QuickTab Midyear Design Review

Joseph Biegaj, EE, John Bonk, EE, Lindsay Manning, EE, and Jacob Prescott, EE

Abstract— **QuickTab is a system that enables guitarists to produce tablature, a common form of musical notation that indicates finger position rather than pitch. QuickTab determines these finger positions based on the mechanical vibrations detected in the strings and body of the guitar.**

I. Introduction

TABLATURE is an alternative form of representing music for stringed instruments which simplifies the confusing notation associated with traditional sheet music into an easy and intuitive format that is perfect for beginners and experienced musicians alike. Unfortunately, the existing software that sets the convention for generating tablature, Guitar Pro[1], is both expensive and time consuming to learn. With QuickTab we eliminate both the need for expensive software and an arduous transcription process.

QuickTab allows the user to generate tablature by simply playing the desired musical phrase on a guitar. Designed to be lightweight and unobtrusive, QuickTab attaches onto the base of the guitar by the bridge. A User Interface enables the user to calibrate the device to the guitar's specific tuning as well as dictate the beginning and ending of the recording. QuickTab is designed to provide more accessibility and ease of use to the tablature creation process.

As a commercial tool, QuickTab is ideal for guitar instructors who want to give their students something to take home and practice after the lesson. By using QuickTab they will be able to efficiently create customized lessons for their students, allowing them to zero in on that particular student's areas of difficulty, resulting in more effective teaching.

QuickTab would also be incredibly impactful on the musically community as a whole. Services like Songsterr[2], which hosts an interactive, online repository of tab which anyone can add to or learn from, would benefit greatly from a product like this, as simplified tablature generation enables more people to contribute to the database, increasing the availability and quality of tab. The use of QuickTab heralds growth in the musical community as it provides users who might have been discouraged from learning guitar, due to the cost of buying music or a lack of musical education, with ready accessibility to a comprehensive library of tablature; the comparatively cheap and easy way to learn guitar.

So far there are no products in the market that can accomplish what we are trying to do. However several attempts have been made, the most recent and most like ours is *AutoTabber* from SDP15. Team *AutoTabber* attempted to use six hexaphonic pickups to capture the signal off each string which would then be fed into a microcontroller where the frequency spectrum would be used to determine the fret that was played on the string [3]. Another example comes from the German company *M3i technologies,* their laser pitch detector used lasers coming from the bridge of the guitar to determine the length of the string being played and thus the fret being pushed [4], It was set for commercial release in 2012 however like team *AutoTabber* it was ultimately unsuccessful.

What makes QuickTab different is how we are generating our signal, *AutoTabber* used pickups, *M3i Technologies* used lasers and QuickTab uses a single accelerometer, which senses the vibrations of the guitar as a note is played giving us a cleaner signal to work with and ultimately making it possible to generate the tablature we desire.

<u>Specification</u>	<u>Value</u>
ADXL345	
Weight	1.27g
Height	3.14mm
Length	25mm
Width	19mm
Power Usage	75.9□W (Active) 0.25□W (Standby)
Raspberry Pi	
Weight	45g
Height	10mm
Length	85.6mm
Width	56.6mm
Power Usage	4W

Table 1: Project Specifications

The specifications given in *Table 1* show that the electronics attached to the guitar will be lightweight and relatively small, accomplishing one of our goals for the project to be sufficiently small and lightweight so that when it is attached to the bottom of a guitar it will not be to cumbersome for the user. Power consumption is not issue at the moment because we are not using our own power supply. As a stretch

J. Biegaj from East Hampton, CT (E-Mail: jbiegaj@umass,.edu)
J. Bonk from Northborough, MA (E-Mail: jbonk@umass.edu)
L. Manning from Hopkinton, MA (E-Mail: lmanning@umass.edu)
J. Prescott from Mashpee, MA (E-Mail: jprescott@umass.edu)

goal we have discussed using a power supply we design ourselves which would mean again taking weight considerations into account as well as power usage by the device.

II. DESIGN

A. Overview

Our project consists of five main blocks: sensors, user interface, data logging, signal processing, and tab compiler. Each of these pieces is required for our project to be fully complete.

