QuickTab
Final Project 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 for 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 hone 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 in conjunction with six flex sensors, which senses the vibrations of the guitar and strings as a note is played, giving us a cleaner signal to work with and ultimately making it possible to generate the tablature we desire.

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

***Table 1**: Project Specifications*

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)

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 attached to the bottom of a guitar, it will not be too cumbersome for the user. Power consumption is not issue because we are not using our own power supply. As a stretch goal we discussed using a power supply we designed ourselves, which would mean again taking weight considerations into account as well as power usage by the device. However, we decided to forgo this addition.

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, allow us to summarize the main functions of our system. When the user decides to create tablature, they power up 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, two files are created by the Pi and are 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.

Since MDR we have done numerous things to improve the project. First we needed to decide on which sensors to use to detect the string vibrations. Once this was done, a custom PCB was designed to interface these sensors and the UI (buttons) with the RasPi. The custom PCB included buttons and LEDs for the purpose of user control and feedback. Lastly the data recorded by the vibration sensors and the accelerometer were then transferred to a computer via email or ethernet. Our biggest challenge here was having the data from the vibration sensors and the accelerometer coordinate to produce accurate tablature. Once these two pieces of data were synergized, we were able to effectively implement the Tab Compiler which produced a visual representation of what the user had played during their recording.

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, located next to the strings, are designed to detect when a string is played. The vibration sensors are made of piezo film material. When a string is played, it hits the adjacent vibration sensor. The analog output of the sensor increases with the amount of flex it experiences. With the data received from the digital accelerometer and the vibration sensors, the information required to determine hand position is secured.

