Automatic Guitar Tuner

Trenton Ahrens, Alex Capo, Ernesto Wong

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450 U.S.A.

***Abstract* — This document describes the design of the Automatic Guitar Tuner, which include specifications and procedures for constructing and testing. The project involves the use of signal processing and amplification, integrated circuit manipulation, and Bluetooth integration with an Android user interface. The aforementioned processes are utilized in unison to control the bass guitar's sting frequencies via DC gear motors. This paper also shows results and data of the procedural approaches such as waveforms, frequency ranges, and schematics.**

***Index Terms* — analog-digital conversion, analog integrated circuit, Bluetooth, DC motors, frequency, signal processing algorithm.**

I. Introduction

The practical idea of automatically tuning a guitar explores a wide range of technologies and processes. The project targets beginner guitar players to assist in tuning a bass guitar to a certain, pre-set tuning preference. Intricate hardware and software applications are utilized to bring together a complex system. The overall goal of Automatic Guitar Tuner project is to have the user automatically tune the bass guitar by selecting a tuning preference through an Android mobile application that applies Bluetooth wireless technology. The project uses the Android Studio Integrated Development Environment (IDE) for the development of the Graphic User Interface (GUI). The Java language is the default language for the IDE. Using the common built-in Bluetooth capabilities found in most mobile devices, the communication is established between the mobile device and the Bluetooth module which is connected to the PCB containing the Atmega328p chip. The Atmega328p is the 'brains' of the system, as it is responsible for the analog-digital conversion (ADC) and signal processing algorithm. As the user plucks a string to commence the automatic tuning process, the analog signals being inputted from the pickups of the bass guitar are converted to digital signals. The digital signals are then analyzed for frequency so that it may be processed and outputted for determining the strings correct tension, controlled by standard DC gear motors mounted on the headstock of the guitar. The entire process is repeated on the remaining strings. The results of the entire procedure brings forth a perfectly tuned bass guitar that is aesthetically pleasing to the ear.

II. System Components

To understand the project in its entirety, it is easiest to overview each component and subsystem individually. The later sections will describe the components in depth with technical details.

A. Motors

The tuning mechanism of the Automatic Guitar Tuner was driven by DC gear motors, the ROB-12472 standard gear motor to be exact. These motors were used because they were cost efficient and also capable of turning the pegs of the guitar at their highest load.

B. H-Bridge

The Automatic Guitar Tuner required its tuning mechanism to have bidirectional rotation capability. In order to achieve this requirement, the system needs four total H-Bridge motor drivers. The L293DNE dual H-Bridge by Texas Instruments was used to meet these requirements. Each integrated circuit (IC) contained two H-Bridges, so only two chips were needed.

C. Power

The Automatic Guitar Tuner requires a couple different power sources to operate correctly. Battery power was the means of power decided on by the team. An 8xAA battery pack holster and two 9V batteries were used to make this work. Also, a LM7805CT voltage regulator was included to drop down voltages from the battery pack for certain components that use a lower voltage.

D. Microcontroller

The brains of this project is the Atmel Atmega328p microcontroller (MCU). This chip was chosen for its small form factor, its ease of programming and interfacing, its ADC inputs, and the necessary outputs. An Arduino dev board was used to program and test the Atmega chip, as well as the Arduino IDE.

E. Op Amp

The analog signal received from the guitar needs to be amplified before it can be analyzed by the microcontroller. This is accomplished by using a TL082CP op amp from Texas Instruments. This is a widely used and very common general purpose operational amplifier. It is very low cost, requires low supply current, and has a large gain bandwidth product.

F. Bluetooth

The project aims to have wireless communication from the user interface to the Atmega328p. The low cost and low power consumption provides an adequate medium for communication. The built in Bluetooth capabilities within the mobile Android device allows the user to send serial data through Bluetooth's radio technology called frequency-hopping spread spectrum. The Atmega328p receives the incoming serial data from the user via the HC-05 Bluetooth module, which is the ideal choice for integration onto the PCB.

G. Android Application

The user interface is an Android application designed and developed using the Android Studio IDE. The Android platform is open source which made development boundless. The Java programming language, which utilizes object-oriented structuring of data, is used as the default programming language for the Android Studio IDE. Android application is set to run on Android version 4.4 and up. The Android application utilizes the mobile device's built-in Bluetooth capabilities to allow communication to the Atmega328p. The application allows the user to select from a set of five tuning preferences with a click of a button for each preference.

