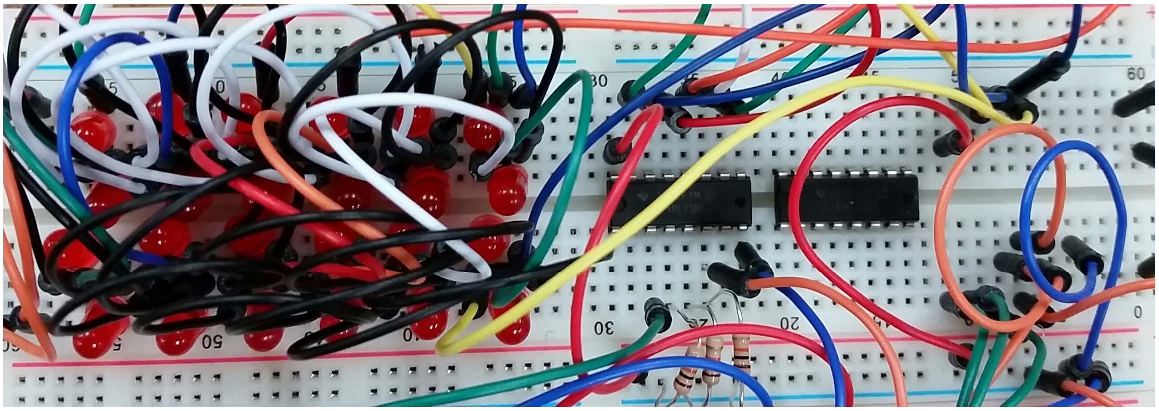


Figure 65. Breadboarded decoders and LED array.

5.4 Software/LED Module Interfacing

After the LED Lighting Module was received and populated, testing with the whole LED Module could take place. The PC was connected to the GMCB-housed Atmega 2560 and decoders of the Lighting Control Module. The GMCB sent its lighting signals through the ribbon cable to the LED Lighting Module, as seen in **Figure 66**. Sending test sequences to the decoders verified the condition and orientation of the LEDs. These sequences were used to make sure that all of the LEDs were functional and to ensure good connections through the ribbon cable.

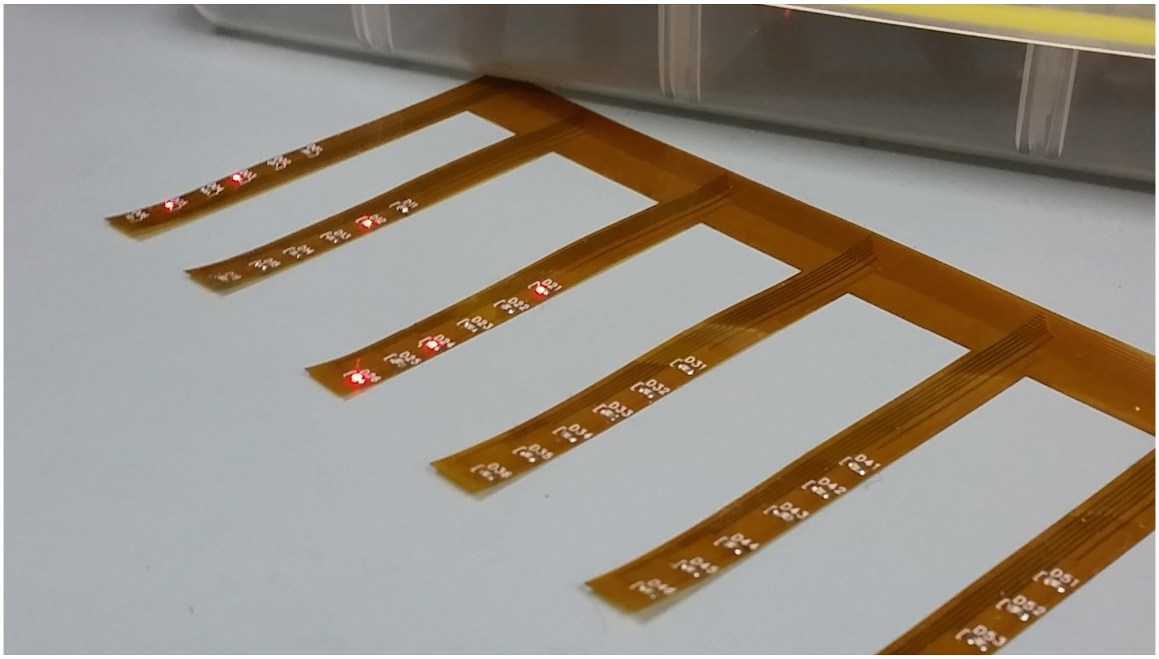
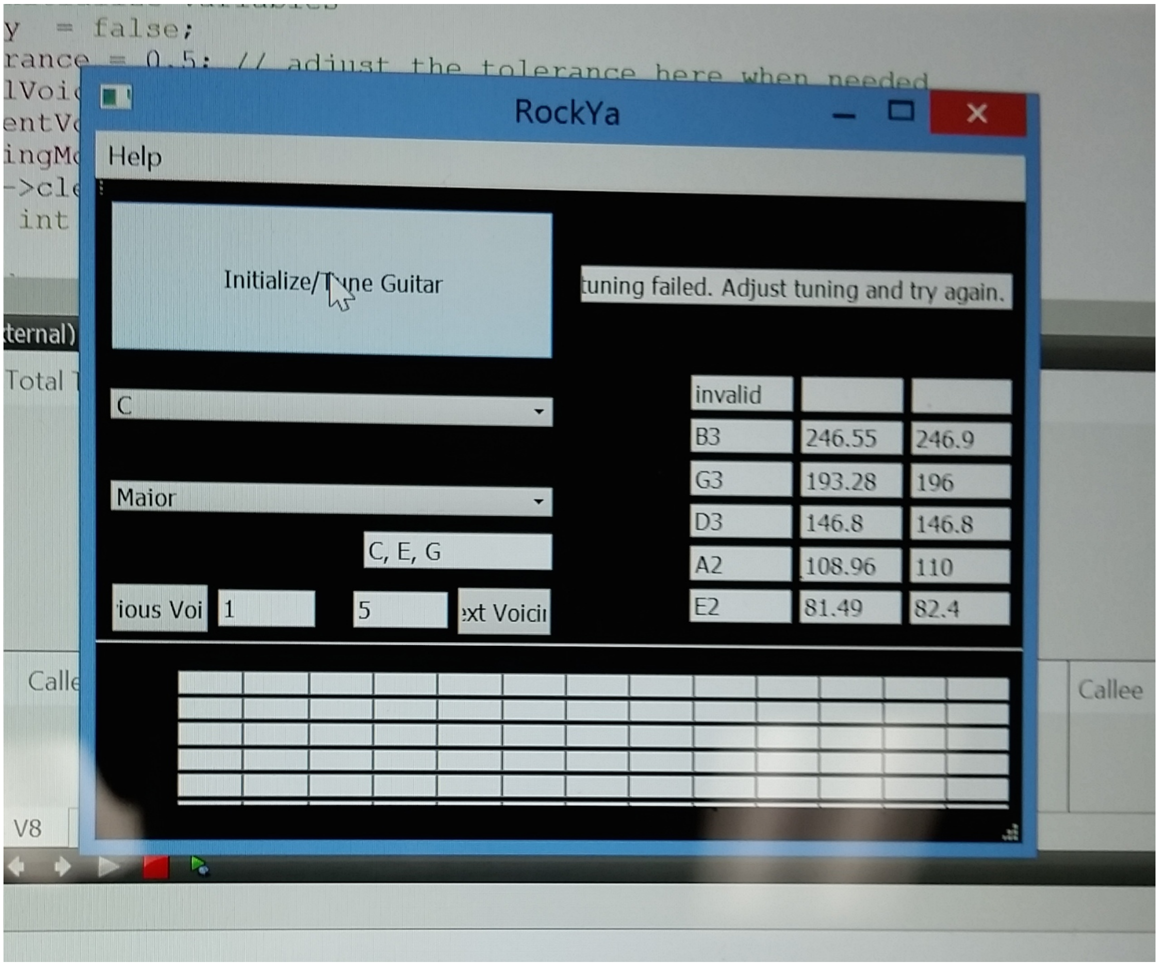


Figure 66. LED Lighting Module test output.

5.5 Software/Pitch Detection Interfacing

The goal of these tests was to establish the connection from the GK-3 pickup to the software. As a follow-up to the testing in 5.2, the values of frequency that were detected and displayed in the serial monitor were passed to the software via I^{2}C. Successful communication between the pitch detectors and the software would result in the display of the frequencies of the notes played on the right half of the GUI, as shown in **Figure 67.** These tests also provided an opportunity to remedy the pitch detection issue, wherein only a portion of the frequencies could be read (can be seen in the box that reads "invalid" in **Figure 67** below) or sometimes an incorrect frequency would be displayed.



tuning failed. Adjust tuning and try again.