Before we go over each of the blocks in detail, we summarize the main functions of our system. When the user decides to create tablature, they power on the QuickTab and start/stop the recording using the user interface. As soon as they begin to strum the guitar, the sensors register activity and transmit their data to the Raspberry Pi. After recording the user's inputs on the guitar, a file is created by the Pi and is then transferred to our signal processing block where the data is parsed and analyzed using Short Term Fourier Transforms and logical comparators. Once the frequencies have been identified, the entire recording is turned into tablature and available to view.

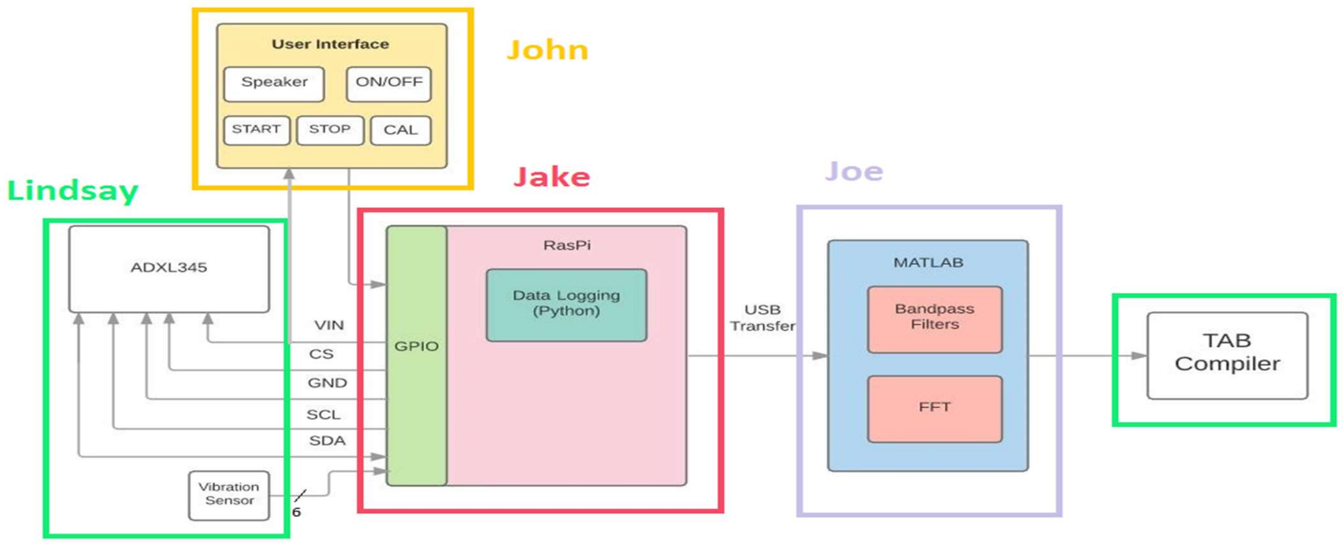


Fig. 1: Block Diagram

B. Sensors

The purpose of the sensors is to measure the mechanical vibrations of a guitar body and to detect when a string is used. Only a single accelerometer is required to measure the cumulative frequency of the guitar body at a given instance. The complex waveform output can then be broken down via Fourier transforms where individual Fourier components can be identified. Vibration sensors on the strings are designed to detect when a string is plucked. The vibration sensors must provide an impulsive output when the string is initially played. With the data received from the digital accelerometer and the vibration sensors, the information required to determine hand position is secured.

1) Accelerometer:

QuickTab utilizes a MEMS accelerometer to measure the vibration of the a guitar's body. The ADXL345[5] 3-axis digital accelerometer from Analog Devices measures the static acceleration of gravity, as well as, dynamic acceleration resulting from motion or shock. The device has a selectable measurement range of ±2 g, ±4 g, ±8 g, or ±16 g for each axis of the accelerometer; X, Y, and Z. Higher ranges allow for the tracking of high speed movements (i.e. vibrations). The device has an adjustable transfer rate up to 3.2kHz, which is suitable for our design since the guitar produces frequencies up to approximately 1kHz. The 32-level FIFO buffer allows the ADXL345 to store data this data to be read out at the user's discretion. The ADXL345 is surface mounted on a pre-assembled breakout board distributed by Adafruit. The package is positioned at the bridge of the guitar and oriented with the z-axis perpendicular to the strings. Our next steps regarding

the ADXL345 include refining the position of the device and its attachment to the guitar.