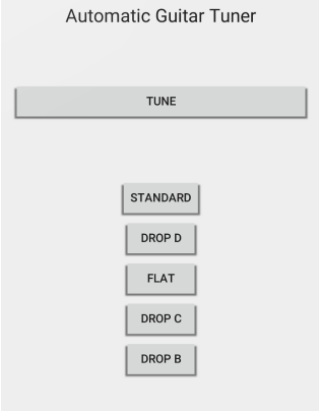

Fig. 1. Depicted above is the GUI that the user will be interacting with when selecting the tuning preference.

H. Bass Guitar

This project is designed around using a standard 4-string bass guitar. The bass guitar consists of four strings, tuning pegs, an auxiliary jack, and pickups. The strings are a standard gauge round wound configuration. No external pickups were necessary while designing the Automatic Guitar Tuner because the built in set worked fine. The input jack is connected directly to the pickups so no additional circuitry was needed to listen to the string's signal.

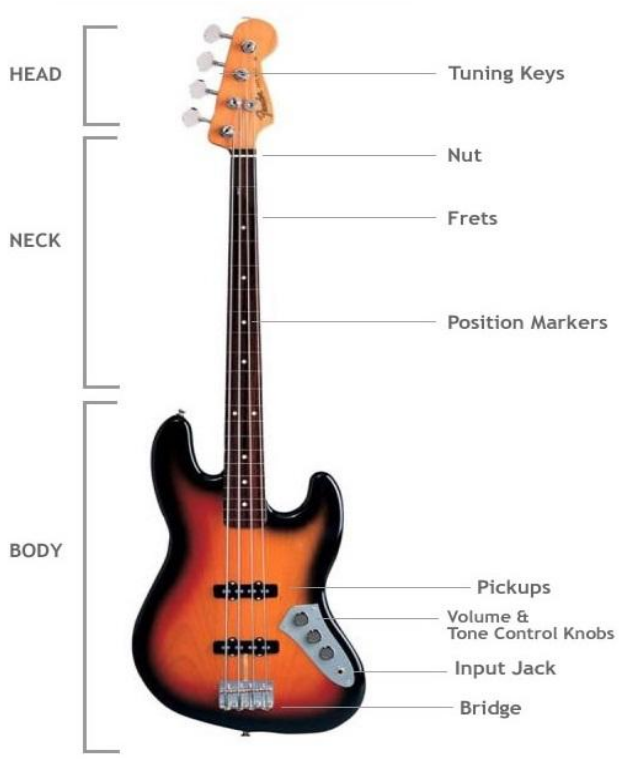

Fig. 2. Shown above is the standard design and anatomy of a bass guitar.

III. SYSTEM CONCEPT

In order to obtain a visual interpretation of how the system works as a whole, a flow chart of how the Automatic Guitar Tuner works is displayed below.

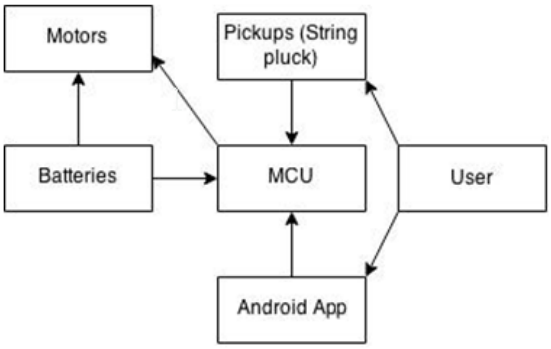

Fig. 3. Complete flowchart of the system illustrates how the user interacts with the guitar as well as how the subsystems are linked together.

It is displayed in Fig. 3 that there are six main parts to the system as a whole. The user will be interacting with

two parts of it only, which are the mobile Android application and the guitar pickups. Both the mobile application and pickups then interact with the MCU, which applies the appropriate algorithms from each input. Finally, based upon the output of the algorithms, the particular command will be outputted to the motors.