C

Major

C, E, G

1

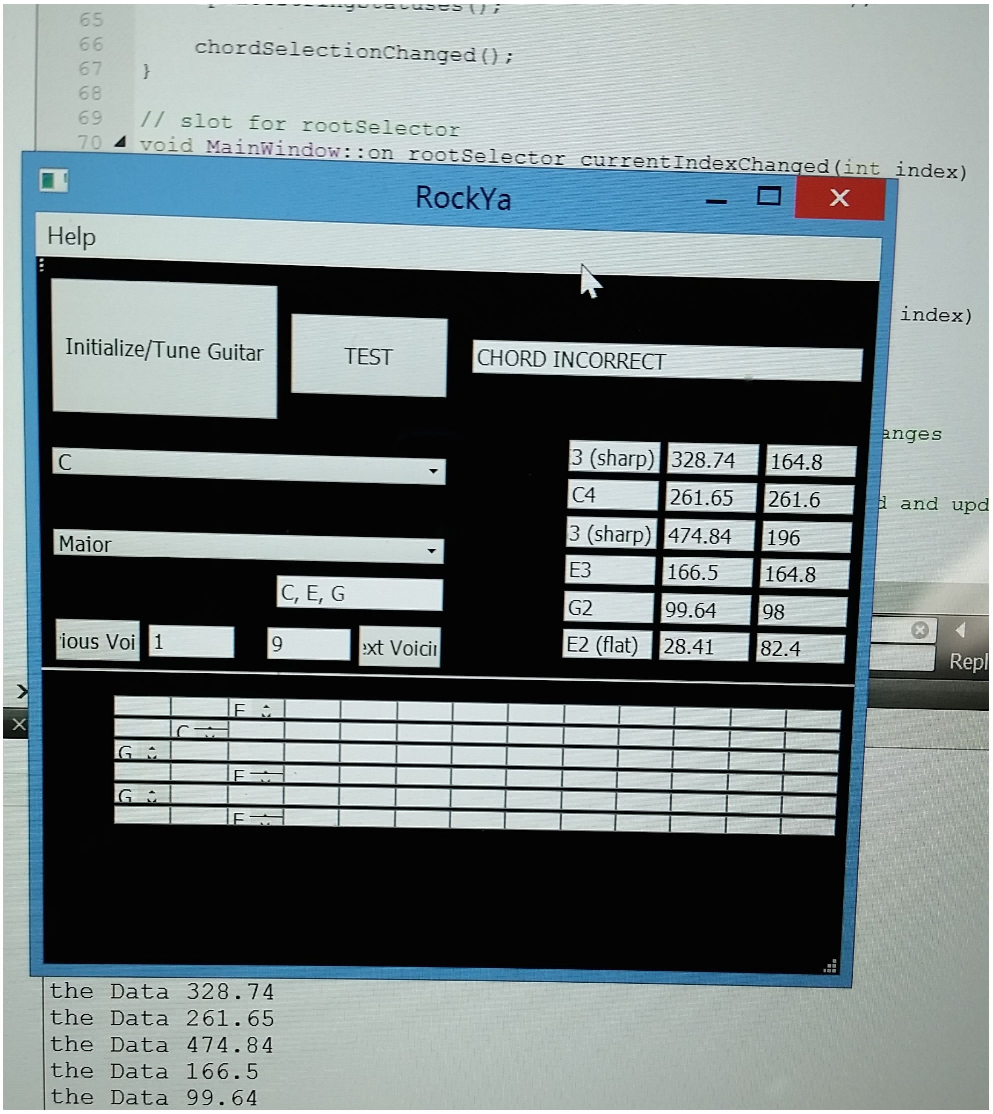
5

invalid		
B3	246.55	246.9
G3	193.28	196
D3	146.8	146.8
A2	108.96	110
E2	81.49	82.4

Figure 67. Guitar tuning initialization testing.

5.6 Software Feedback Testing

The software was designed to be able to interpret whether or not a note was played correctly. It was also designed to detect if a note was out of tune. To test the feedback capabilities of the guitar, notes that make a desired chord of an in-tune guitar were played. Successful interpretation by the software would indicate that the notes were played correctly, as was indeed the case. Second, a chord was played incorrectly to see if the software would be able to tell which note in the chord was wrong. Further, the guitar would be initialized in the correct tuning, but later intentionally reconfigured to be slightly out of tune while playing a chord. A successful test would result in the software seeing which guitar notes were out of tune while playing the note. In **Figure 68** below, it can be seen that an incorrect chord was played due to incorrect frequencies detected (the values in the middle column) when compared to the ideal frequencies (the values in the right column).



```
    chordSelectionChanged();
}

// slot for rootSelector
void MainWindow::on_rootSelector_currentIndexChanged(int index)
```

CHORD INCORRECT

C

Maior

C, E, G

1

9

3 (sharp)	328.74	164.8
C4	261.65	261.6
3 (sharp)	474.84	196
E3	166.5	164.8
G2	99.64	98
E2 (flat)	28.41	82.4

```
the Data 328.74
the Data 261.65
the Data 474.84
the Data 166.5
the Data 99.64
```

Figure 68. Software feedback testing example.

5.7 Full Integration Testing

Once the previous 6 tests have produced successful results, all components of the design could be brought together. These tests involve using the guitar as it was intended, as outlined in the Operating Instructions in Section 4. The project meets all of the engineering and marketing requirements discussed in detail in Section 9.1, as well as full functionality.

6. Budget Information (KR/JF)

Goal Budget

Squier Stratocaster (Guitar Center)	$200
Roland GK-3 Pickup (Amazon)	$170
Parts Request Order #1	$99.77
Flexible Circuit Board	$240.00
Main Circuit Board	$100.00
Total	$809.77

Actual Budget

Squier Stratocaster (Guitar Center)	$200
Roland GK-3 Pickup (Amazon)	$170
Roland GK Connector (Amazon)	$30
Parts Request Order #1	$99.77
Parts Request Order #2	$1.58
Flexible Circuit Board (MyroPCB)	$177.00
Guitar-Mounted Circuit Board PCB (Oshpark)	$137.30
Connector Cable PCB (Oshpark)	$2.30
Total	$817.95

The estimated budget in the fall turned out to be close to the actual expenses in the spring. The total expenses in the project amounted to be $8.18 over budget. Most notably, the largest difference in cost was that of the flex PCB, whose initial cost was estimated to be $240. Once a new quote from a different manufacturer was obtained, the cost for 3 flex PCBs was reduced to $177. An addition to the actual budget was the Roland GK cable.

7. Project Schedule (KR/JF)

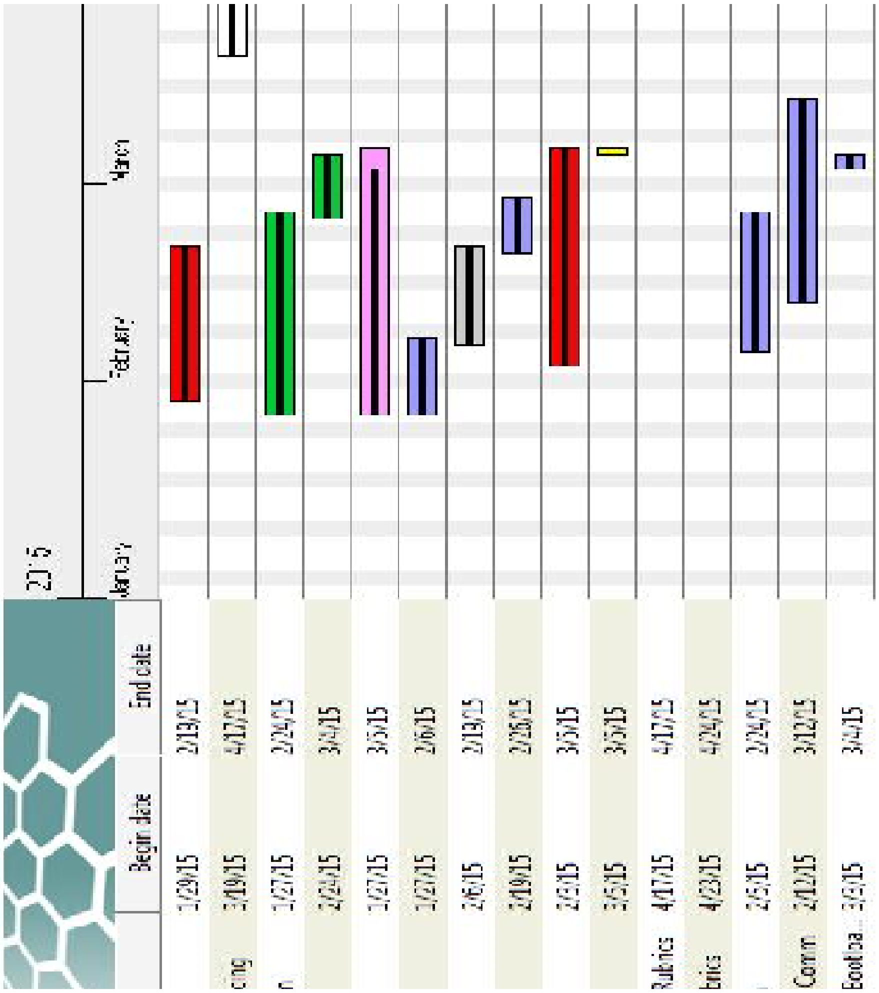
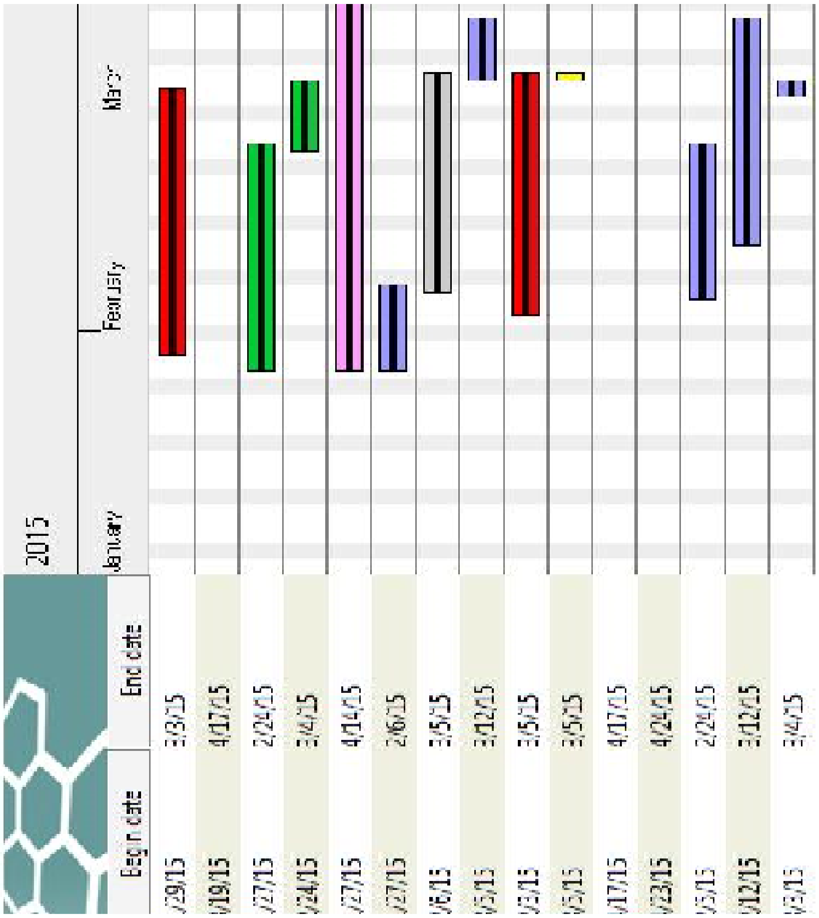


Figure 69. Idealistic Gantt chart.



Begin date	End date
/29/15	3/3/15
/19/15	4/17/15
/27/15	2/24/15
/24/15	3/4/15
/27/15	4/14/15
/27/15	2/6/15
/6/15	3/5/15
/5/15	3/12/15
/3/15	3/5/15
/5/15	3/5/15
/17/15	4/17/15
/23/15	4/24/15
/5/15	2/24/15
/12/15	3/12/15
/3/15	3/4/15

Figure 70. Actual Gantt chart.

From a broad perspective, all tasks were finished before the hard deadline on April 17. The differences between the estimated project timeline and the actual timeline include the longer development time of the graphical user interface, the lead time of the flexible printed circuit board (due to the Chinese New Year), and the lead time for the Guitar-Mounted Circuit Board. All other tasks were completed as scheduled.

