Guitar Tuner - Signals and Systems Final Project

James Jang, Christopher Wallace, Filippos Lymperopoulos

May 6, 2015

Abstract

Our goal for our Signals and Systems final project was to design and build a portable device that would tell you if your guitar is in tune or not. In order to make it portable we knew we had to use a small device-microcontroller to facilitate the functions necessary.

1 Introduction

In the beginning of the project, we were considering constructing the system in the analog world. Using a bank of bandpass filters combined with an oscillator circuit that spans a range of frequencies proved to defeat the purpose of our initial goal of making a portable device. Hence, to accomplish our task we proceeded by searching for a microcontroller.

2 Microcontroller Specification and Selection

2.1 Beaglebone

Having concluded on the digital world implementation of our system, we were debating on the merits of using either a Beaglebone Black or an Arduino Uno. The Beaglebone is an interesting device that can fit in one's pocket, while at the same time being able to run on a full Linux system. It is relatively powerful, and is able to read analog, as well as digital inputs.

It looked as if it would be the perfect choice. We could even run Python on this device and use libraries which we were accustomed to, including numpy, scipy and thinkdsp. Its only downfall was its sampling rate. We could only sample at a rate of around 800Hz, which was barely high enough for us to record the frequencies a guitar generates.

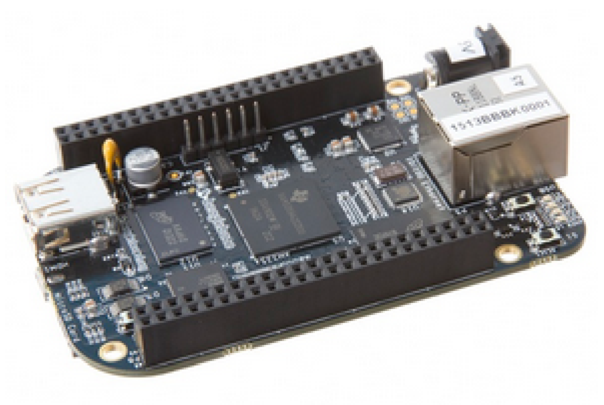

Figure 1: Beaglebone Black

2.2 Arduino UNO

On the other hand, the Arduino UNO microcontroller is a much lower level device than the Beaglebone, closer to a true microcontroller. Arduino runs C, and has several digital and analog inputs. It has an analog sampling rate of around 8kHz, which would be high enough for the frequencies we would be dealing with. As it turned out, its major drawback is its power. It is not very powerful, and running advanced computations on a large set of numbers may prove really difficult for the Arduino to handle.

Figure 2: Arduino UNO

3 The System

As indicated in the schematic below, our system is relatively simple. It is composed of 5 main subsystems. On the left side of the schematic, the beaglebone and the speaker perform the receiving and computation of the audio signal of the system. It is important to point out that nine wires are the outputs from the Beaglebone board, addressing the two banks of LED's. The next subsystems, include the amplifier, with an internal resistance $Rin = 3.3k\Omega$, an anti-aliasing low pass filter with a cutoff frequency of 500Hz. Finally, two banks of LED's; the first LED system includes 6 LED's each serving as a designation for the note a person is playing, and the 3-LED bank designates whether a note is sharp, flat or tuned.