The mobile application will allow the user to select the desired frequency settings to be obtained. In the mobile application, the user will select one of the five preconfigured settings being: Standard, Drop D, Drop B, Drop C, and Flat tuning. As far as the user's interaction with the pickups go, this refers to the plucking of the string and the frequency in which the pickups read from the string. The pickups read the vibrations from the string as a voltage, and based upon the signal being read, is compared to the correct frequency. The MCU's frequency detection algorithm then outputs either a forward or reverse signal to the motors, which rotate accordingly until the frequency being read matches the correct setting.

A. Tuning Mechanism Concept

Now that an overall concept has been portrayed, further detail can be explained on the system's tuning mechanism. The MCU will determine whether the motor being used needs to rotate clockwise or counterclockwise. This decision is made by comparing the current string frequency to the actual frequency it should be at. Based upon this difference, the MCU will write the appropriate pin high, and the other low for forward, and vice versa for reverse. Once the MCU recognizes that the string is in tune, both the forward and reverse pins will write to a logical low to brake the motor. Shown below is the flowchart of how exactly this action occurs.

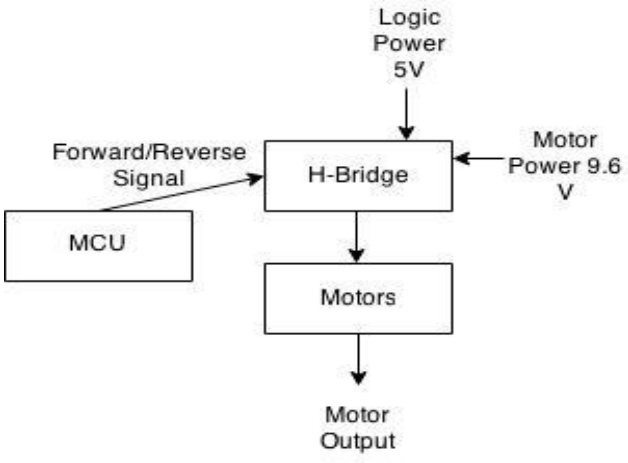

Fig. 4. Representation of how the systems tuning mechanism operates during the tuning process.

B. Power Distribution Concept

It was depicted in prior pictures on how the power is being distributed throughout the system, but it is only a vague interpretation. The system uses two separate battery sources, the battery holster and the series configured 9 V batteries. The 9 V batteries are only being used on the op amp, whereas the battery pack is used for a number of inputs. Below is a diagram of exactly how these two sources are being used in the system.

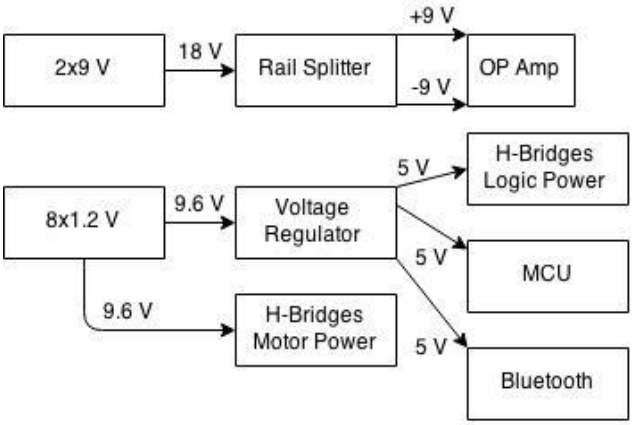

Fig. 5. A complete representation of how the power is being distributed throughout the PCB.

C. MCU Concept

It is important to understand what exactly the system's MCU is doing during the tuning process. This is the driving force of the entire system because it acts as the decision maker for it. In addition to making the decisions, it also has all of the systems operating code stored onto it.

The code stored onto the MCU consists of different parts, mainly being the frequency detection portion, the comparison portion, and the Bluetooth transmit and receive serial data. In the following figure it is illustrated as a flow of operations, showing the analog input that is read from the MCU's ADC channels being fed through the signal processing algorithm. Once the sampled signal is realized, it is compared to the controlled signal that is stored into the MCU's memory, which is selected from the Android application. After the difference in frequency is confirmed, the MCU then outputs the appropriate forward/reverse signal to the motor to correct the difference.