8. Design Team Information

Jacob Barb, Computer Engineering, Hardware Manager
Mike Bolin, Computer Engineering, Software Manager
Justin Fiser, Electrical Engineering, Project Manager
Kellen Reusser, Electrical Engineering, Archivist
Dr. De Abreu-Garcia, Faculty Advisor

9. Conclusions and Recommendations

9.1 Satisfying the Design Requirements

Table 46 below shows how the design meets the engineering requirements listed in the Design Requirements Specification.

Table 46. Electronic Learning Guitar Engineering Requirements

Engineering Requirements	Implementation that satisfies requirement
7. *The pitch detection should be able to detect frequencies ranging over B1 (61.7 Hz) to G5 (784.0 Hz).*	The Atmega328P is more than capable of detecting this range of frequencies.
8. *The electronics will allow for 10 cents sharp or 10 cents flat as acceptable error range for notes considered in tune.*	For a particular note, the software has a band of allowable frequencies associated with 10 cents sharp/flat.
9. *The microcontroller should meet the size requirements for I/O.*	With 86 pins that can be used for I/O, the Atmega2560 was enough to accommodate the 16 pins required by the design.
10. *The process of pitch detection and feedback should execute in less than 2 seconds.*	Processing from the strum of the guitar to display of feedback can be executed in under 300ms.
11. *The system should be able to operate from a source of 9V.*	All components in the design operate at ±7 V or +5 V.
12. *Chords will be displayed over frets 0 through 12 (first octave).*	The LED Lighting Module was designed to display notes over frets 0 through 12.

Table 47 below shows how the design meets the marketing requirements listed in the Design Requirements Specification.

Table 47. Electronic Learning Guitar Marketing Requirements

Engineering Requirements	Implementation that satisfies requirement
1. *The guitar should be a real, playable electric guitar.*	A commercially available electric guitar was purchased and modified accordingly.
2. *The guitar should display each note via lit LEDs.*	The LED Lighting Module contains an array of LEDs that fits under the guitar strings.
3. *The guitar should be able to process multiple notes simultaneously.*	The Hexaphonic Pickup Module and Pitch Detection Module process 6 notes at once.
4. *The guitar should be able to display and detect chords.*	The overall system design, as described in the Technical Design section, accomplishes both of these tasks.
5. *The guitar should work regardless of tuning and will inform user if guitar is out of tune.*	Software-based tuning calculation and note verification determines how the guitar is tuned and if a note is not at the right pitch.
6. *The guitar should have an easy-to-read user interface to interact with the player.*	The PC includes a GUI to accept inputs from and provide feedback to the user.

9.2 Recommendations Regarding Customization

Many aspects of the design can be reproduced to work on a different guitar. For instance, the Roland GK-3 hexaphonic pickup is designed to be universal for electric guitars, provided that the strings are wound with a metal to excite the windings within the pickup. For guitars with a curved body, adapter kits are available online to assist in mounting the pickup. The guitar-mounted circuit board (GMCB) can also be placed on the guitar in a location comfortable to the user. However, the cable that connects the GMCB to the LED Lighting Module must be resized to reach both circuit boards. The flexible circuit board for the LED Lighting Module was designed to fit the guitar used in this project. Because fret sizes and string spacing vary among guitars, the LED Lighting Module would not need to be modified accordingly.

10. References

[1] Harrison, E. (2010). Challenges facing guitar education. *Music Educators Journal, 97*(1), 50-55.

[2] Müller, M., Ellis, D. P. W., Klapuri, A., & Richard, G. (2011). Signal processing for music analysis. *IEEE Journal of Selected Topics in Signal Processing, 5*(6), 1088.

[3] Liarokapis, F. (2005). Augmented reality scenarios for guitar learning. *EG UK Theory and Practice of Computer Graphics*

[4] Breitweiser, F. W., Jr., & Weeks, P. F., Jr. (1995). Brietweiser Music Technology Inc., *Musical instrument training system having Displays To identify fingering, playing and Instructional Information* (84/477 R; 84/464 A; 84/455 ed.). New Jersey, United States: G10G 1/02.

[5] Bartos, J. (2011). Heslin Rothenberg Farley & Mesiti, *Self-teaching and entertainment guitar systems* (84/485 R). New Jersey: G09B 15/06.

[6] Luther, M. (2006). Jacobson Holman PLLC, *Training system for musical instruments* (84/602 ed.). Washington, D.C.: G10H 7/00.

[7] Hardy, D. (Aug 12, 2012). Starplayit Pty Ltd, *Music display system* (84/485 R). Australia: G09B 15/00.

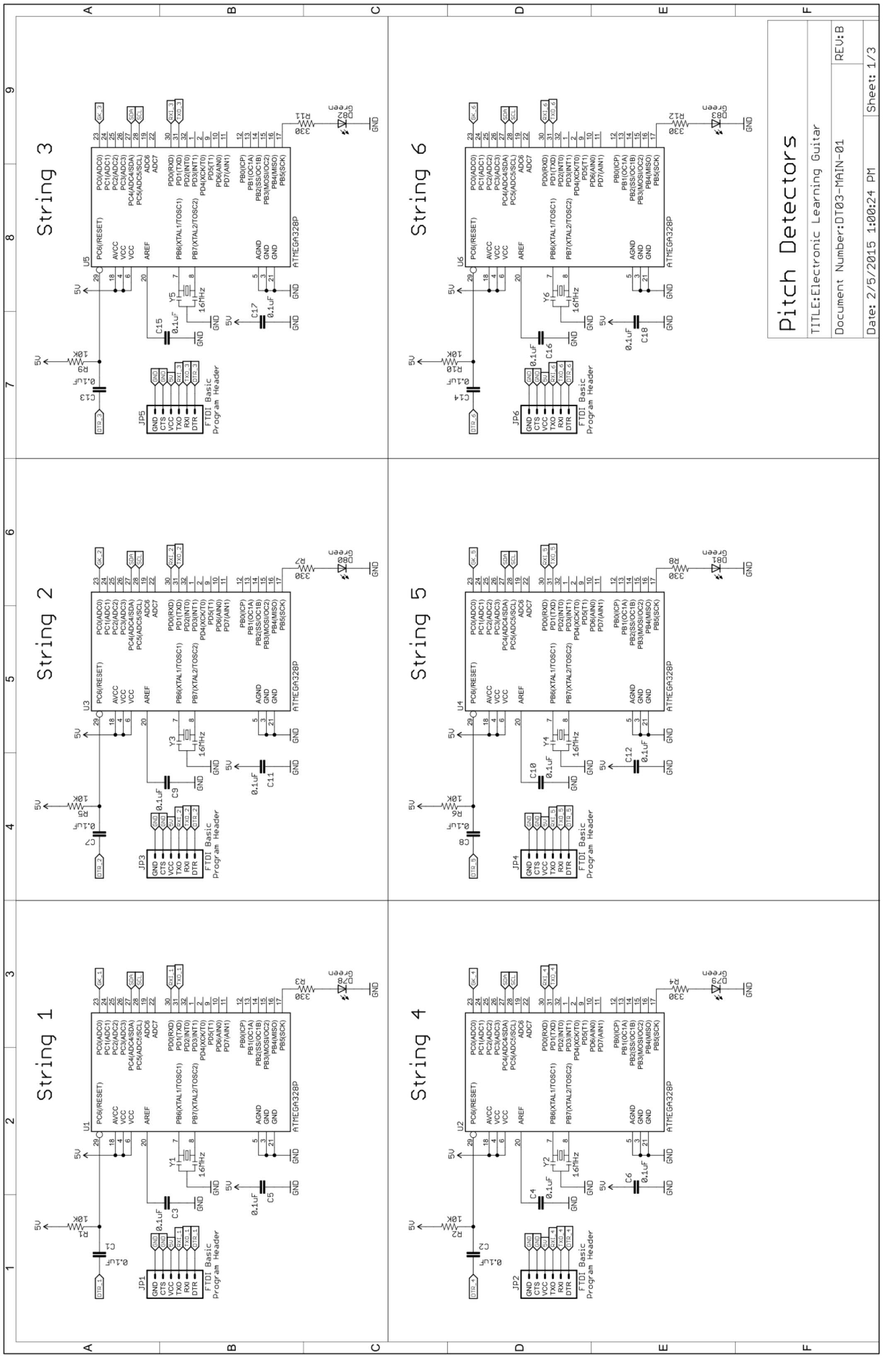
[8] Kyozo, Ota. (1978). Keio Giken Kogyo Kabushiki Kaisha, (84/454; 84/1.01). Tokyo, Japan: G10G 7/02.

