Lab 3b Handout

The Chromatic Tuner: Undergraduate Final Project

Lab Overview

A chromatic tuner is a tool used by musicians to properly tune their instruments. The musician begins to play the instrument at a frequency, for example, 440 Hz is A4, or the note A in the 4th octave. The chromatic tuner responds by displaying the closest note and octave as well as how far the tone of the instrument is off from the note.

1 Introduction

Before beginning the final project, all students must have a functional Vivado RTL Design of their embedded system from Lab 3A. By now you have accumulated many different .c and .h files and have a reasonably efficient Fast Fourier Transform (FFT) implementation. Now you need to make your early prototype of the chromatic tuner into a finished embedded system. Create organized C code by incorporating the FFT code into the QP-Nano based HFSM GUI built in previous labs. Lab 2B provides a start for the structure of The Chromatic Tuner design. You will need the key functionality of the microphone code, rotary encoder and LCD code functions merged with the HFSM design of The Chromatic Tuner. Create a combined list of signals and states for your QP Nano GUI controller. Incorporate the interrupt based control logic used in Lab 2A to use the Rotary encoder as a menu input for the chromatic tuner. This should already be complete from Lab 2B. Test to make sure it is working flawlessly. The goal in Lab3B is to build a polished chromatic tuner. **You will be graded on the overall appearance, fluidity, usability and functionality of your finished design. Nicer fonts, color choice, algorithm and interface stability, accuracy and range will all contribute to the final grade.** The write-up needs to describe key steps and decisions you made to achieve your design. Although you can assemble default code to make a working project, consider the practical usability of the design – and whether you could sell it...

2 Tuner Design Features

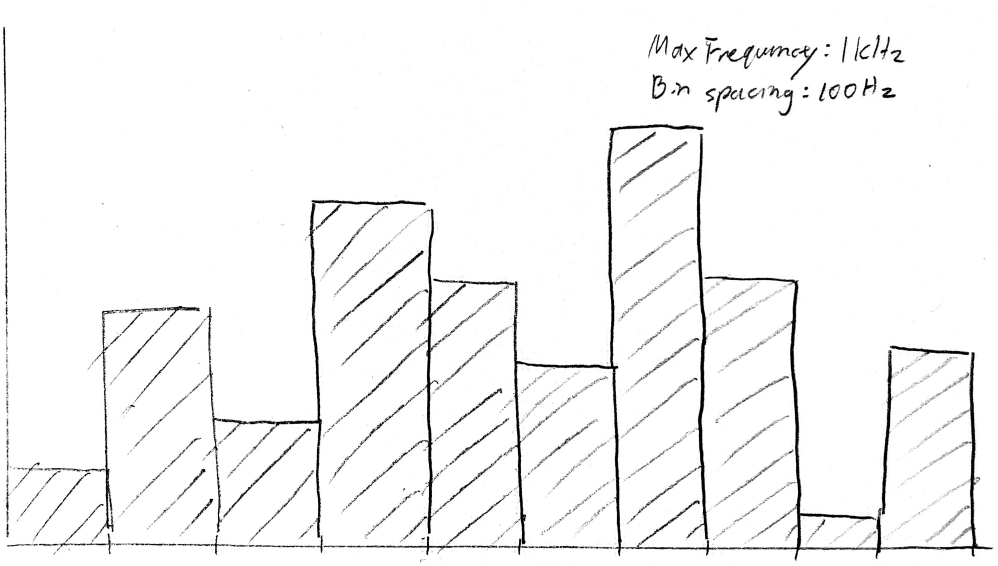
1. The display should show the nearest note and its octave (preferably in a big, easy to read font) and the frequency being played.

2. Also display the error estimate of input tone in geometric cents. Generally, equal-tempered notes are spaced at geometric spacings of $2^{1/12}$ with 100 geometric divisions between each successive note (thus called 'cents'). This error must be displayed as some form of horizontal graph, with the note in the center and a bar to the right if the played frequency is too high, and to the left if it is too low. Most tuners annotate the graph with colors depending on the tuning – green if very close to) cents error, other colors for flat (below the note) or sharp (above the note) so that the tuning can be seen at a glance. (One could color the note display as well...)