2) Vibration Sensors:

The next step for data acquisition is to determine what string is being used when a note is played. To attain this information, vibration sensors will be placed on each string. A SignalQuest SQ-MIN-200[6] is a small, lightweight, and omnidirectional vibration microsensor, which could be placed directly on a guitar string without interfering with the sound quality of the instrument. By attaching the sensors closer to the bridge, vibrations from surrounding strings will have less of an effect on an individual sensor.

Successful implementation of the sensors will be confirmed when the result of a single string being plucked is a single voltage spike on a computer generated graph. Next steps include physically attaching a sensor to a string and recording how much interference is received from other sources of vibration. The data received after attachment will determine if the SQ-MIN-200 is a suitable component for the system. The data from the vibration sensor should be in the form of an impulse. To achieve an impulsive output further signal processing must be implemented. In this form, the string used at a specific time can be recorded.

C. User Interface

The User Interface(UI) is an essential part of project QuickTab, allowing the user to control recording, calibration, and power, all with audible feedback. A custom PCB designed in EagleCAD will be used to implement this portion of the project, consisting of a series of buttons and a speaker. More specifically, it will require buttons for Start/Stop, Power On/Off, Calibrate, and a speaker to provide user feedback. The UI, containing the Raspberry Pi, buttons and speaker, will be placed at the bottom edge of the guitar, as illustrated inside the red box of Fig. 2. In order to implement the interface, we will draw from our circuits and electronics classes. The design for the buttons and speaker setup will be especially influenced by this past experience. Coding experience from our computer systems lab and data structures class will also be useful for this portion because the UI will need to be connected to the GPIO of the Raspberry Pi and control the running of scripts. Interrupts in C/Python are a keystone of this block and will need to be learned to ensure proper functionality of the buttons on the UI in coordination with the code for recording.

A test that can be done to prove functionality is to link several buttons to the Raspberry Pi and create a small test code that will print out “X button has been hit” upon button press. After implementation with the code used for recording data from the accelerometer, we will have visible verification that these interrupts are correctly triggered by their corresponding buttons. Another experiment is to take an oscilloscope and verify that when a button is pressed that a pulse is sent through the Raspberry Pi GPIO, verifying that

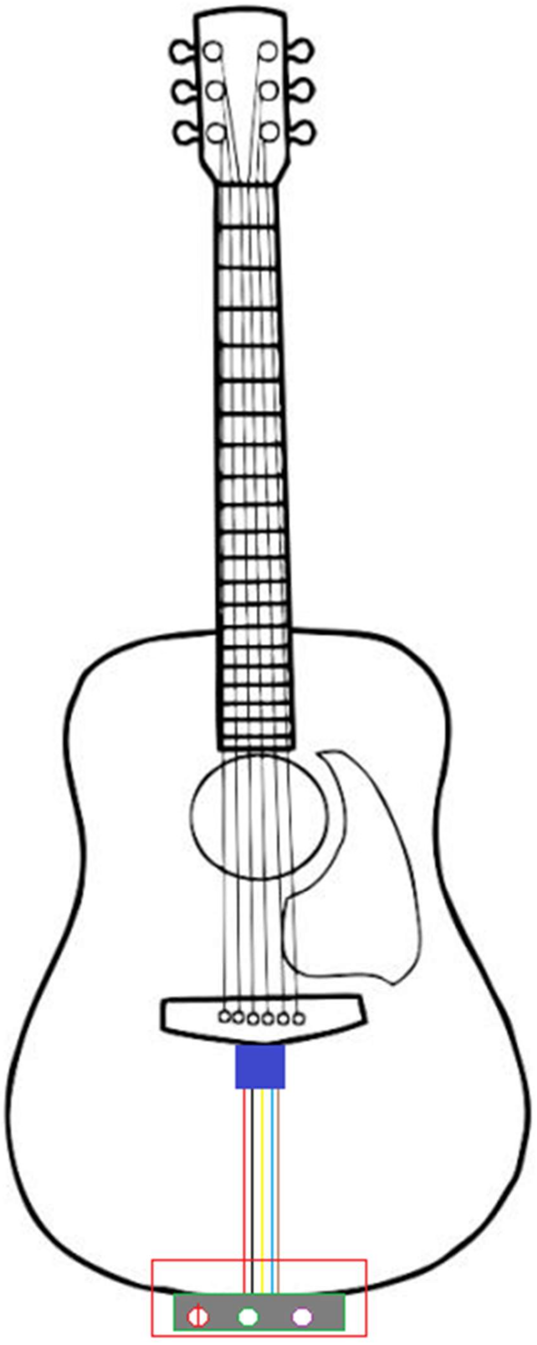
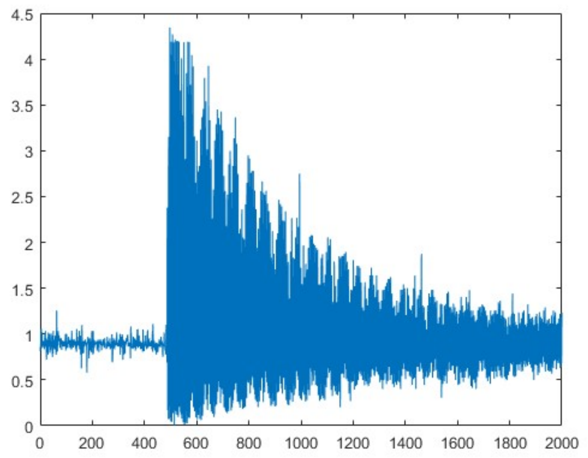