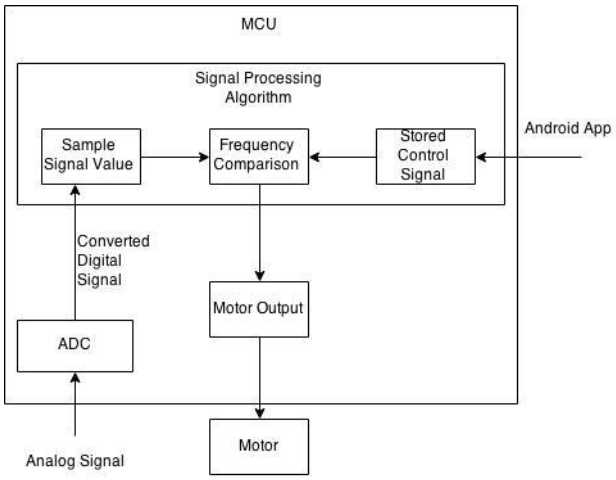

Fig. 6. The representation of the system's microcontroller flow of operations.

IV. HARDWARE DETAIL

Each major system component that was outlined in Section II: System Components will now be explained in depth with more technical information.

A. Motors

The selection of motors is an important part for the Automatic Guitar Tuner. They should not be too heavy that the user feels discomfort while playing the instrument, and also should be capable of supplying the required torque in order to turn the tuning pegs of the guitar. It was discovered on a website that the heaviest string would need about 37 lb-in of torque to turn the tuning peg. [1]

There were issues initially when considering motors to use for the design, mostly being that small enough motors were not capable of turning the tuning pegs. A gearbox could have been used to overcome the inadequate torque output of the stepper and servo motors, but the area in which is available on the headstock of the guitar restricted this from being a viable option.

The ROB-12472 standard gear motor not only met the torque requirements, but also rotated at such a speed that worked out nicely for the systems frequency detection algorithm. At six rotations per minute, the frequency read algorithm was able to operate in parallel with the motors as they turned, and the frequency change is subtle enough at this rotational rate that accurate measurements are able to be obtained. The motors are capable of delivering up to 38 lb-in of torque and proved to be plenty to turn the heaviest string at its highest required tension for each tuning preference. Another notable feature is the wide input voltage range that the motors accept being from 3-12 V.

The motors were mounted onto the guitar with a custom made sheet metal bracket. This bracket was designed in Autodesk Inventor and cut out by hand with a drill press and a jig saw. The motors were attached to the bracket with 6-32 screws. Holes were drilled into the sheet metal and then tapped with the correct thread size allowing the motors to be attached to the bracket. The motor shafts were connected to the tuning peg by attaching a flathead bit to the motor and making a notch in the back of the tuning peg. A Dremel was used to create a notch on the back side of the tuning peg, allowing a flathead bit to fit securely. The D-shaft on the motor was connected to a flathead bit using a coupler, allowing the motor to properly turn the peg.

B. H-Bridge

The frequency of a string that is out of tune can be either above or below the desired frequency, which means that the motors require bidirectional rotation capabilities. In order to achieve this requirement, the system's final design needs four H-Bridges in order to control all four motors. Building an H-Bridge from scratch was an option in terms of design, but the size profile would be much larger than what was desired. In order to work around this and to make the most convenient experience for the user, IC's made more sense.

The L293DNE dual H-Bridge by Texas Instruments proved to be the exact component needed. Only two total IC's were needed because a single IC was capable of handling the bidirectional rotation of two separate motors. The chip is capable of outputting up to 1A of current and has motor voltage supply range of 4.5-36 V. Because the chip was capable of outputting enough current to turn the motors at their highest loads, the final PCB design did not require and current amplifying components such as transistors, which yielded a simpler schematic.

The L293DNE bridges are dual in-line package (DIP) IC's with sixteen pins. Below is a list that describes each of the pins functionality.

(1) & (9) - Motors 1 and 2 logic control, respectively.
(4) (5) (12) & (13) - Ground.
(2) & (7) - Forward/Backward turn logic for motor 1.
(10) & (15) - Forward/Backward turn logic for motor 2.
(3) & (6) - Motor 1 input leads.
(11) & (14) - Motor 2 input leads.
(8) - Motor power at 9.6 V.
(16) - Logic power at 5 V. [2]

C. Power

There are multiple components in the design that require some amount of input voltage to operate. This amount of input voltage varies per components and thus required voltage regulation. In particular, the ATMega328P, logic power for the L293DNE, and the HC-05 Bluetooth module all use 5 V inputs. The L293DNE also requires the voltage input that the motors will be using to operate which is anywhere from 4.5-36 V. Finally, the TL082CP op amp required a +/- 9 V input.