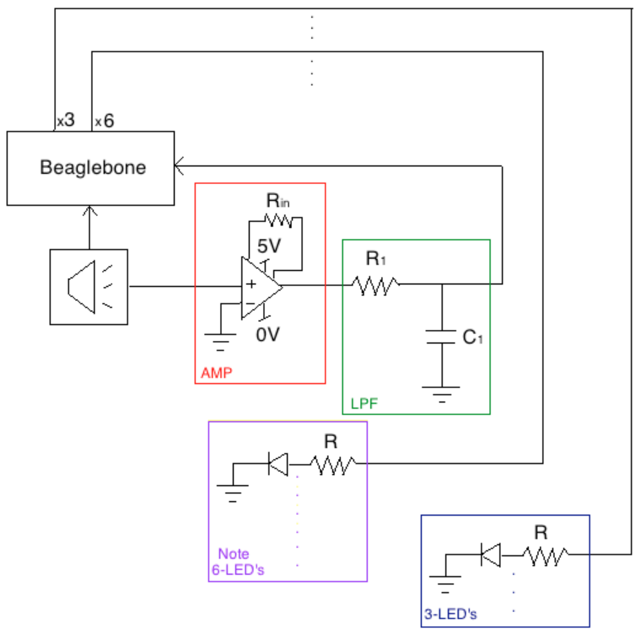

Figure 3: The System

In order to process the input signal, we had to turn the noise of the guitar into an analog signal. To perform this, we would back-drive a speaker. This would give us an analog signal of very low voltage. The next step would be to amplify this voltage into the range appropriate for our microcontroller, and then put this signal into an anti-aliasing low pass filter. This low pass filter will reduce aliasing, because all of the high frequency components that would show up as aliasing will be decreased in magnitude. Since we know we are tuning a guitar we picked a value of 500 Hz as the cutoff frequency of our filter, given the fact that guitar notes range from 82 Hz to 330 Hz. It should finally be pointed out that by by-passing our system with a 1uF capacitor, we were able to further smooth out the signal examined.

4 Attempts

4.1 First Attempt

We first looked into the Arduino because all of us had experience dealing with Arduino C and the nature of the controller. We were able to write a function to perform auto-correlation on the analog signal and extract the frequency of the note that we created with an analog discovery, however this method was often inconsistent.

4.2 Second Attempt

We knew that the Beaglebone Black would have a much slower sampling rate than the Arduino. In order to attempt to compensate for that we would take a longer period of samples and use the increased computational power of the Beaglebone to process these longer signals.

Since our signals were going to be noisy, we decided we would ”clean” them by getting the autocorrelation of the signal and then taking the FFT of that autocorrelation function. The result, as given in Figure 4, is a cleaner graph. The graph below shows the spectrum of a guitar strum prior and post running autocorrelation, respectively.

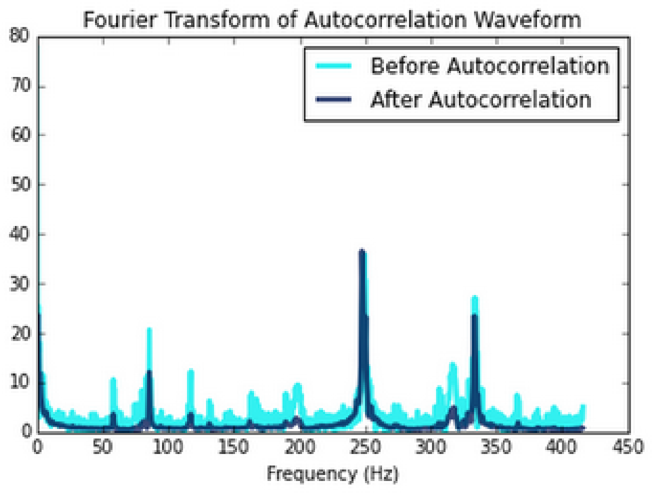

Figure 4: Autocorrelation on Input Signal

As the graph above indicates, the sound recorded from the string B, is approximately 250Hz. The expected ”correct” frequency is 247Hz, indicating that our system was able to detect the note and light up the correct LED’s both for the note and sharpness of the note.

5 Conclusion

Implementing the second attempt to the problem, we were able to construct a system that would record a guitar string and detect what note that string was. The LED being lit in the first 6-LED bank would correspond to the note one was trying to play, while the LED getting lit in the second bank would indicate if you were sharp or flat on that note.

6 Difficulties

Our difficulties mainly stemmed from fiddling around with hardware components to receive sound signals and performing the right transformations to tune the note. As aforementioned, instead of a microphone, we used a speaker and back-drove it to pick up sound. In order to pick up the sound, we used an amplifier, yet the output of this system didn't work as we expected. We decided to switch the operational amplifier to an amplifier and we were able to extract the output.

Another main dilemma we had through the progress of this project was actually selecting with which microcontroller to go. The trade-offs involved with both, yielded the selection process to be very time consuming.

We noticed at certain parts of the experimentation that the Arduino was able to find the correct frequency for some notes, however, it was not consistent. More precisely, it could not find the correct frequency for some of the low notes. We tried varying the sample size but it didn't help the inconsistency. We decided to move on to the Beaglebone after working with the Arduino failed. It is self-explanatory to point out that the use of a Beaglebone was much easier due to the python libraries we were able to use.

We encountered an issue with our Beaglebone board, while experimenting with the recording of a specific note. After this issue occurred, our controller wasn't able to work and we were left with the data saved, as previously explained in the report. You can find our code to replicate this project in our github repo: https://github.com/cwallac/SigSysFinalProject

We would like to thank the entirety of the teaching team and the teaching assistants-NINJAS for all their help and support throughout this final project.