Fig 2: Implementation of the User Interface

the interrupt code will be able to trigger on a rising-edge. If the first experiment does not work and the sample code is having difficulty triggering an interrupt, then we will use the oscilloscope experiment to verify that a rising-edge is sent. If the interrupts are still not being triggered correctly, then there is something wrong on the coding side of the implementation. The UI should be a smooth, intuitive system that ensures functionality and interactivity for the project.

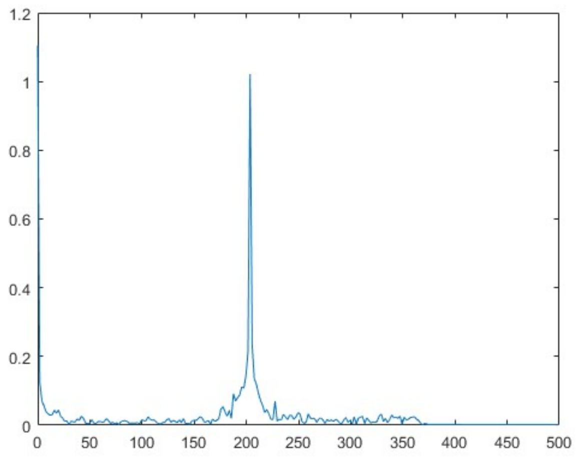
D. Data Logging

In order to reconstruct the guitarist’s performance in tablature form, the signals received from the accelerometer and vibration sensors are recorded by a Raspberry Pi that



accurately stores and organizes the data without missing any data points. Specifically, the method by which the data is collected must adhere to a few functional requirements. First and foremost, the data emitted by the ADXL345 must be collected by the Pi at the same sampling rate in order to accurately interpret every note played on the guitar. Additionally, the data collected from the accelerometer and vibration sensors must be aligned in the file with regard to time such that the code for generating tablature will have data for determining the frequency and string plucked as well as a timestamp that indicates that these two measurements are correlated and should work together to a new note on our tablature. The precise mechanism by which new notes will be added to the tablature will be discussed in Section F.

In order to properly collect and organize the received signals, we will be building upon the fundamentals that we learned in Data Structures & Analysis as well as any other



basic programming courses we have taken. Because our code utilizes Python, we must familiarize ourselves with the libraries available for data collection, as well as compiler differences such as an inability to explicitly set the clock rate. While software is the bulk of this technical block, it is important to consider the intrinsic link between hardware and software. Drawing from our experiences in Computer Systems Lab, we must manually configure our GPIO to operate under the parameters we give it and ensure that only digital signals are sent to the Raspberry Pi, as it does not support analog inputs without analog-to-digital conversion.