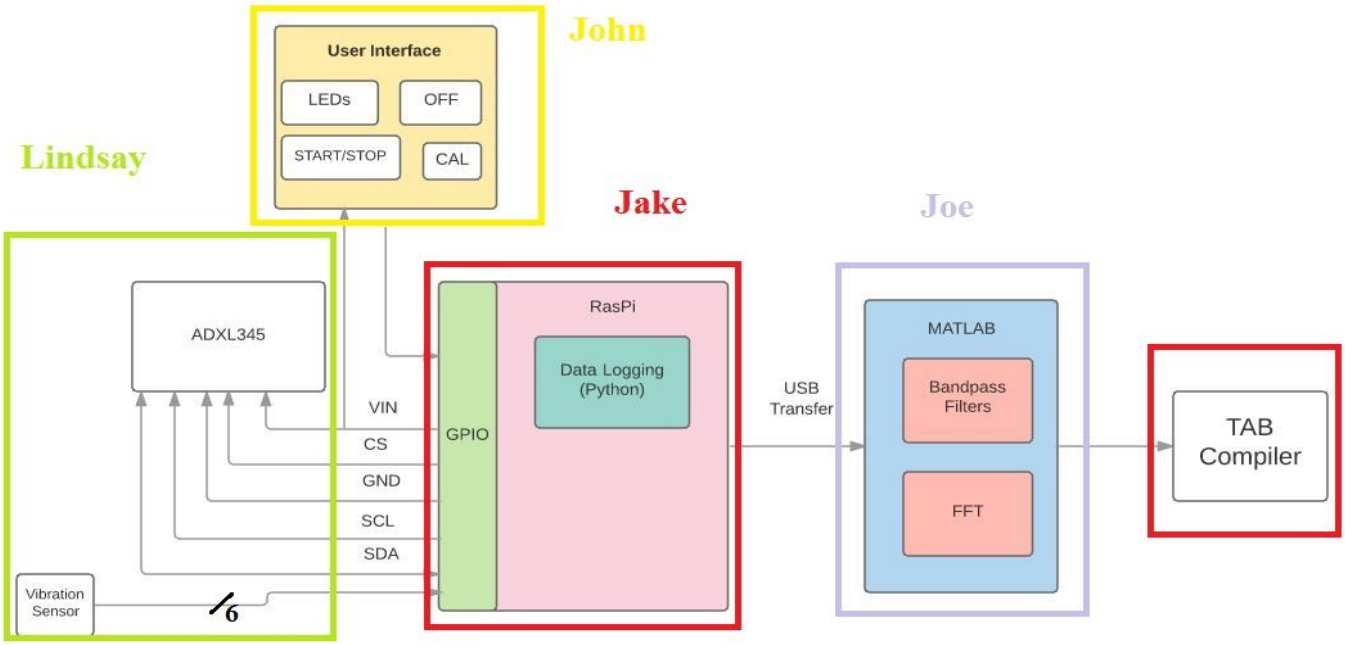

***Fig. 1**: Block Diagram*

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 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 orthogonal to the strings. Our final positioning for the ADXL345 kept it in the original place at the base of the bridge, as this was the ideal place to collect vibration data.

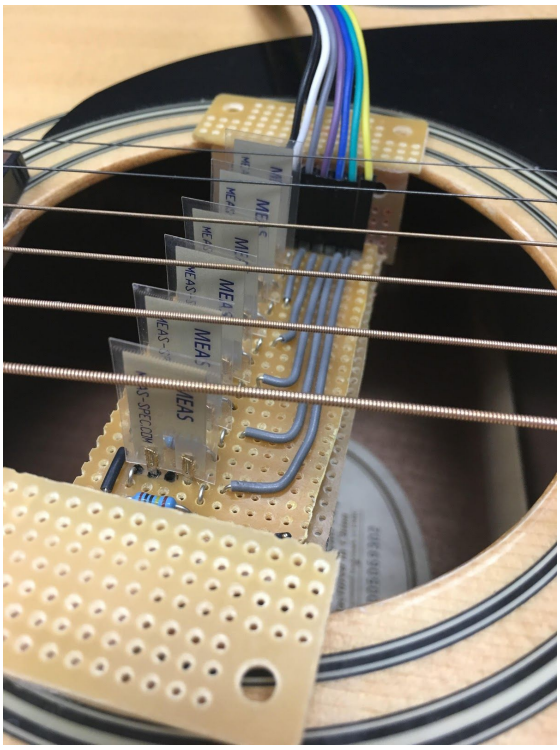

***Fig 2**: Implementation of the Vibration Sensors*

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 are placed next to each string **(Fig. 2)**. Each piezo vibration sensor is made of flexible polymer film. The resistance of the film changes as it flexes back and forth. The voltage output is then inputted into an analog-to-digital converter to allow for use of the Raspberry Pi's digital input pins. The MCP3008 ADC interacts directly with the RasPi's SPI. The sensor strip is placed at an angle relative to the strings to reduce the sound of buzzing and the resulting harmonics that are produced by the strings hitting the sensors.

Data from the sensors are logged by the RasPi. Within the python code that controls data logging, threshold values are set for each of the six sensors. These thresholds limit the amount of data produced upon a string being played. Multiple vibrations are recorded per note played and are filtered out based upon the intensity of the vibrations. Despite the threshold values in place, multiple vibrations are still recorded and placed into an array. Each entry in the array contains the number of the vibration sensor (numbered one through six) and the timestamp corresponding with when the vibration occurred. In Matlab, further data filtering occurs and duplicate sensor data is eliminated based on the time differences between data entries. If multiple entries are recorded from the same string within 500ms, then all but one entry is eliminated. The remaining timestamps are used to run FFTs at specific times.

The final implementation of the sensors **(Fig. 2)** was successful. The timing of each note was recorded accurately and the string used can be identified. The tray designed to house the sensors is made of a lightweight prototyping board. The tray is attached to the guitar using velcro. The sensor tray can be easily removed and replaced. The sensor sub-system was successfully incorporated into the QuickTab design.

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 visual feedback. A custom PCB designed in EagleCAD was used to implement this portion of the project, consisting of a series of buttons and the A/D conversion of the vibration sensors. The final version of the PCB can be seen in **Fig. 3**. On the right side of **Fig. 3**, you can see the A/D converter used for the vibration sensors..