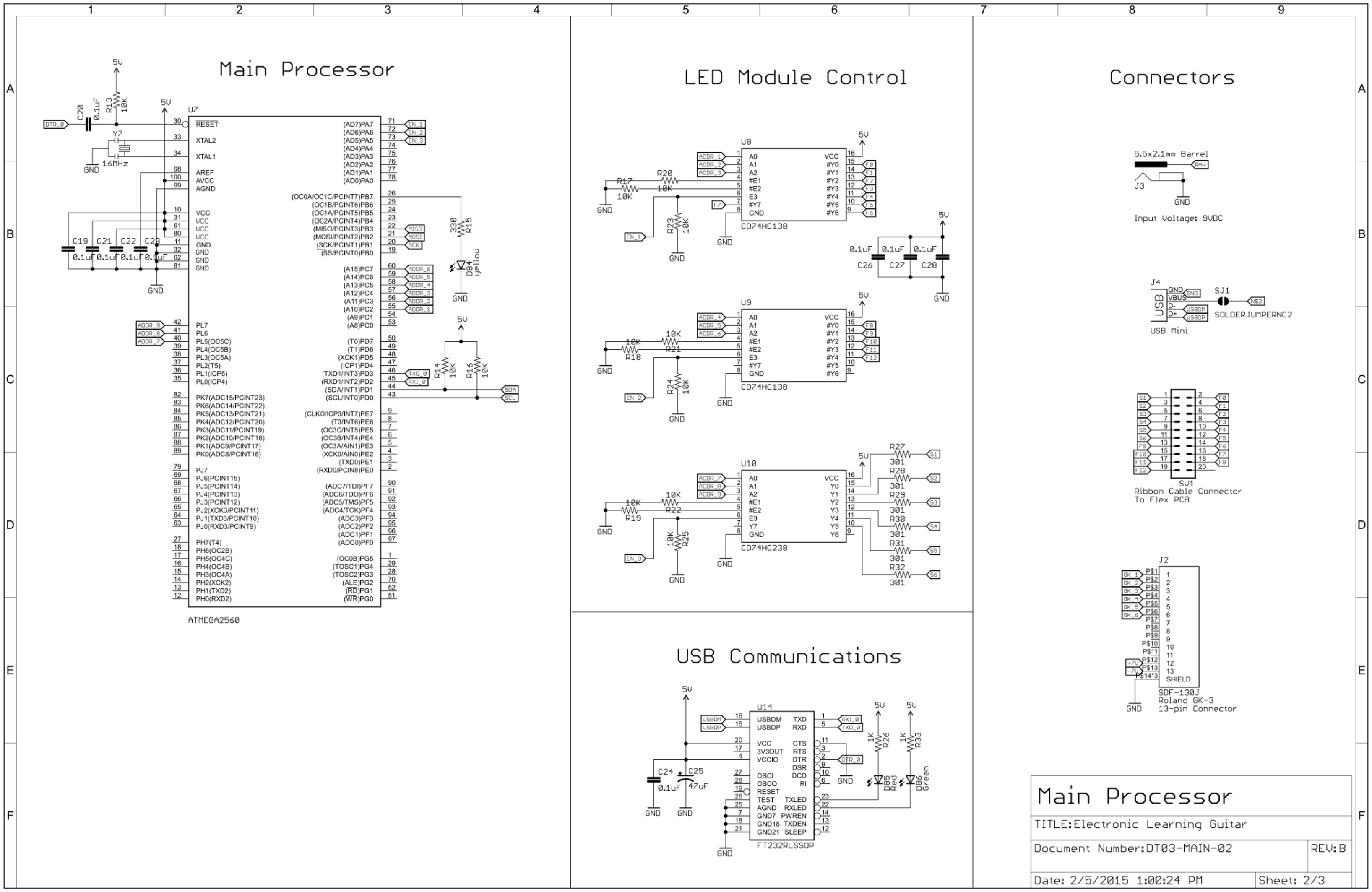
Appendix A: Datasheets

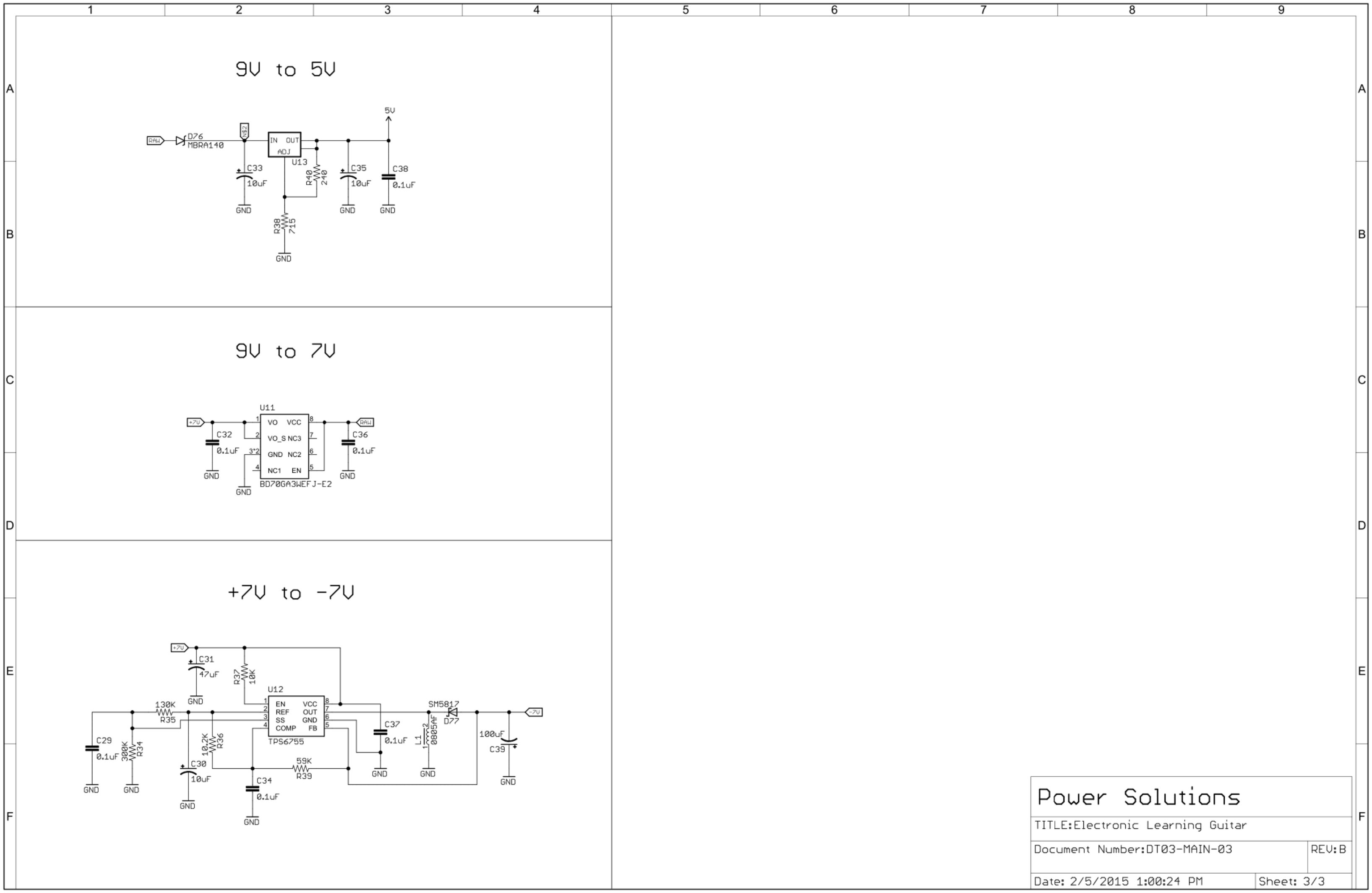
Reference Designator	Manufacturer Part Number	Datasheet Link
J0, J1	DF40C(2.0)-20DS-0.4V(51)	http://www.hirose.co.jp/cataloge_hp/ed_DF40_20140305.pdf
J2	SDF-130J	http://www.cui.com/product/resource/sdf-xxj.pdf
Y1, Y2, Y3, Y4, Y5, Y6, Y7	CSTCE16M0V53-R0	http://www.murata.com/~/media/webrenewal/support/library/catalog/products/timingdevice/ceralock/p16e.ashx
U1, U2, U3, U4, U5, U6	ATMEGA328P-AU	http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf
U7	ATMEGA2560-16AU	http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf
U14	FT232RL-REEL	http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf
U11	BD70GA3WEFJ-E2	http://rohmfs.rohm.com/en/products/databook/datasheet/ic/power/linear_regulator/bdxxga3wefj-e.pdf
U13	LM1117-N-5.0	http://www.ti.com/lit/ds/symlink/lm1117-n.pdf
U8, U9, U10	CD74HC138/238	http://www.ti.com/lit/ds/symlink/cd74hc238.pdf
U12	TPS6755	http://www.ti.com.cn/cn/lit/ds/symlink/tps6755.pdf
D0 – D75	LNJ237W82RA	http://www.semicon.panasonic.co.jp/ds4/LNJ237W82RA_E.pdf

Appendix B:
Guitar Mounted Circuit Board Schematic

(See following pages)

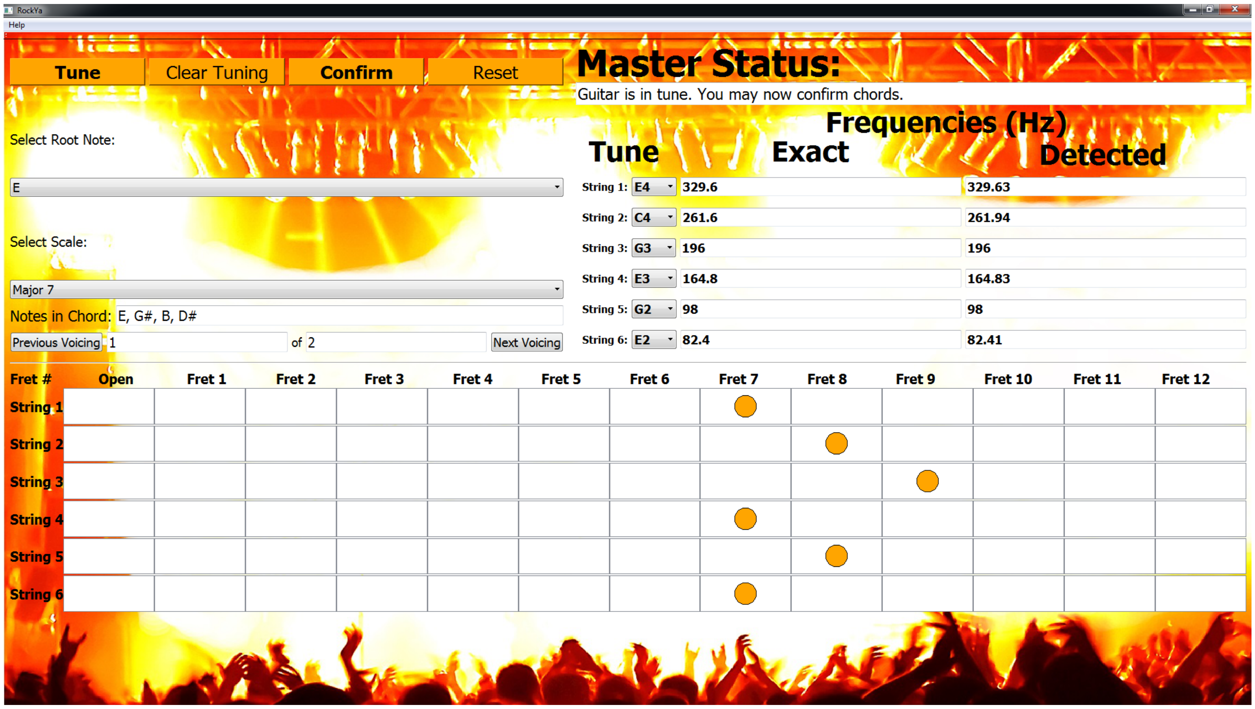

Pitch Detectors
TITLE:Electronic Learning Guitar
Document Number:DT03-MAIN-01
REV:B
Date: 2/5/2015 1:00:24 PM
Sheet: 1/3
String 1
String 2
String 3
String 4
String 5
String 6
ATMEGA328P
FTDI Basic Program Header
16MHz
0.1uF
10K
330
Green
5V
GND