In order to ensure full functionality, this block requires three tests. Firstly, the ability of the Pi to collect samples at the same rate as they are emitted by the ADXL345 must be tested by recording samples of notes played at up to 1600Hz. If samples of these high frequency notes can be accurately recorded and converted to the proper frequency in the Matlab code, then the Pi must be working at the correct sampling rate of the ADXL345. Otherwise, we would not be sampling at the Nyquist Rate and would be unable to retrieve the correct frequencies.

Secondly, the vibration sensor data must be precisely logged with respect to time. We can verify accuracy by playing multiple notes with specific time delays between them and observing the spikes in our received signals for confirmation that these time delays are of the same duration.

Our third and final test brings the first two components together. In order to verify that both sensors are aligned, we must view their data side by side and ensure that the spikes in both sets of data occur at the same time. These three tests together will ensure the accuracy of our data logging both individually and collectively.

E. *Signal Processing*

After the data has been recorded by the Raspberry Pi it is off loaded to a PC where it is processed and the correct frequency is determined. Currently we are using MATLAB for our signal processing needs but as we refine and hone our code down to its final form we would like to switch over to a free software that would make our product far more available to the average person.

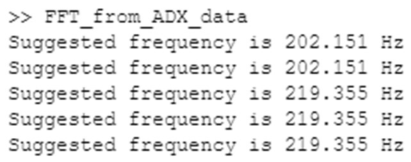
Fig.3: Raw ADXL345 Output of A3 Being Played

Fig. 3 shows data that is sent from the data logging stage to be utilized by the MATLAB code. First the data is sent through a number of bandpass filters, we do this to remove any extraneous noise and to eliminate any frequencies that cannot appear on the string being played, which we determine via the data gathered from the vibration sensors described in section II-B. This helps to refine the FFT as well as make it easier for the logical statements, which will be discussed later, to determine which note is being played.

Next we perform FFT's on the data that are 500 points wide and spaced 250 points apart. This overlap is to ensure that we do not miss any notes that happen to be in between the 500 point samples that we are taking. Fig. 4 shows the output after an FFT is taken.

Fig. 4: FFT Showing Note A3 (220Hz)

After each FFT is taken we send the data through a series of logical statements, using the function find peaks in order to determine the maximum spikes and from there determine the frequency of the notes that are being played. This process is repeated for the duration of the signal. Fig. 5 shows a MATLAB printout of this stage, after 5 FFTs are taken.



```
>> FFT_from_ADX_data
Suggested frequency is 202.151 Hz
Suggested frequency is 202.151 Hz
Suggested frequency is 219.355 Hz
Suggested frequency is 219.355 Hz
Suggested frequency is 219.355 Hz
```

Fig 5: Printout for the Above Signal, Displaying Frequency

F. *TAB Compiler*

The TAB Compiler is responsible for taking the data received from the signal processing and creating readable tablature. To achieve this, **(1)** will be used to determine the fret played at given instance. The first necessary component to produce accurate tablature is the frequency of the open strings. This data is obtained when the user calibrates the device. Calibration is important because even if the instrument is slightly out of tune, the fret can still be determined. Then, the string that was used when the note was played needs to be confirmed. If the string can be identified, then the compiler can determine what open string frequency to use in **(1)**. The next step would be to take the frequency determined via the signal processing method to calculate the number of semitones (n) that separate the note of the open string and the note played. Because each adjacent fret difference is exactly one semitone, we can determine the fret being played as equal to n.

$$n = 12\left(\frac{\log\left(\frac{F_{input}}{F_{openstring}}\right)}{\log(2)}\right) \tag{1}$$

Tablature, such as the example depicted in Fig. 5, uses lines and numbers to represent the six guitar strings and the frets used to create a note. To reproduce this form, arrays that correspond to each of the six strings will be filed with n values in the order that they occurred in. The values will be printed in the form seen in Fig. 5. The TAB Compiler will be completely implemented when the system is able to determine what string is used when a frequency is measured. The output of the TAB Compiler is the final product of the QuickTab.