The PCB contains three buttons; record, calibrate and OFF. It also contains three LEDs pertaining to each of these buttons. The red LED, next to the record button, is lit when the user is currently recording. The green button, next to the OFF button, is lit while recording and when the RasPi is ready for to receive additional inputs. **Fig. 3** shows the PCB as well as the three buttons and their corresponding LEDs. You can see where the PCB was placed in **Fig. 4**, just to the side of the strings. This allows the connections to the vibration sensors to be short while still being out of the way of the user. The user interface is managed by the python script running on the RasPi. Interrupts are used for each of the buttons.

One of our goals since MDR was to eliminate the need for a monitor to display the RasPi. To achieve this, additions were made to the RasPi startup code that would launch the python script needed for the UI and recording/sending the data. Any functionality the user would need was available on the UI and if troubleshooting was needed, the monitor was still available during demo day.

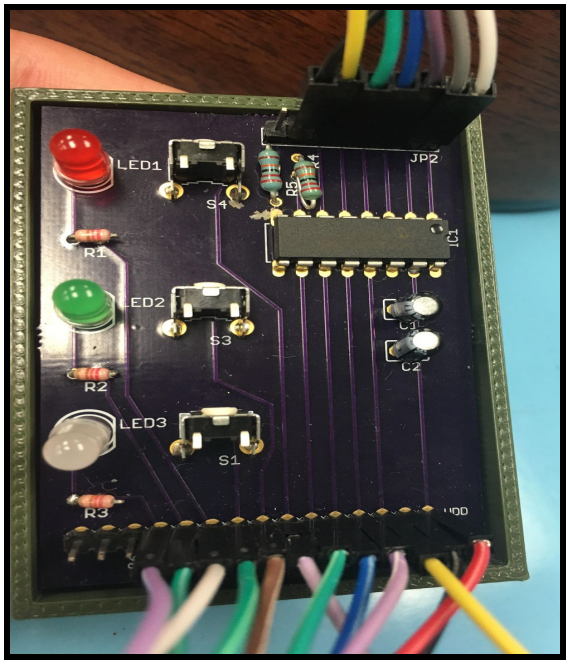

***Fig 3**: Final Printed Circuit Board Design*

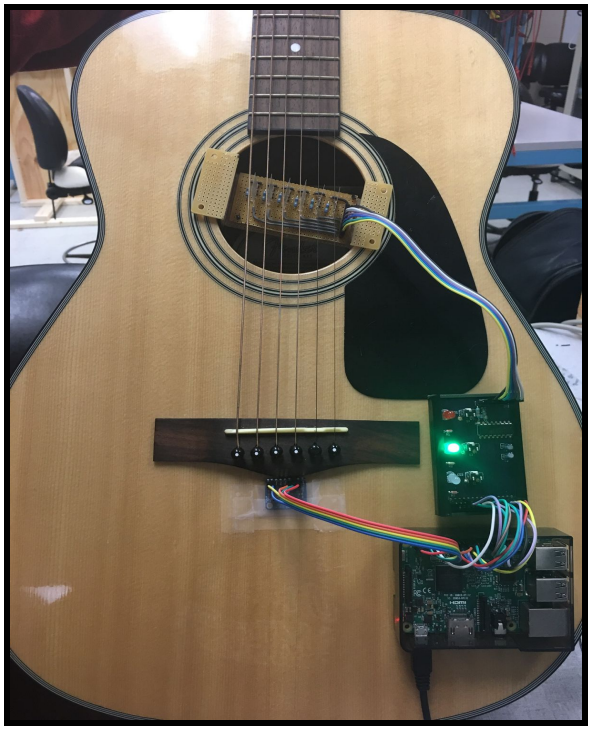

***Fig 4**: Implementation of the User Interface*

Transmission of data was another portion that indirectly tied into the UI. Upon finishing a recording, the data from both the vibration sensors and the accelerometer would be written into .py files which would then be emailed or transferred via USB.

As seen in **Fig. 4**, the UI position on the guitar has changed since the MDR report. At that time we were planning on placing

The UI of this project was a success. The user was capable of having all the functions needed to accurately record tablature all on a small PCB. The vibration sensors, PCB, and RasPi were all successfully attached to the guitar and secured in a neat fashion. They were out of the way, provided minimal modifications to the guitar, due to the velcro that held them on, and were very light weight. This accomplished all the goals of our project while still succeeding in providing the user with functionality.