The motors proved to be capable of moving the load with the 9.6 V (8x1.2 V NiMH AA batteries) input directly fed from the battery holster. The aforementioned voltage regulator was used to drop down this 9.6 V to 5 V to use as an input for MCU, the H-Bridge's logic power, and the Bluetooth module. The final component that used power was the systems op amp, which required a +/- 9 V supply.

To power the op amp, a supply of an equal plus and minus voltage was required. To accomplish this, a TLE2426 rail splitter was used. This IC works by taking a voltage supply and splitting it into two equal positive and negative voltages by creating a virtual ground. This was preferred over using two batteries individually or a resistor divider, because either of these can become unbalanced and cause one battery to drain faster than the other. The rail splitter also has a noise reduction pin with a 1µF capacitor. Below is a schematic showing the layout of the rail splitter.

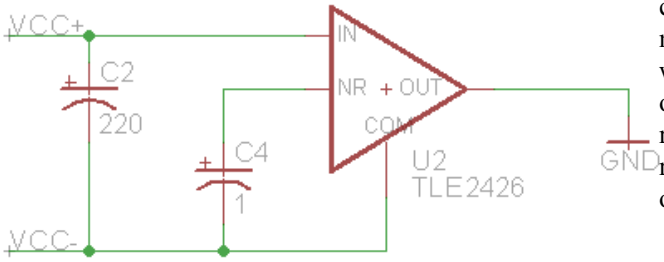

Fig. 7. Schematic of TLE2426 Rail Splitter.

The VCC+ is the positive end of the two 9V batteries and VCC- is the negative end of the two batteries. When the output is connected to ground, the VCC+ is at 9V when compared and VCC- is at -9V when compared to ground.

D. Microcontroller

The Atmega328P is the component that ties all the other systems together. The microcontroller is powered with 5V from the voltage regulator. It is run with a clock speed of 16MHz from an external crystal oscillator. The microcontroller receives the analog guitar signal from the op amp and the signal is processed by the analog to digital converter. It communicates with the HC-05 Bluetooth module via the TX and RX pins. The microcontroller uses eight digital output pins to signal the H-Bridges to control the motors.

E. Op Amp

The TL082CP op amp is powered with a +/- 9V supply from the rail splitter. The op amp is connected in a non-inverting configuration as shown in the following figure.

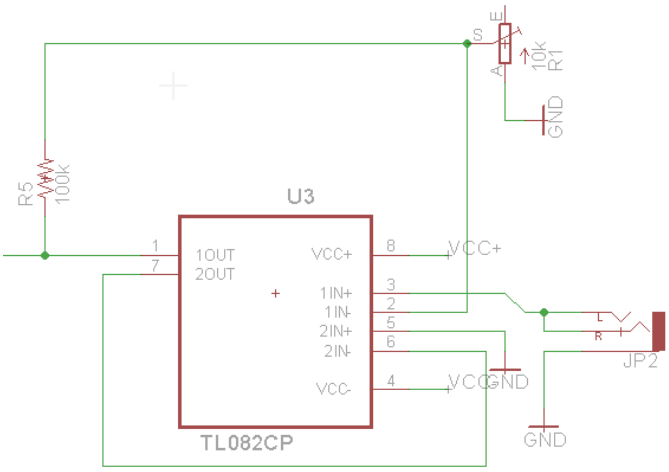

Fig. 8. Schematic of TL082CP Op Amp.

The positive signal from the guitar pickups is connected to the positive terminal on the op amp. The negative terminal is connected to a 10kΩ potentiometer which is connected to ground. This allows the gain of the op amp to be adjusted easily to ensure the Arduino receives a full signal from 0-5V. On the output, a 100kΩ resister is connected to the negative input. The gain of the op amp can be calculated by the following equation:

$$A = 1 + \frac{R_2}{R_1}$$

On the output of the op amp there is a 10µF capacitor to block an DC voltage and ensure that the signal is centered around 0V. After the DC blocking capacitor, there is a voltage divider that consists of two 100kΩ resistors that center the signal around 2.5V. This ensures that the analog to digital converter on the microcontroller receives the whole signal as anything above 5V or below 0V cannot be read by the ADC.