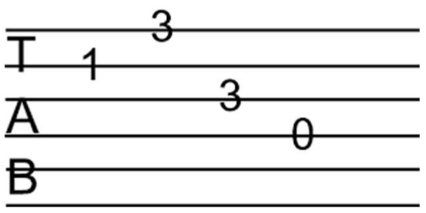


Fig.6: An Example of Tablature

III. Project Management

MDR Deliverable	Status
Identify 10 notes on one string with 90% accuracy	Complete
Verify that latency is sufficiently low so that we can record notes and rhythms within 100ms of being played	Not needed

Table 2: MDR Deliverables

We marked the first deliverable from Table 2 as Complete because we can record and identify 10 notes on one string with 100% accuracy, identify two notes played at the same time on two different strings, and identify three notes played in quick succession on one string. Not only does this indicate end to end functionality of our system, it also demonstrates the individual successes of our subsystems. We accurately collect data that we can parse to determine note frequency and the beginning of each note. During the course of our work we determined that our second deliverable was not going to be necessary as when the deliverables were written we were not sure if we were going to be doing the tablature conversion in real time. If we did the conversion in real time, the deliverable would have been relevant.

So far, Team QuickTab has been able to successfully record data from the accelerometer attached to the base of the guitar, take that data and analyze it in MATLAB using FFTs to determine the frequencies played. We have done several different types of tests to show these accomplishments. These tests included: playing notes simultaneously to prove that we would be able to detect both frequencies, playing notes in quick succession to illustrate that there is the FFTs will be able to detect each of these frequencies, and finally playing every other note on one string to show that we can determine the correct frequency for an entire string.

Even with this progress, there is still much to be done. The User Interface and TAB Compiler must be created, the sampling rate must be corrected for the Data Logging, MATLAB must be expanded to incorporate all six strings, and the vibration sensors must be attached to the guitar and embedded into the system. The User Interface and its custom PCB will be designed and implemented to work in coordination with the Raspberry Pi to give the user control over power, recording, and calibration. The TAB Compiler will take suggested frequencies and vibration sensor data to match correlating vibrations and frequencies to give a print out of what was played on a Tablature sheet. Data logging will need adjustment because the sample rate does not give enough bandwidth to be able to support the frequency range of an entire guitar. The MATLAB will need to be modified to support determining the frequency on any string without knowing which is being played beforehand. Vibration sensors will be placed onto the strings and implemented into the project to identify when a string is played to correctly determine the string a given frequency was played on.

Each group member brings a set of skills to the group; Jacob and Joe both have extensive experience with both coding and the guitar, while Lindsay and John have experience with coding and the hardware components used in the project. Together these skills allow us to create a working project, from end-to-end. We feel that our team has been working well with each other, and we understand that this is a group effort and that we should communicate what we are working on, what we are having problems on and the progress each of us has made. We have weekly meetings with our advisor to convene and talk about progress made, solutions to problems, and our goals for the week. This helps us keep on top of things, stay informed of others' progress, and refocus our efforts towards critical tasks that must be completed.

From PDR to MDR, our primary goal was getting accurate data from the accelerometer that could be analyzed using MATLAB. Each of us helped on the python code and the accelerometer setup, and the MATLAB coding was spearheaded by Jacob and Joe with the assistance of Lindsay and John. This was the most crucial part of the project and also a prerequisite into other parts, which is why we worked together mostly because we could not split up the project until this was complete.

IV. Conclusion

Currently, we have implemented hardware on our guitar that allows us to retrieve the signals played on one string from the lowest to highest fret. We can also view two notes played at the same time on different strings.

As identifying the correct notes played on a guitar is believed to be the most difficult part of our project, we believe that we are in a good position to complete our project by April. The main components that are still required in our project are a user interface with functional interrupts, a method to detect the string being played, and a print out of our tablature.

We have gotten the note identification to an accurate state for frequencies up to 500Hz. Our first objective is to overcome that drawback by matching the sample rates of the Pi and ADXL345. Next, we will identify the string being played with our vibration sensors. At that point, the biggest task is to verify that our tablature printout is accurate. Once we have a working system, the user interface will allow us to easily demo and present our finished project. At that point in the project, we will be proud to present QuickTab as the first successful 'performance to tablature' system for any guitar that UMass has ever seen.