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 built upon the fundamentals that we learned in Data Structures & Analysis as well as any other basic programming courses we took. Because our code utilizes Python, we familiarized 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 manually configured our GPIO to operate under the parameters we gave it and ensured that only digital signals were 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 required 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 could 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 verified 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 brought the first two components together. In order to verify that both sensors are aligned, we viewed their data side by side and ensure that the spikes in both sets of data occur at the same time. These three tests together ensured the accuracy of our data logging both individually and collectively.

Since MDR, we were able to satisfy all three of these tests by making a few functional improvements. First, we increased the baud rate of the Raspberry Pi from 100KHz to 1MHz. This enabled us to retrieve all of the samples taken by the accelerometer sampling at 3200Hz, as well as retrieve the information sent from the vibration sensors. In order to verify that the accelerometer was giving us useful data that did not have terribly skewed harmonics, we adjusted our sampling method from including all three axes to simply recording from the z-axis, the one orthogonal to the strings. This allowed us to stop taking the magnitude of all three axes and removed the squared component of our signal. Also, in order to send the data to be parsed by the Matlab code, we included code that sent the files to an e-mail address so that the data was available from anywhere.

E. Signal Processing

After the data has been recorded by the Raspberry Pi the raw ADXL data and the data from the vibration sensors is off-loaded to a PC where it is processed via our MATLAB code and the correct frequency is determined. **Fig. 5** shows data that is sent from the data logging stage to be utilized by the MATLAB code. First the vibration data is parsed through and we attempt to eliminate any duplicates from the vibration data by checking to see if we have multiple of the same string appear in a row with a timestamp that differs by half a second or less, because we remove any entry from the vibration data that is closer than half a second this limits the speed at which you can play to two notes per second. After the duplicates have been removed from the vibration sensor data we iterate through the array of strings detected to find each note played. For each event that is detected we first run the data through a bandpass filter based on which string was detected, we do this to remove any extraneous noise and to eliminate any frequencies that cannot appear on the string being played. 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.

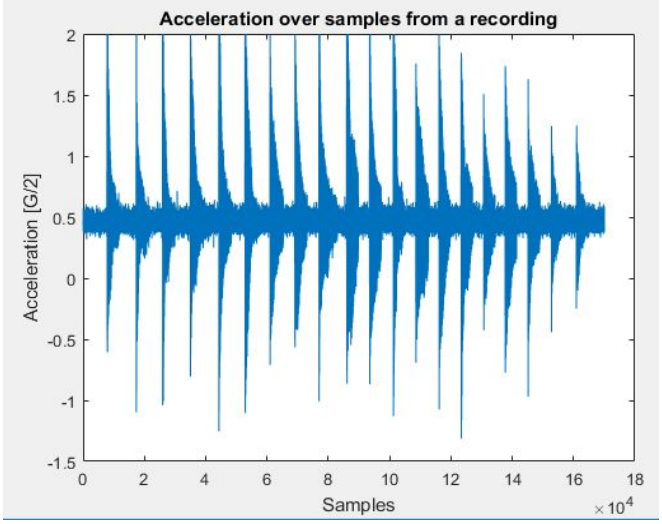

***Fig. 5**: Raw ADXL345 Output of A3 Being Played*

Next we look at the timestamp for the event which is detected, we take the time in seconds and multiply it by the sampling frequency which gives us the exact sample that the note starts on. Next we perform an FFT on the data at that specific point that is 500 points wide. **Fig. 6** shows the output after an FFT is taken.

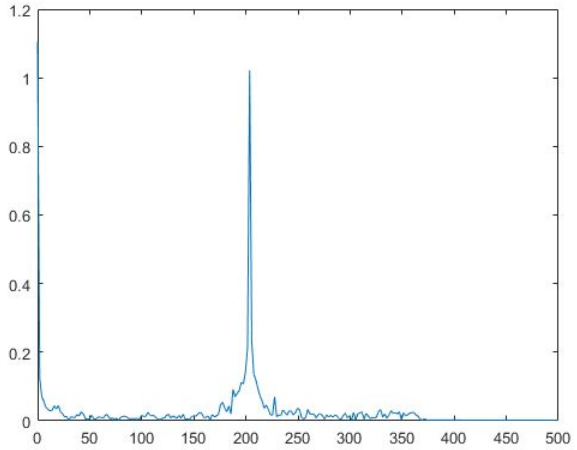

***Fig. 6**: FFT Showing Note A3 (220Hz)*

After the 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 every event that is detected via our vibration sensors. **Fig. 7** 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. 7**: Printout for the Above Signal, Displaying Frequency*