The TL082CP package contains two individual op amps. The unused op amp cannot simply be left floating,

as this can affect the other op amp. They cannot simply be tied to ground either. To solve this problem, the positive input of the unused op amp is tied to ground, and the negative input is tied to the output.

F. Bluetooth

Along with the built-in Bluetooth capabilities on the mobile Android device, the HC-05 Bluetooth module is used by the Atmega328p to allow serial communication from the Android application. The HC-05 operates between 3.3 V to 5V. It consists of Tx and Rx pins which provide transmission lines of UART data input and output to the Atmega328p. While pairing with the mobile Android device, the built-in LED on the module blinks at a rate of 2 Hz and 1 Hz for standby. The HC-05 allows the option to switch between master or slave mode through the AT commands. By default, the module is set and kept on slave mode, as the mobile Android device acts as the master.

V. SOFTWARE DETAIL

Each major system component that was outlined in Section II: System Components will now be explained in depth with more technical information.

A. Android Application

Developed using the Android Studio IDE, the Automatic Guitar Tuner (AGT) application serves as the user interface. Upon development, several files needed to be made for the application to compile as whole: (1)The Main Activity class file (2)The Main Activity extensible markup language and (3)The Android Manifest extensible markup language.

The Main Activity class file serves as the main source code for the application. It is where the libraries for Bluetooth are imported and where variables, or objects, and methods are defined to carry out the data transfer process from the application to the Atmega328p. Upon launching the application, the activity starts with the onCreate( ) method. Here, setContentView(int) is called with a layout resource defining your user interface, and using findViewById(int) to retrieve the widgets in needed to interact with programmatically. For the project, the application has five buttons representing five different tuning preferences the user can select from. For the first button, which represents Standard tuning, upon click, a character '1' is converted to bytes in the sendData( ) method, where the serial data is transferred to the Atmega328p via Bluetooth. The same is done for the remaining four buttons, however, each button sends a unique character corresponding to the set tuning preference. After the onCreate( ) method comes the onResume( ) method, which is called when the activity will start interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. For the project, this method retrieves the Bluetooth adapter and pairs with the HC-05 based on the MAC address. After onResume( ) comes onPause( ), which handles the activity for when the system is about to start resuming a previous activity. Here, onPause( ) checks and maintains Bluetooth communication with the HC-05. Any communication error that arises will close the Bluetooth sockets and notify the user. [3]