Our project was ambitious from the start, but we are proud of the progress we have made and have plans of attack for eliminating the issues in our system. We look forward to succeeding where all others have failed.

Acknowledgment

We would like to thank Professor Kelly for his guidance and support. We would also like to thank our evaluators Professor Bardin and Professor Kundu for their invaluable feedback.

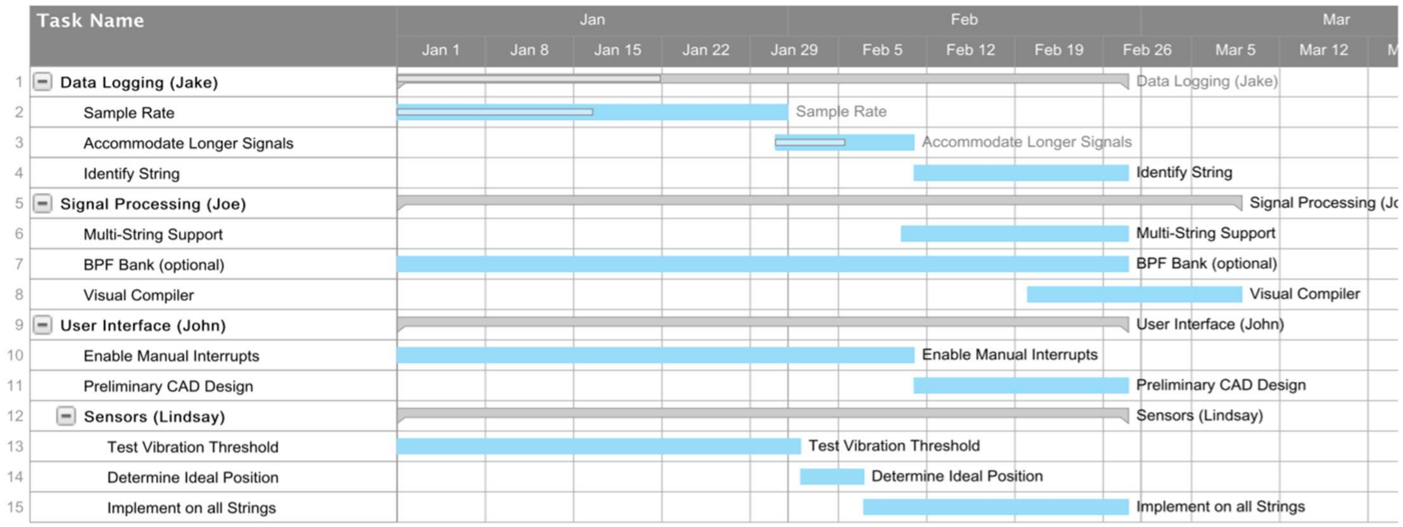


Fig.7: Gantt Chart

References

[1] G. P. 6, "Guitar pro 6 - Tablature software for guitar, bass, and other fretted instruments,". [Online]. Available: https://www.guitar-pro.com/en/index.php.

[2] "Guitar tabs with rhythm," Songsterr Tabs with Rhythm, 2017. [Online]. Available: https://www.songsterr.com/.

[3] T. Alsagoff, M. Murphy, M. Shtilman-Minkin and M. Wojick, "AutoTabber: A Frustration-Free Guitar Tabbing System", 2014.

[4] P. Ridden, "System uses lasers to detect the pitch of a guitar string before a note is played", *Newatlas.com*, 2011. [Online]. Available: http://newatlas.com/laser-system-detects-guitar-string-pitch/19278/. [Accessed: 21- Dec- 2016].

[5] Analog Devices, "3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer," in Analog Devices. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf. Accessed: Dec. 21, 2016.

[6] "SQ-MIN-200 NANO-POWER TILT AND VIBRATION SENSOR," in *SignalQuest*, 2014. [Online]. Available: https://signalquest.com/download/Tilt%20and%20Vibration%20Sensor%20SQ-MIN-200.pdf. Accessed: Dec. 22, 2016.