Due to the fact that determining accurate fret information requires the knowledge of the base frequency of the string that the note was played on, our system has a calibrate function that allows you to change the stored values of the base frequencies within the code. To do this you simply read in calibration files from the ADXL and the same process from above is performed except instead of printing out the tablature at the end, the new values of the base string frequencies are put into the frequency arrays.

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)** is 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 is 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.

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

Tablature, such as the example depicted in **Fig. 8**, 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 are filled with n values in the order that they occurred in. The values will be printed as tablature. The TAB Compiler has been completely implemented, as 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.

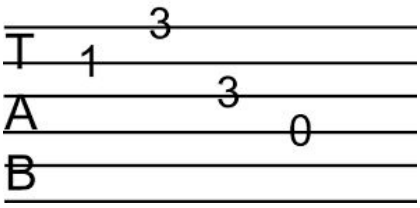

***Fig.8**: An Example of Tablature*

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.

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 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.

After MDR, we still had much to do. The User Interface and TAB Compiler had to be created, the sampling rate had to be corrected for the Data Logging, MATLAB had to be expanded

to incorporate all six strings, and the vibration sensors had to be attached to the guitar and embedded into the system. The User Interface and its custom PCB needed to 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 needed to 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 needed adjustment because the sample rate did not give enough bandwidth to be able to support the frequency range of an entire guitar. The MATLAB needed to be modified to support determining the frequency on any string without knowing which is being played beforehand. Vibration sensors had to be placed onto the strings and implemented into the project to identify when a string was played to correctly determine the string a given frequency was played on.

Every single item on our "to do" list was completely checked off. In the end, we were able to generate a fully working project that generated tablature with ninety percent accuracy under reasonable circumstances. The vibration sensors functioned perfectly in conjunction with the accelerometer to collect the specific data required by our code to determine the strummed frequencies; our User Interface functioned ideally for the purposes with which it was designed, providing valuable feedback and performance; the data was collected accurately and consistently and transferred to our code with efficiency; the logic of our code was sufficient to determine the frequencies and times at which notes were played; and finally, the tab compiler visually displayed all of the data we had collected with precision and grace.

Each group member brought a set of skills to the group; Jake 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 allowed us to create a working project, from end-to-end. We feel that our team has worked well with each other as we understood that this was a group effort that required communication of what we were working on, what we were having problems on and the progress each of us had made. We had weekly meetings with our advisor to convene and speak about progress made, solutions to problems, and our goals for the week. This helped us keep on top of things, stay informed of each other's progress, and refocus our efforts towards critical tasks that had to 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 for other parts, which is why we worked together as we could not split up the project until this was complete.

Since MDR, our group made significant progress with Project Quicktab. Up to MDR our project was focused mainly on getting results from the accelerometer. Because of this our group was very concentrated on successfully completing and progressing this goal, which led us to have less individual goals than what our evaluators would have preferred. After MDR, because we were able to get results, we were able to each have clear and achievable goals for each subsection of our project. As shown in the block diagram, John was in charge of the UI, Lindsay was in charge of the vibration sensors and the accelerometer, Jake was in charge of the RasPi python scripts/data logging, and Joseph was in charge of the MATLAB/signal processing portion of the project. Each of us were able to work on our parts efficiently, and therefore there were no bottlenecks for our project. From time to time there would be problems with different sections, but when these issues did arise we were able to effectively communicate and problem solve together. Each of the group members has contributed to this successful project. We are proud to say that we are the first group who has attempted a project like this that has had it working consistently.

Another success of our project was its cost. Our project was relatively inexpensive compared to its alternatives such as Guitar pro[1]. As you can see in **Table 3**, the majority of our cost is from the RasPi and the vibration sensors.