Main Processor
LED Module Control
USB Communications
Connectors
U7
ATMEGA2560
RESET
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
Y7
16MHz
C20 0.1uF
R13 10K
C19 C21 C22 C23 0.1uF
DTR_0
EN_1
EN_2
EN_3
MISO
MOSI
SCK
ADDR_1
ADDR_2
ADDR_3
ADDR_4
ADDR_5
ADDR_6
ADDR_7
ADDR_8
ADDR_9
TXD_0
RXI_0
SDA
SCL
R15 330
D84 yellow
R14 10K
R16 10K
5V
U8 CD74HC138
U9 CD74HC138
U10 CD74HC238
R17 10K
R18 10K
R19 10K
R20 10K
R21 10K
R22 10K
R23 10K
R24 10K
R25 10K
R27 R28 R29 R30 R31 R32 301
C26 C27 C28 0.1uF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
S1 S2 S3 S4 S5 S6
U14 FT232RLSSOP
USBDM
USBDP
TXD
RXD
CTS
RTS
DTR
DSR
DCD
RI
3V3OUT
VCCIO
OSCI
OSCO
TEST
TXLED
RXLED
PWREN
TXDEN
SLEEP
C24 0.1uF
C25 47uF
R26 1K
R33 1K
D85 Red
D86 Green
5.5x2.1mm Barrel
J3
RAW
Input Voltage: 9VDC
J4
USB
USB Mini
SJ1
SOLDERJUMPERNC2
SV1
Ribbon Cable Connector
To Flex PCB
J2
SDF-130J
Roland GK-3
13-pin Connector
GK_1 GK_2 GK_3 GK_4 GK_5 GK_6
+7V
-7V
SHIELD
Main Processor
TITLE:Electronic Learning Guitar
Document Number:DT03-MAIN-02
REV:B
Date: 2/5/2015 1:00:24 PM
Sheet: 2/3


1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
9V to 5V
RAW
D76
MBRA140
N$2
IN
OUT
ADJ
U13
C33
10uF
GND
R40
240
C35
10uF
C38
0.1uF
5V
R38
715
9V to 7V
U11
+7V
VO
VCC
VO_S
NC3
GND
NC2
NC1
EN
1
2
3*2
4
8
7
6
5
BD70GA3WEFJ-E2
C32
0.1uF
C36
0.1uF
RAW
+7V to -7V
+7V
C31
47uF
R37
10K
U12
EN
REF
SS
COMP
VCC
OUT
GND
FB
TPS6755
130K
R35
C29
0.1uF
300K
R34
C30
10uF
10.2K
R36
C34
0.1uF
59K
R39
C37
0.1uF
L1
0805AF
SM5817
D77
-7V
100uF
C39
Power Solutions
TITLE:Electronic Learning Guitar
Document Number:DT03-MAIN-03
REV:B
Date: 2/5/2015 1:00:24 PM
Sheet: 3/3

Appendix C: GUI Screenshots

The application window on successful tuning



Master Status:

Guitar is in tune. You may now confirm chords.

Select Root Note: E

Select Scale: Major 7

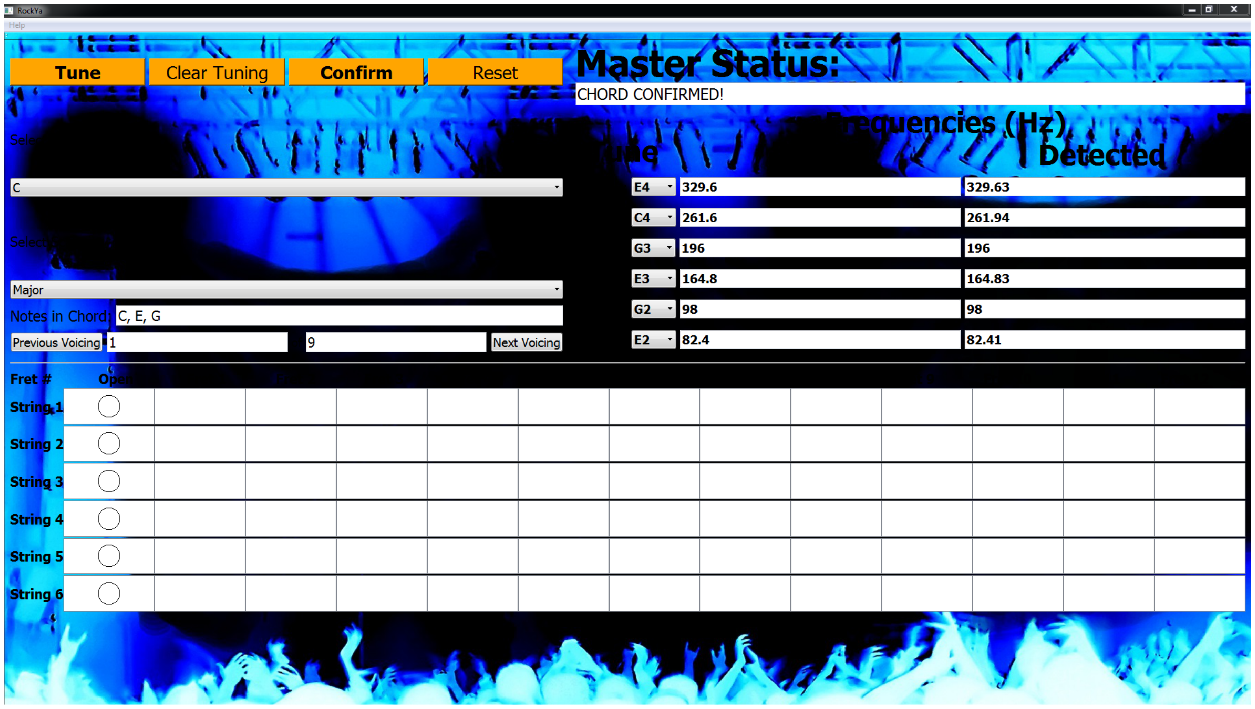
Notes in Chord: E, G#, B, D#

Previous Voicing 1 of 2 Next Voicing

	Frequencies (Hz)	
Tune	**Exact**	**Detected**
String 1: E4	329.6	329.63
String 2: C4	261.6	261.94
String 3: G3	196	196
String 4: E3	164.8	164.83
String 5: G2	98	98
String 6: E2	82.4	82.41

Fret #	Open	Fret 1	Fret 2	Fret 3	Fret 4	Fret 5	Fret 6	Fret 7	Fret 8	Fret 9	Fret 10	Fret 11	Fret 12
String 1								●					
String 2									●				
String 3										●			
String 4								●					
String 5									●				
String 6								●					

The application window on successful confirmation



Master Status:

CHORD CONFIRMED!

Sel

C

Selec

Major

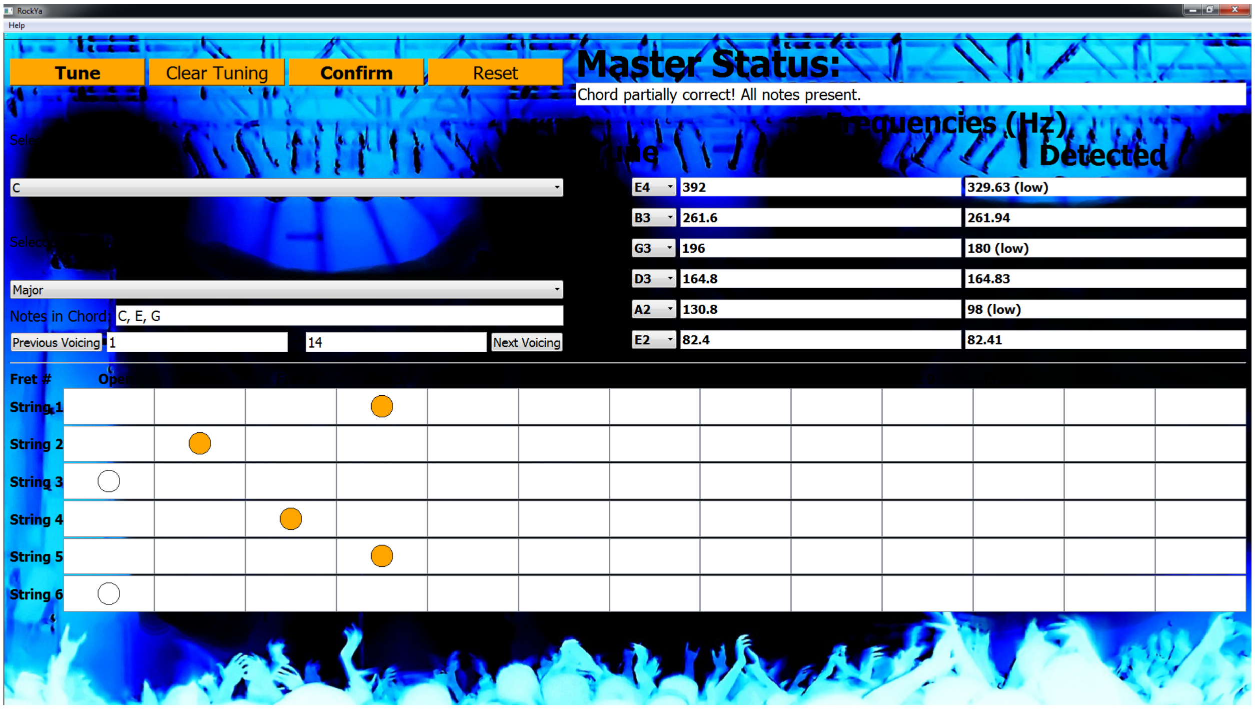
Notes in Chord: C, E, G

Previous Voicing 1 9 Next Voicing

	Frequencies (Hz)	
une		**Detected**
E4	329.6	329.63
C4	261.6	261.94
G3	196	196
E3	164.8	164.83
G2	98	98
E2	82.4	82.41

Fret #	Open												
String 1	○												
String 2	○												
String 3	○												
String 4	○												
String 5	○												
String 6	○												

The application window on partial confirmation



Chord partially correct! All notes present.