Along the class file is the extensible markup language file (.xml) where the layout of the application is designed. The file contains a layout view which allows the developer to design the GUI by placing various widgets, in this case the text fields and buttons, wherever is seen fit. The file also has text view, which contains the screen elements for each widget with a series of nested elements, similar to web pages in HTML.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Flat"
    android:id="@+id/button3"
    android:layout_below="@+id/button2"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Drop C"
    android:id="@+id/button4"
    android:layout_below="@+id/button3"
    android:layout_centerHorizontal="true" />
```

Fig. 9. A screenshot of the nested elements within a button.

The Android Manifest file presents essential information about the application to the Android system and information the system must have before it can run any of the code. The following is handled by the manifest:

- It names the Java package for the application, which is a unique identifier for the application.
- It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of. These declarations let the Android system know what the components are and under what conditions they can be launched.

- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It declares the permissions that other applications are required to have in order to interact with the application's components.

This file is where the permissions to use Bluetooth is established. [4]

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Fig. 10. The user permissions for Bluetooth.

B. C++ Code

The code for the project was done in the Arduino IDE in the Arduino version of C++. The code consists of several important parts; analog to digital conversion, frequency calculation, Bluetooth communication, and motor control.

The most important part of the code is that which calculates the frequency from the analog input. The analog signal from the guitar is converted to a digital value by the ADC. Whenever a new ADC value is ready, an interrupt service routine stops the program to execute a certain function. This function uses a method of threshold crossings to determine the frequency. If the frequency is increasing and crosses a midpoint (2.5V), it starts timing until it crosses that point again with a positive slope to calculate the slope of the signal.

The microcontroller communicates with the mobile application vie Bluetooth simply by performing a mySerial.read to read what is sent from the Android application. The application sends over characters based on what button is pressed. The program reads the character and performs a certain function based on what value was received. For example, if the tune button is pressed, the application will send a '1' character. When that '1' is read, it begins tuning the guitar.

TABLE I
Standard Tuning For a Four String Bass Guitar [5]

String	**Note**	**Frequency (Hz)**
1 (thinnest)	G	98
2	D	73.42
3	A	55
4 (thickest)	E	41.2

The motor control is done simply by writing the outputs to the H-Bridges, either high or low. In the beginning of the program, all the motor outputs are written low, which tells the H-Bridge not to turn the motors. When the frequency is read and it is determined that a string needs to be tuned, the program will write one pin on the H-Bridge high depending on whether the frequency is above or below the correct value. Once the correct value is reached, the motor will stop turning and the program will move on to the next string.

VI. BOARD DESIGN

All of the components in this project were designed to fit on a dual layer printed circuit board. This board includes the microcontroller, the two H-Bridges, the op amp, the rail splitter, the voltage regulator, power input jacks, an audio input jack, and the necessary passive components. The board was designed in eagle and printed by OSH park. It is roughly 100mm x 43mm in size. The board has sockets to connect to the Bluetooth module as well as the four motors on the guitar.

VII. CONCLUSION

This overall experience has been fulfilling, challenging and fun. The Automatic Guitar Tuner project is a final measurement of our knowledge that we've gained during our time at the university in the College of Engineering.

As engineering students, the team is naturally curious, especially in electronics. During the time spent at UCF and on this project, the team is able to apply their knowledge to handle a real world problem that catches the team's interest. Several meetings were taken place to evaluate the production and testing of the prototype, presenting certain experiences that cannot be obtained in the classroom. For example, collaborating as a team to compromise on a direction to take with certain subsystems is one of those experiences. These experiences are all fulfilling, in that there was something to be gained from each one.

Several challenges arose when constructing and testing the prototype. There were instances where the team had to order several HC-05 modules because one of the pins kept getting fried due excess voltage. Another instance is having the Arduino code to control three out of the four motors. These are only a portion of challenges that the team had to overcome in order to get the system to work as a whole. The issues were easy fixes, however identifying the problem was the biggest challenge.

As avid music lovers, the team enjoyed tackling a problem pertaining to music. This gave extra motivation to overcome a challenge that is common for musician, especially beginners. This project gave a chance for the team to strive for a solution in something they love.

THE ENGINEERS

Trenton Ahrens is a 22-year old graduating Electrical Engineering student looking to work for a company in the automation industry working with PLC's. The rides and shows industry is particular field of automation that he finds interesting.

Alex Capo is a 24-year old Electrical Engineering student graduating at the end of the Summer semester. All throughout his life, he has had a general interest in electronics of all kinds. As far as his future is concerned in the field, he hopes to acquire a job in either a power distribution field or communications systems.

Ernesto Wong is a 23-year old graduating Electrical Engineering student looking to pursue a career in Systems Engineering or Electrical Engineering. He has an interest in testing and integration due to experience in Systems Engineering at Lockheed Martin.

ACKNOWLEDGEMENT

The authors wish to acknowledge the assistance and support of Dr. Samuel Richie and the University of Central Florida.

REFERENCES

[1] D'Addario & Co (2008), "*String Tension 101*," Retrieved 18 September 2014, World Wide Web: http://www.daddario.com/DAstringtensionguide.Page?sid=9bc4dfd7-316a-41be-9d07-6d0fc48b94eb

[2] Texas Instruments Inc. (2004), "*L293, L293D Quadruple Half-H Drivers",* Retrieved 23 October 2014, World Wide Web: http://datasheet.octopart.com/L293DNE-Texas-Instruments-datasheet-93368.pdf

[3] Android (2015), "*Activity,"* Retrieved 15 October 2014, World Wide Web: http://developer.android.com/reference/android/app/Activity.html

[4] Android (2015), "*App Manifest,"* Retrieved 15 October 2014, World Wide Web: http://developer.android.com/guide/topics/manifest/manifest-intro.html

[5] Grant Green (2000-2002), "*Frequencies and Range,"* Retrieved 1 October 2014, World Wide Web: http://www.contrabass.com/pages/frequency.html