Cost

Development		Production	
Part	Price	Part	Price
Raspberry Pi	35	Raspberry Pi	35
Accelerometer	18	Accelerometer	18
Vibration Sensors(x6)	45	Vibration Sensors(x6)	18
PCB (All components)	35	PCB (All components)	13
Total	133	Total	84

***Table 3**: Development and Production Cost*

IV. Conclusion

At the end of Senior Design Project, Team QuickTab successfully created a prototype for automated tablature generation that operated at a ninety percent success rate. Each subsystem operated with individual success and met the goals and requirements that we had pursued from the beginning.

For our piece by piece breakdown, we shall look at each subsystem individually. Spearheading our sensors, Lindsay accomplished the objective of using the ADXL345 to pick up the vibrations in the guitar from which we determined the frequencies. She also met our goal of determining which string was played and at what time by using flex sensors resting on

each string.

John accomplished an implementation of the User Interface that was intuitive, entirely accurate, and well-designed. In addition, John also introduced additional functionality to our system by setting up the RasPi to run without the need for a monitor by running the code as soon as it was powered.

Joe fully implemented the Matlab code with all of the logic and Fourier transforms necessary for the code to interpret the vibrational data with sufficient accuracy. Within his Matlab code was also additional code segments for handling guitar calibration so that the code could always accurately determine the fret played.

Jake ensured that the data collected by the Raspberry Pi was collected from the correct locations of the ADXL345 and stored in a reliable way so as to be correctly interpreted by the Matlab code. Additionally, he ensured that the final output of our project was precise, according to the conventions of traditional tablature.

Acknowledgment

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

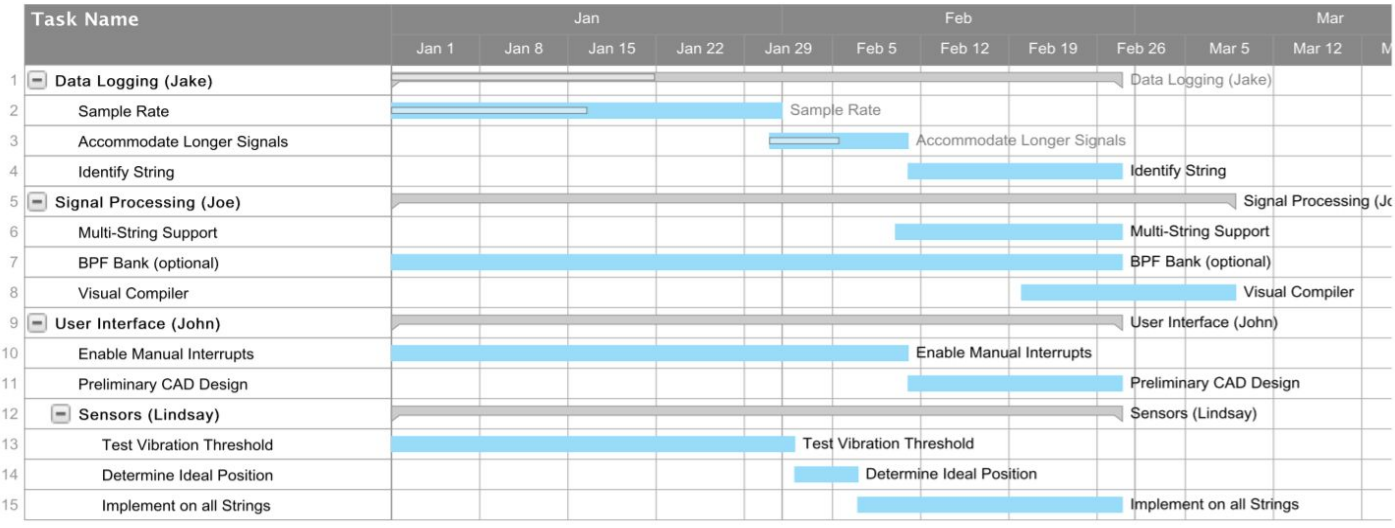

Fig.9: Gantt Chart

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.