C

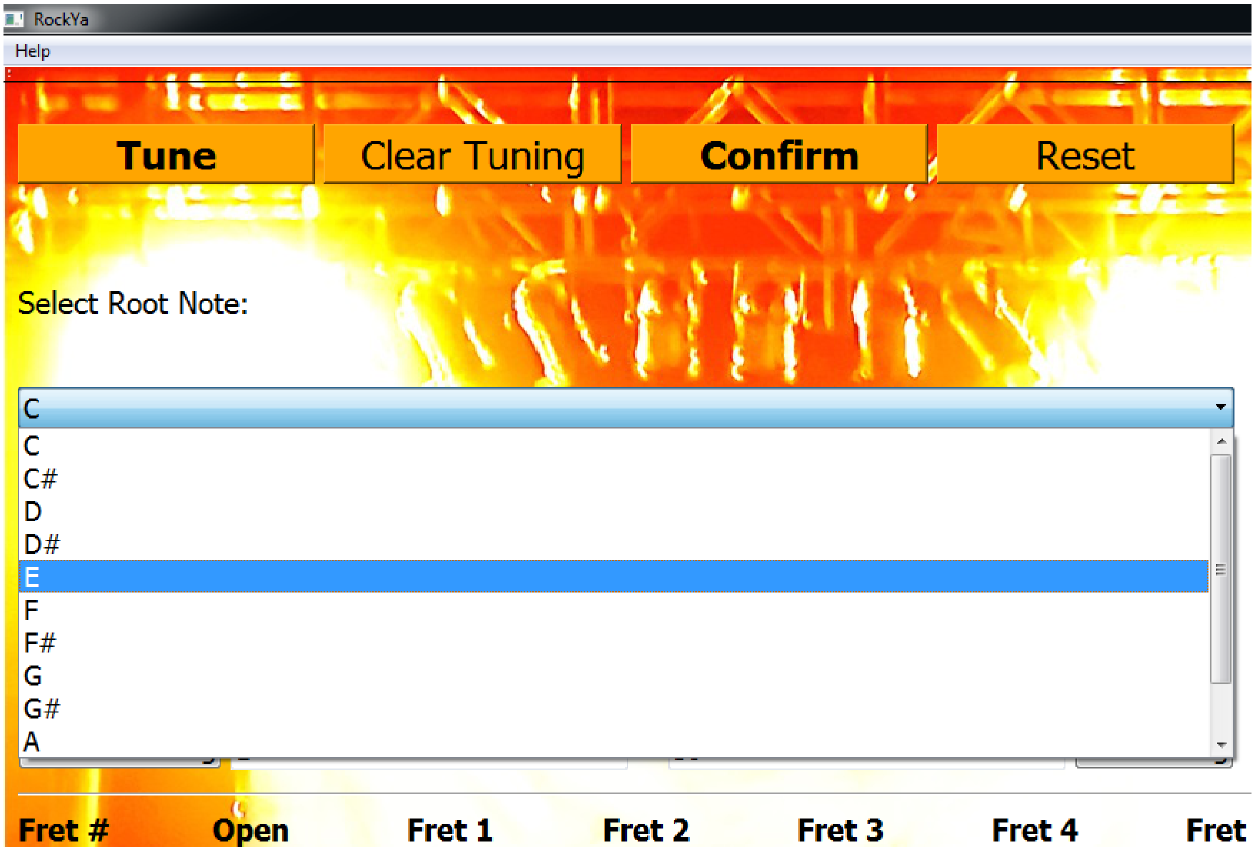
Major

Notes in Chord: C, E, G

Previous Voicing 1 14 Next Voicing

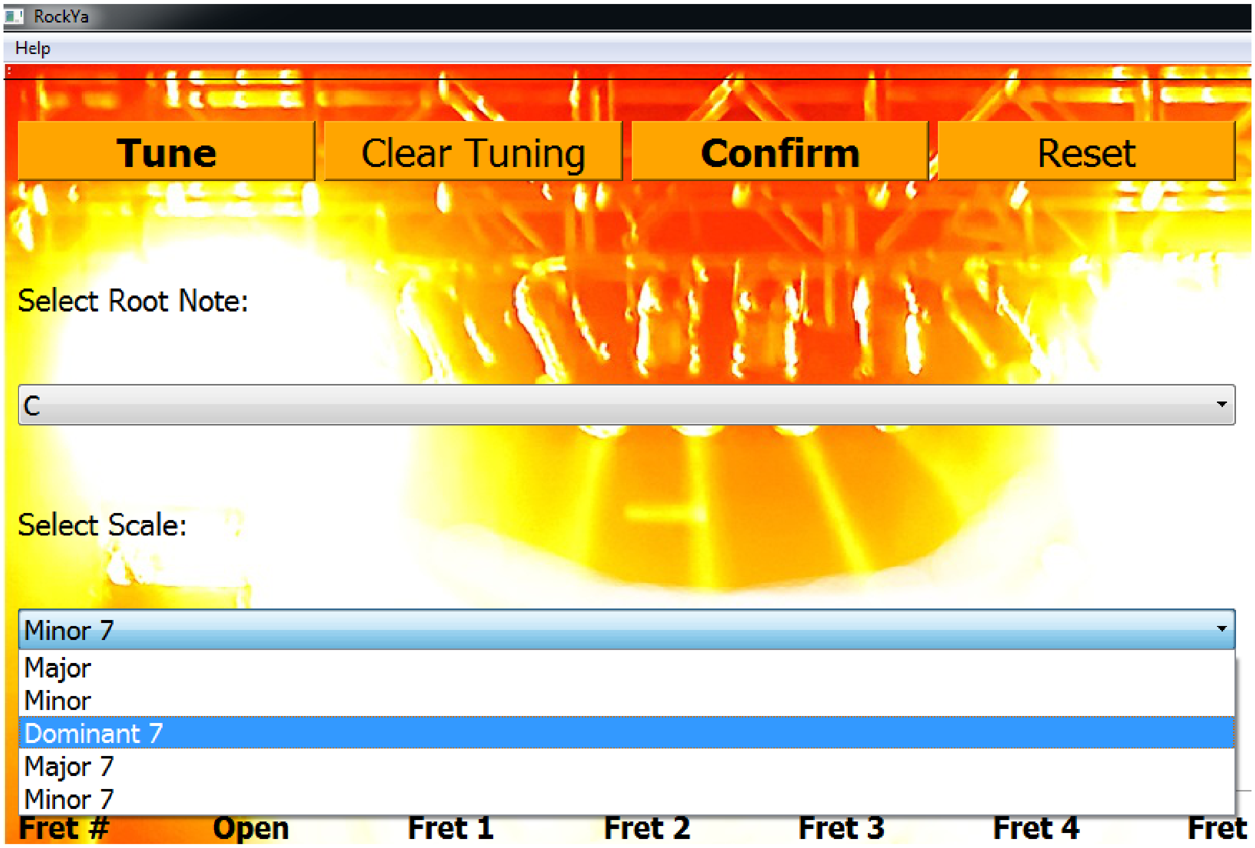
		Detected
E4	392	329.63 (low)
B3	261.6	261.94
G3	196	180 (low)
D3	164.8	164.83
A2	130.8	98 (low)
E2	82.4	82.41

Selection of root note



Select Root Note:

Selection of scale



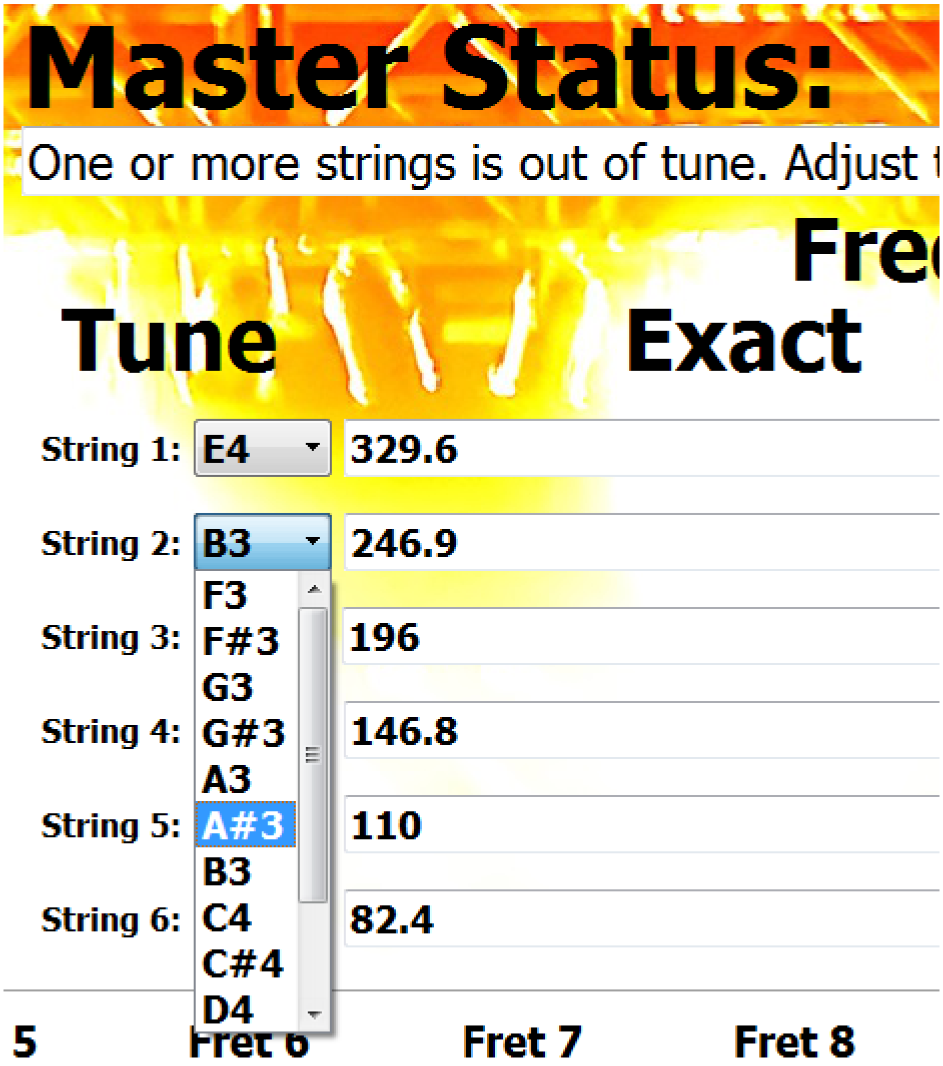
Select Root Note:

C

Select Scale:

Minor 7

Selection of desired tuning



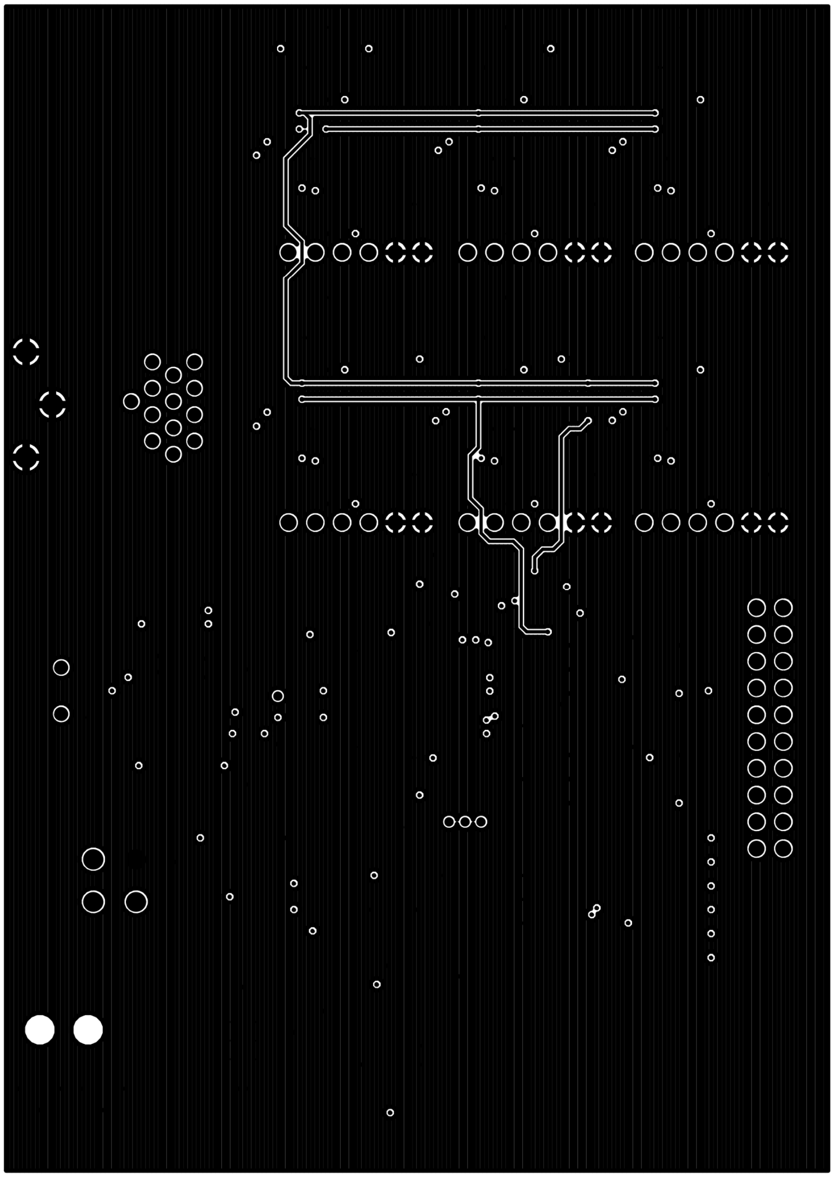
Master Status:

One or more strings is out of tune. Adjust t

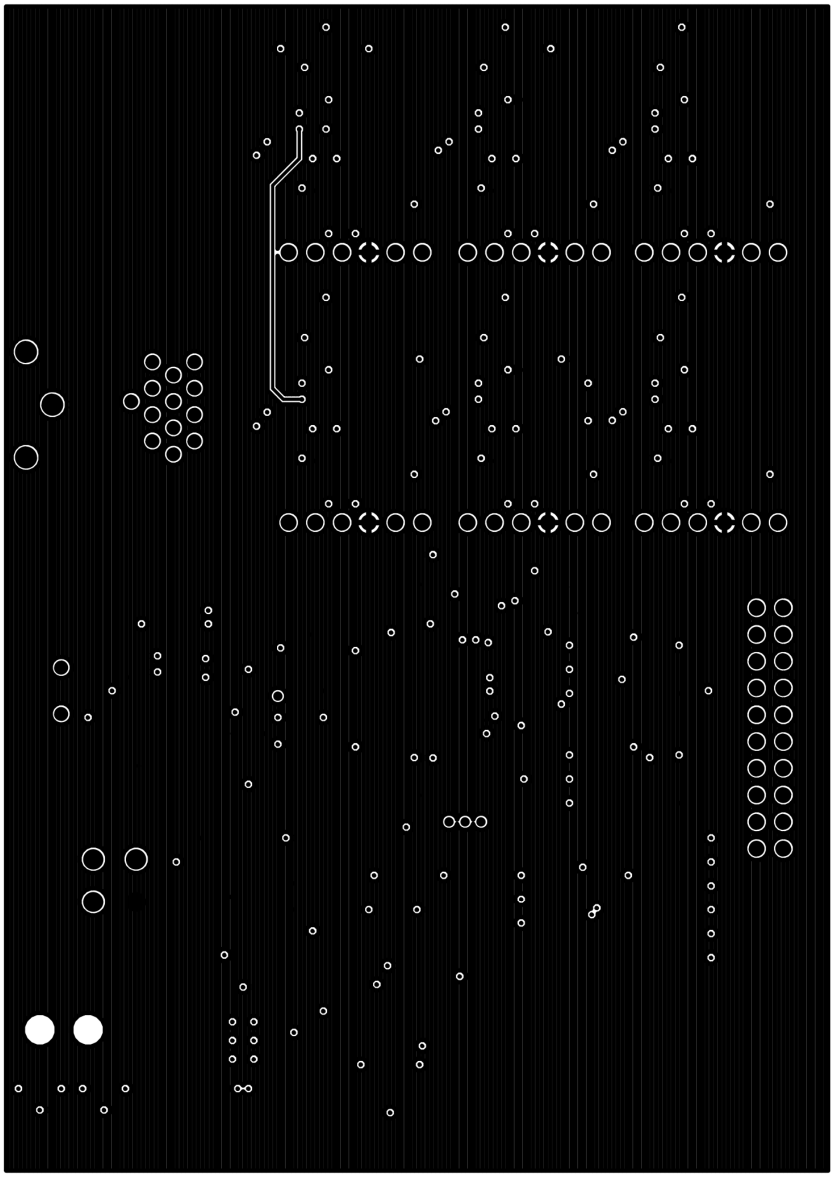
	Tune	Fre Exact
String 1:	E4	329.6
String 2:	B3	246.9
String 3:		196
String 4:		146.8
String 5:		110
String 6:		82.4

Appendix D:
Guitar Mounted Circuit Board Gerbers

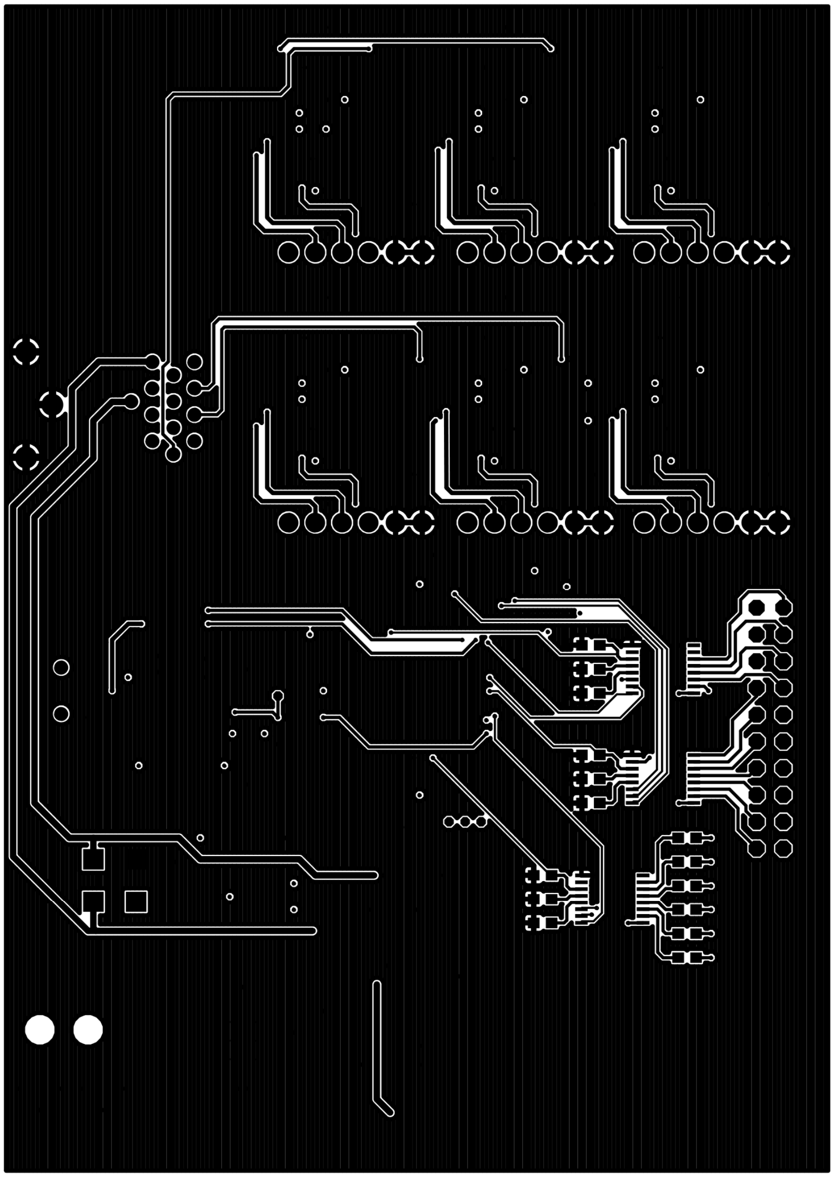
(See following pages)



ELG_MAIN.G2L Scale=2.35 Wed Apr 08 21:03:30 2015

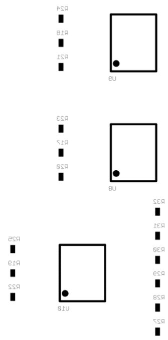


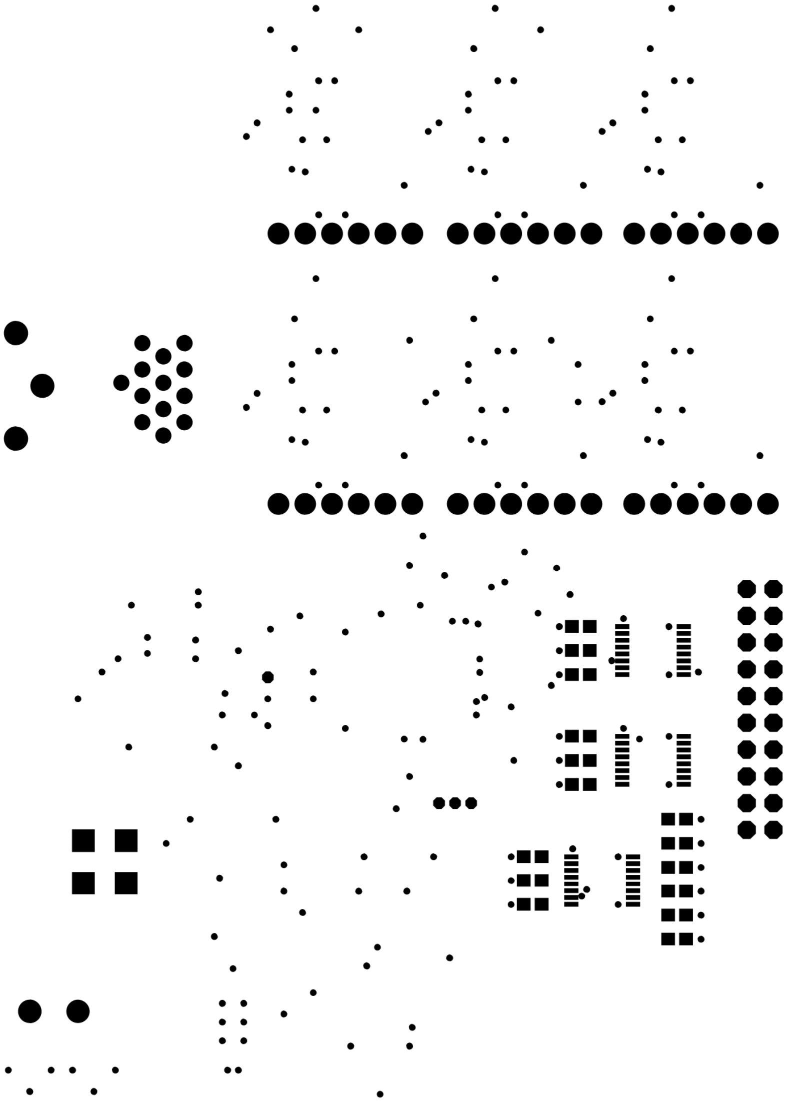
ELG_MAIN.G3L Scale=2.35 Wed Apr 08 21:03:30 2015