3. The tuner needs to accommodate alternative base tuning (e.g. A != 440 Hz) to allow for relative tuning with fixed instruments such as old church organs, harps or older pianos that cannot be tuned to A=440. Instead, A4 should be variable from 420 Hz to 460 Hz in 1 Hz intervals. For this feature, tie the value of A4 to your rotary encoder. As soon as the encoder is turned, display the current frequency value for A4 and increment/decrement it by 1Hz per rotation click. Once the encoder has stopped moving, remove the overlay and update your internal tuning based on the new value (I.e. if A4 is switched to 430 Hz, every note value should shift to the new base: A5 would be 860, not 880 Hz etc.)

4. You are required to make one or more test modes to help incrementally code and test your design. We recommend implementing a graphical output for the FFT. (we will be asking what you did in the final demo – note these test features need not be real-time – but should not have gui glitches, they are for your own testing)

   FFT display Create a function mode that shows a graph of the FFT. The display should also show what the highest frequency and the bin spacing. This way you can get an idea of what the received sound inputs look like in frequency space. This function is for early debugging and test of your signal sources. It probably makes sense to restrict the bins to the range you care about, including sensible scaling (e.g. log or linear) and amplitude scaling, since your alternative is to dump data to the console.



   Octave selection Create a menu that locks the signal processing to a given range (e.g. C2-C3 or A5-C7 or whatever ranges are relevant to your processing). Once you've selected the range, the tuner will work as described previously, however, the value (and error displayed) are derived by the signal processing (averaging/filtering) for that range, and will likely be noisy or aliased for other frequencies.

   Other You may also create any other test functions that you believe will be helpful.

5. You may need a menu that allows you to select between the chromatic tuner functionality and the test functions or modes.

   Control You can switch between the menus as you like, and make use of inputs from the rotary encoder twist and push, as well as button inputs to navigate different modes. You can also be as creative as you like with the background and display of the menus – the different menus can be displayed on the same screen or you can change your screen

to only show details relevant to the current mode. The aim of your final design is fluid operation and relatively rapid determination of the input note, selecting the signal processing you need if it is dynamic and providing the most accurate result you can. Effectively, the design should auto-range over the several octaves of possible input. Pragmatically, if it doesn't work on low notes, or if it is always slow to respond, you won't sell many. On the other hand, if it responds quickly, but is a bit off – but improves the accuracy successively, it is probably quite usable.

Encoder The rotary encoder should function in real time. We already solved a part of this problem in Lab3A by improving the FFT response time.

Display The display is often the slowest part of the system – taking tens of mS to paint large sections. It is possible to improve the display performance is several ways: 1. increase the SPI clock rate (as long as your displays operation is stable). 2. build and use more of the display command features (see the Rinky-Dink or Adafruit libraries to see how). 3. Limit display update call rates: If the function is going to take 8mS, it is dumb to allow or start a new display function in 2mS, before the old one is done. Humans can easily see 1 mS glitch that shouldn't be there because of vision persistence, however, by the same token, if you only make clean updates every 20mS, you are still updating at 50Hz, more than twice the rate movies are played. This can be done by checking the elapsed time between calls to the screen... and ignoring ones that are too quick...

3 Recap on Octaves, Improving FFT Accuracy and Stability

The frequencies detectable by the microphone range from tens of hertz to $> 15kHz$. Octaves are used to divide this wide range of frequencies into smaller groups of frequencies on a logarithmic scale that are labeled 0-9. Each group contains 12 notes:

$$C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B$$

The frequency corresponding to a note is described by the following formula:

$$f = 440\ Hz * 2^{\frac{(n-9)}{12}+k-4}$$

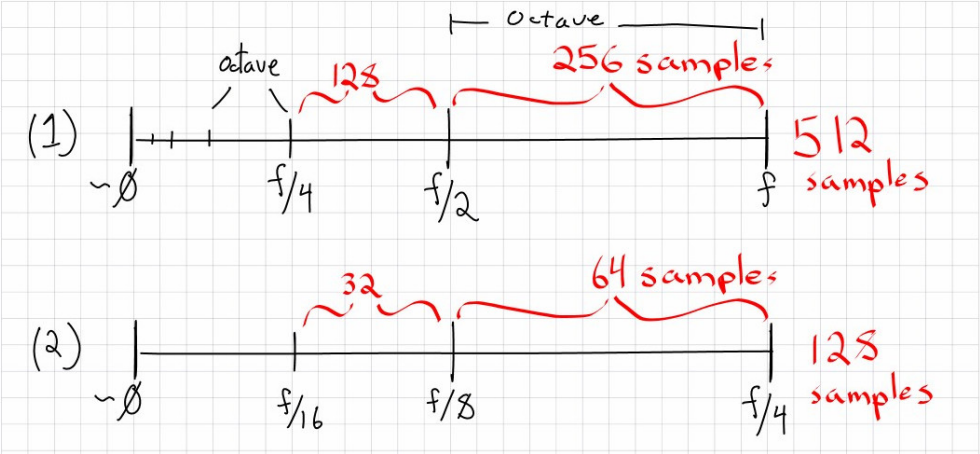
where n is the number corresponding to a note (C is 0, A is 9, A# is 10, etc.) and k is the number corresponding to an octave. So $A4$ is 440 Hz, $A1$ is 55 Hz, $C4$ is 261.626 Hz. The note below $C4$ is $B3$, which is 246.942 Hz. $B4$ is the highest note in octave 4, at 493.882 Hz. $C5$ starts octave 5 at 523.251 Hz.

The use of octave ranges is potentially useful when calculating the FFT and can help decrease the time to compute the FFT by allowing a smaller, lower resolution FFT to be used. You can sample continuously for up to 4096 samples, but owing to software invocation of the sample process, you cannot easily make multiple calls to the sampler to get longer sample ranges as you cannot easily predict the delay between invocations.

3.1 Fast Fourier Transform

The Fast Fourier transform places recorded samples into (linearly spaced) frequency bins. The FFT is calculated up to the Nyquist frequency (half the sampling frequency) which here is 24.4140625KHz. Because of the exponential scale of notes, the frequency resolution of notes placed in the upper half of the range is greater than those placed in the lower half of the FFT range. This is shown in the figure below. Looking at Fig. 3.1:plot(1) If you are analyzing the raw samples from the microphone, the upper half would contain frequencies ranging from 12 KHz to 24 KHz. This means that in trying to detect

a lower frequency, like 100Hz, the bin spacing makes high frequency resolution impractical. Furthermore, the microphone response begins to fall off at 15kHz and there is very little (if any) music which is fundamentally tuned in this range, so we are making little use of the most accurate part of the FFT in the default setup.



A simple way to fix the effective resolution, and potentially increase the speed of the FFT is averaging or decimation. This is shown above in Fig. 3.1:plot(2), which computes the FFT using 128 samples. The 128 samples could either be a decimated read from the buffer, where we take only every 4th value, or the samples could be averaged values from the original 512 samples. 4-way averaging the samples makes an effective 128 samples at $48KHz/4 = 12KHz$. Unlike decimation, averaging also acts as a low-pass filter to lower the noise floor by removing higher frequency components as well as their noise. Although the bin spacing is identical, the noise is lowered which improving the accuracy of the parabolic fitting that determines the measured frequency. Unfortunately, there are two issues here: 1. the lower number of samples is faster to compute, but inherently noisier and fewer samples contribute the each bin. 2. the new scheme aliases at 6 kHz, making completely incorrect results for higher frequencies. Higher levels of averaging will alias at even lower frequences.

On the other hand, we could grab, say, 2048 samples at 48kHz and average down to 512 samples at 12kHz to get higher resolution at 3-6kHz, or even 128 samples at 3kHz for a computationally efficient look at the 750Hz-1.5kHz range. This techniques improves the noise and improves the FFT bin-spacing as well. Of course, the longer sampling lowers the speed at which one can report the higher resolution result, increasing measurement latency. (4096 samples at 48kHz take 83mS).

The FFT bin-width is the effective sample frequency divided by the number of samples. Noise is an ever present issue in any measurement system. After you have determined the bin-width of the FFT output you want, remember that you can run multiple FFTs and average the results to get lower noise (by averaging the various noise sources). This may take significant real-time at low frequencies – but can improve the accuracy by lowering the noise. It can be a good idea to rapidly get out the current best estimate – and update as soon as possible with more accurate results. The audio amplitudes as captured by the Nexys-4 microphone are sampled at 48.828125 kHz (100 MHz/2048) – subject to the possible inaccuracies of the main system clock. It is your responsibility to adapt the sampling to allow rapid and accurate determination of the largest amplitude harmonic from the sampled input. (As a last point, small speakers driven to large volume levels at low frequencies (e.g. a phone at 55 Hz) may actually produce 110 or 165 Hz as the actual maximum harmonic because of distortion. This means use a ear or head phone and beware of overdrive. This is part of the rationale for test modes built-in to the tuner.)

