ECE 153A: Chromatic Tuner

Luke Lewis | Eduardo Olmos

Purpose

The purpose of this lab was to combine everything that we have learned thus far in this course by building upon our previous labs and lectures in order to create a fully functional chromatic tuner. This project involved the combined use of timers, interrupts, GPIOs and debouncing, and QP-nano in order to create a tuner that can operate as accurately as possible in a real time setting. Optimizations to provided functions (such as FFT) were also necessary in order to meet real-time constraints that are expected of modern embedded systems. With all of this combined, we also made use of an LCD screen to create a responsive user interface that is both pragmatic and effective with the goal of having it easy to use for the general public.

Design

Overview

We essentially wanted the tuner to always be listening, which meant that our microphone was constantly reading input while the tuner is on. We also wanted our whole UI to fit completely in one screen. The user is able to initiate input in two ways: changing the octave and changing the A4 tuning. These then call different functions which update our sample readings and decimation (when the octave changes), as well as update the frequency ranges of the notes (when the A4 tuning changes). With these things in mind, we just decided to have an on state for the machine which is always taking input from the microphone and calling our FFT function, but is also ready to quickly change the values stated above in order to get accurate readings.

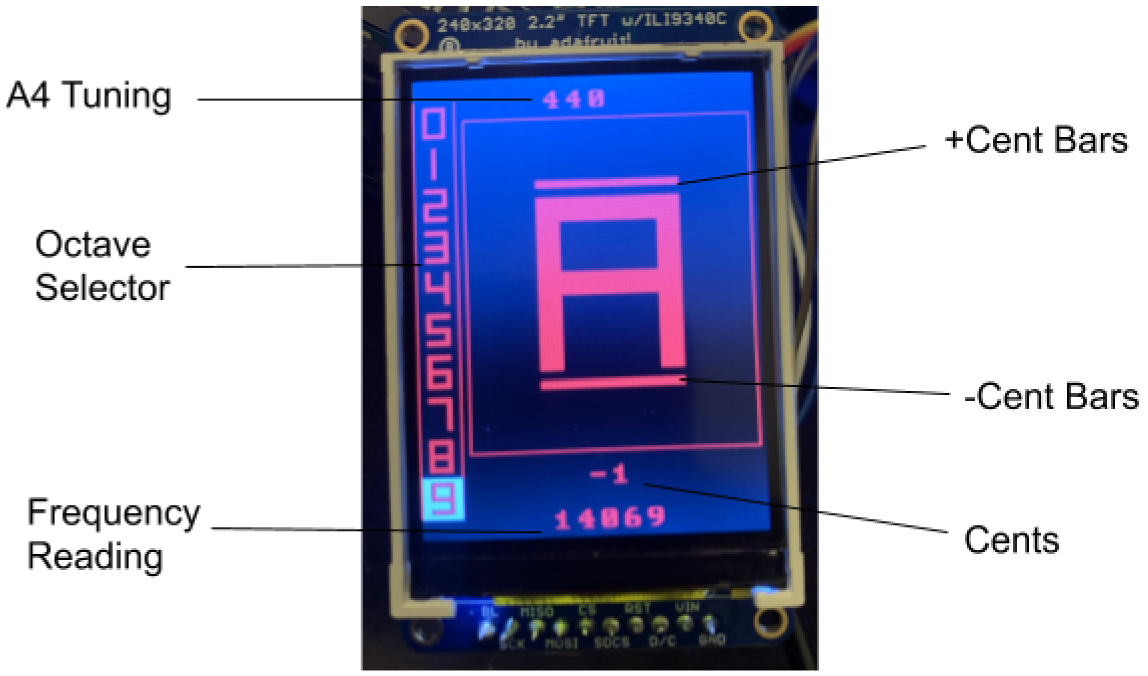
Changing Sample Frequency and Decimation

We decided to have 3 different modes (sample amounts and decimation) for these octave ranges.

	Samples	Decimation
Mode 1 (Octaves 0-2)	1024	2
Mode 2 (Octaves 3-7)	4096	4
Mode 3 (Octaves 8-9)	1024	1

User Interface

Our tuner has two user input capabilities: changing the octave they are testing, and changing the tune of A4. With this functionality in mind, we still wanted to make our interface as user friendly and intuitive as possible--all within one screen. Twists of the rotary change the selection of the octave, with the selected one being highlighted. Presses of the up and down GPIO buttons change the A4 tuning accordingly. There is no need for any extra pressing such as scrolling through a menu and then clicking a selection in order to start clicking/twisting again to change a value.