ELG_MAIN.GBL Scale=2.35 Wed Apr 08 21:03:30 2015

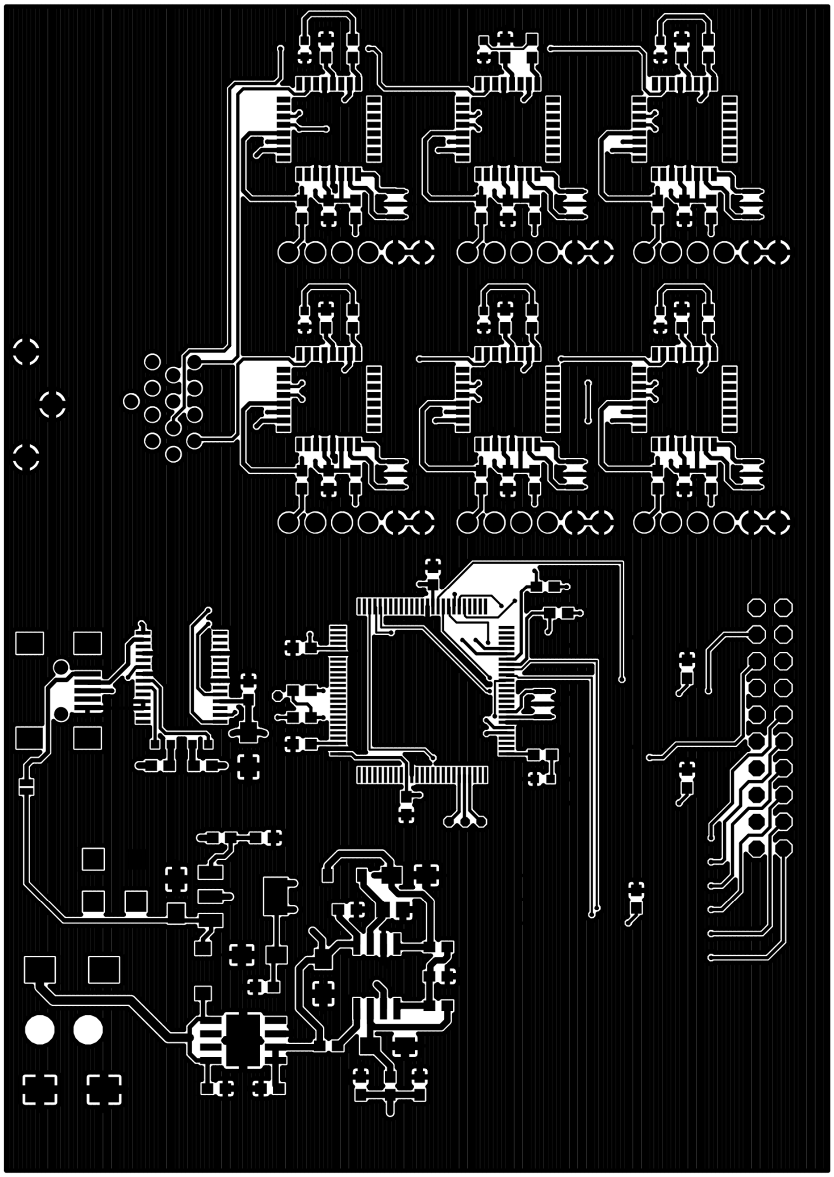
ELG_MAIN.GBO Scale=2.35 Wed Apr 08 21:03:30 2015





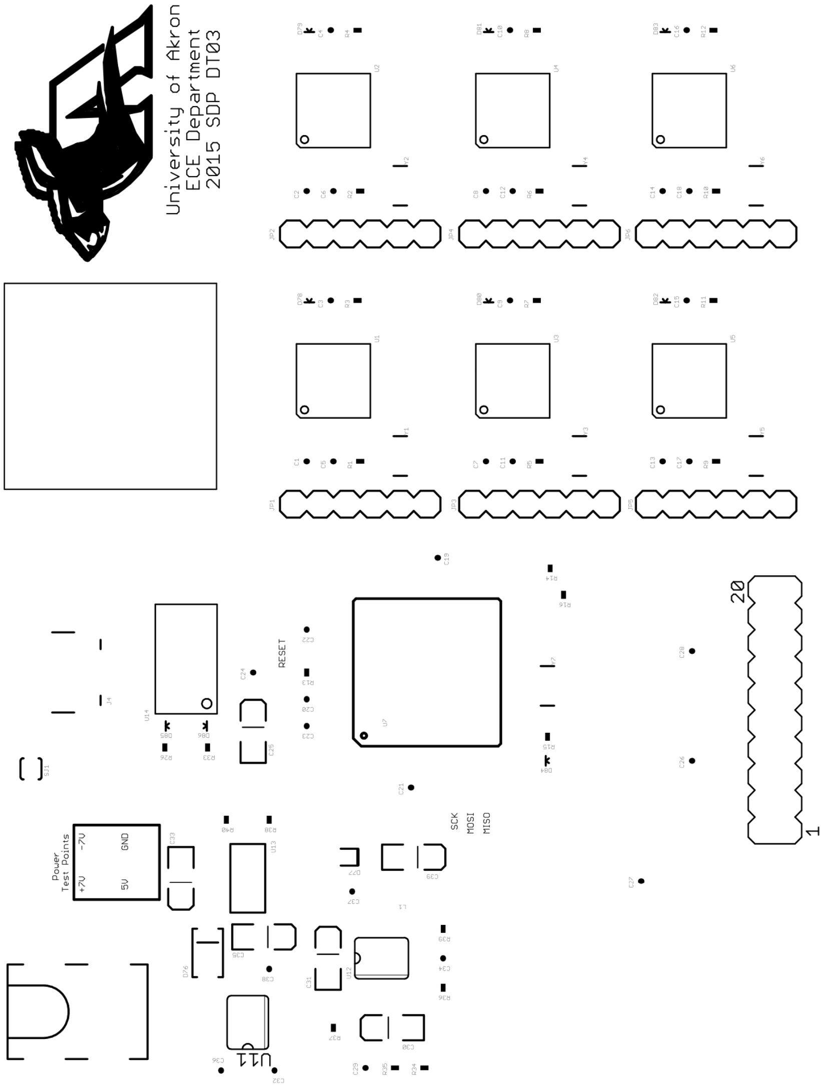
ELG_MAIN.GBS Scale=2.35 Wed Apr 08 21:03:30 2015

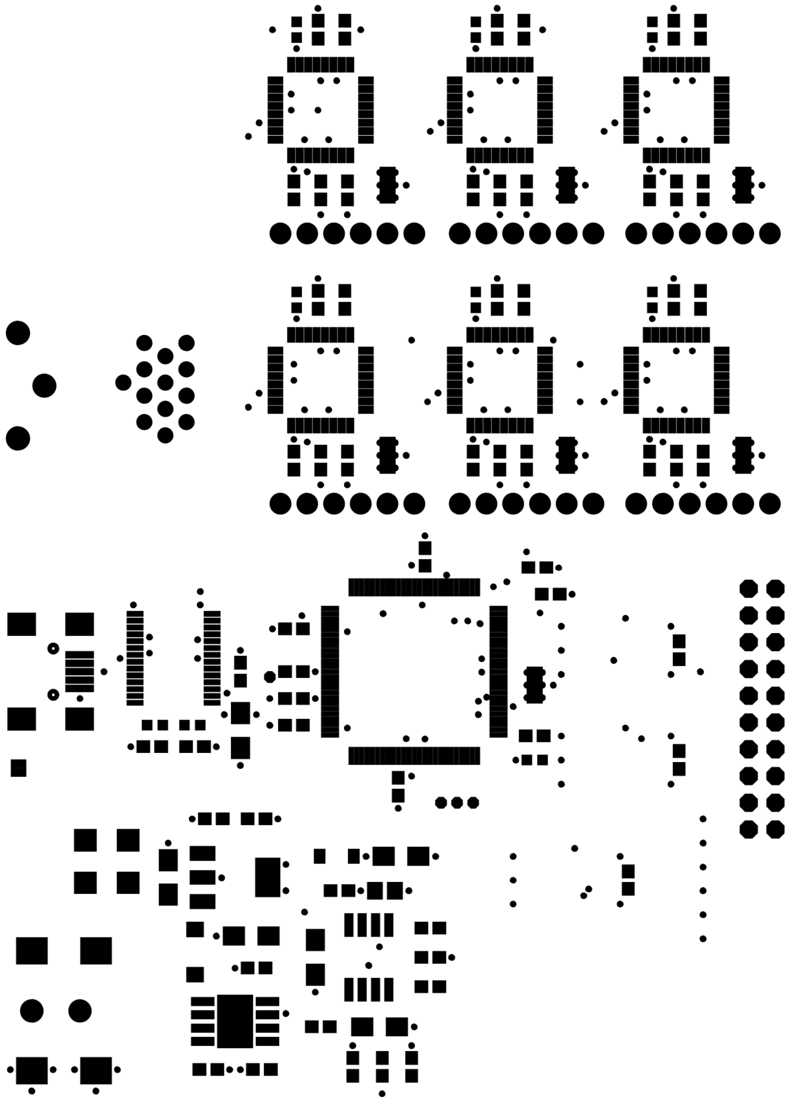
ELG_MAIN.GKO Scale=2.35 Wed Apr 08 21:03:30 2015



ELG_MAIN.GTL Scale=2.35 Wed Apr 08 21:03:30 2015

ELG_MAIN.GTO Scale=2.35 Wed Apr 08 21:03:30 2015





ELG_MAIN.GTS Scale=2.35 Wed Apr 08 21:03:30 2015