1. Focus on the timing around **read_fsl_values()**, which is the function responsible for reading the microphone samples from the Microblaze buffer.

   (a) The default code gets 512 values into the grabber (by default – the buffer can be extended to 4096), then reads

them into the microblaze. Depending on need, read fewer values, or averaging/decimating the buffer data. To improve accuracy and sensitivity, a longer capture can be used.

(b) Given this, experiment with different effective sample frequencies (after decimation) to speed up your FFT and increase the accuracy of your design. Review the earlier section on the FFT and octaves, and think about how you could change the code of the FFT to make sure the target note falls in the upper half of the FFT.

(c) Note that the grabber, once started, runs in the background. Thus one can either pipeline the system (start the next grab while computing the current FFT), or use the non-blocking sample checks in a GUI to be more responsive while waiting for data.

2. Analyze your LCD code and rotary encoder code, and make sure functions which use loops to cause delays are removed throughout your design.

3. The LCD code uses only a subset of the features of the LCD internal processor. It is often blocking code, so breaking up long running functions and careful updating can greatly improve the performance. (Alternatively, both the display and sampler can be run in background with commands passed from the FSM to make updates).

4 Grading/Evaluation

The Chromatic Tuner should perform flawlessly, despite a user who might push buttons mid-update, or excessively twist the rotary encoder (this means that crashing the code from the buttons or encoder should be avoided). The Chromatic Tuner should function over a range of frequencies from 65Hz (C2) to 4,200Hz (C7) and will be expected to perform continuously in real time. The +/- 20cent accuracy of the Chromatic Tuner should hold over the range of 80Hz-180Hz (E2-F3). The +/- 5cent accuracy of the Chromatic Tuner should hold over the range of 200Hz-4200Hz (G#3-C7). This requires some thought spent on calibration – be sure to check your tuner on more than one test application. Finally, while the lab describes the required behavior of the tuner, any additions including nicer fonts, backgrounds, displays or function to make the tuner easier to view or nicer to use will be counted accordingly. Imagine you were a member of a small company making a product that needs to compete against Korg, Sony, etc. – a little bit of thought could change the number you sell... and keep you in business.

4.1 Reporting (50/100 pts)

Since this is the final project, your write-up can be up to 6 pages long. Do not include long code fragments, unless you also provide a caption explaining what the code does and why it is there. Describe the structure of your design, as well as how you modeled, optimized and tested it. Include details on your key design decisions. Testing is a key part of the lab evaluation. If you have an iPhone or Android based phone, there are apps that produce sine tones at any frequency to fairly good accuracy. Guitar tuner and Cleartunes are two such apps that you can use for your input frequency source. If these are not available, there are many test tones available on the internet. One source on the internet that has been tested by us to be accurate is: https://www.szynalski.com/tone-generator/

It is also a good idea to test your design by using earphones to drive the microphone. In our experience, in-ear buds work best, and indeed simpler buds often work much better than very expensive ones as they have no filters or other modifications. Place the bud directly on the mic port on the board (The small hole labeled MIC on the left side of the card between the two

PMOD ports.) Please describe the procedures you used to test the Tuner in the report, sufficiently that a user unfamiliar with the project could perform the test. **Test your design carefully and often.**

4.2 In Person Demo (50/100 pts)

1. **Basic Functionality (15 pts)** Tuner mode (note mode + frequency mode), settings page/mode (base frequency, mode switch), debug page/mode (FFT histogram, octave selection, debug information, etc)

2. **Tuner Accuracy (15 pts)** We will test several off-note frequencies, and measure the accuracy of the tuner. We will test several notes across the range, adding tests if problems are seen or to evaluate usable range.

3. **UI Fluidity (10 pts)** We open the user interface, but grade on ease of use, screen readability, smoothness during display changes, and response time. When measuring low frequencies, sampling time and FFT calculation time should not negatively affect UI response time. Also, don't sacrifice response time for better UI.

4. **Aesthetics and Usability (10 pts)** The UI needs to have good colors, easy to see fonts, nice offset graph display (-50 to +50 cents), good background, sensible use of controls etc.

We hope that working on this project is an enjoyable experience and we look forward to seeing your final projects!