Cent Bars
The cent bars are used to let the user know how far away they are from the nearest note. The further they deviate from a note the more bars will show. Bottom bars show deviations from -50 to 0 cents, while the top bars show deviations from 0 to +50. If they are within 10 cents of the note, a bar above and below their note will show.

Out of Range
If a user is playing a note that is out of the selected octave's range, the LCD will display "Out of range", instead of a note.

Testing

iPhone Apps

Core testing was done by playing set sine frequencies using apple store apps, which played these frequencies out of the iPhone's speaker. We had originally used the free app "Sonic", but we found out that it wasn't playing frequencies very accurately, and were directed to the paid app "ClearTune", which was supposedly a better source of sine frequency audio. Results when using these apps were quite good, with errors generally staying under 10 cents.

Guitars

We also tried testing our tuner with both an electric guitar playing through an amp and an acoustic guitar. Both guitars were first tuned using a clip on D'Addario tuner.

We had fairly accurate readings with the acoustic guitar, in that it showed the correct note at the octave we were playing, however our cent reading was constantly jumping between positive and negative values. We believe that this can be improved by using a balancing/averaging algorithm which returns the average of a set amount of readings (due to time constraints, we were not able to implement this). With the electric guitar playing via amplifier, we weren't able to get accurate readings for the same notes played on the acoustic guitar. We suspect that this was due to the ambient noise coming from the switched on amplifier.

We were curious to see why the D'Addario tuner was able to tune both the electric and acoustic guitar, while ours was only reliable for the acoustic. After reading on the D'Addario tuner, we found out that instead of a microphone, it uses vibration detection through the guitar's headstock (hence why it needs to be clipped on) to determine the frequency. This was quite interesting to learn, as we think it is a very clever workaround to cut out unwanted ambient noise, especially when tuning in a loud setting, like a concert hall.

Performance Monitor Results

	Mode 1	Mode 2	Mode 3
FFT (Avg over 100)	16ms	37ms	37ms
fillRect - Idle (Avg over 100)	31ms		
fillRect - 440Hz (Avg over 100)	37 - 42ms		

The expensive operations in our Tuner were the calculation of the FFT and the UI updates. We optimized the FFT code in Lab 3A to simulate real-time responsiveness. The amount of decimated samples used scales the speed of the FFT the most. This is why there is a ~50% reduction when in Mode 1 than in Mode 2 or 3. The updates to the UI were measured at idle and when reading a 440Hz tone for each frequency reading (FFT calculation). We optimized our UI code to try to only update what is necessary. For example if we are at 0 cents and the +10 and -10 bars are lit up, if we then read 15 cents: the -10 bar is removed and a +20 bar is lit up. However due to limitations in our text memory segment (the size of our code) we redraw the +10 bar again. This is an area of improvement that can be brought up with more efficient code or more memory. When not at idle, the frequency readings may not be so stable and the UI will fluctuate more causing a higher time spent running that function as a total.

Conclusion

Plenty of time went into the creation of this tuner, and we were glad to see the accumulation of all the knowledge we have learned in this course thus far in creating a real time embedded system that has a practical application in the real world.

As is expected when developing an embedded system, a myriad of issues arose during the development of this tuner. For example, limitations on space became very apparent, which forced us to rewrite certain parts of our codebase. Despite the headaches that it caused, these limitations ultimately forced us to make the same functionality happen in a smaller amount of lines, which is very valuable at scale. Plenty more problems came up--some totally out of the blue-- and acknowledging (or at least trying to acknowledge) where these problems came from, along with fixing them, ended with us feeling much more accomplished and knowledgeable.

The expectations that we had to meet for this lab were tough, but definitely realistic. Along with the expected expectations of having fast performance and accurate results, a surprisingly big takeaway was seeing how important a good UI experience is when building embedded systems: even if something does its job well, nobody will buy it if the interface confuses them.

To conclude, through the countless hours put into all the labs, homeworks, and readings that were done in order to create this tuner, we have left the class feeling like better developers, having accomplished the task of creating something one might see in a real world setting. Through this project, and the culmination of the class as a whole, plenty of lessons and better practices were learned, and if opportunity arises in the future, we feel much more prepared to create more robust and functional real time systems.