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 we need to make our early prototype of the chromatic tuner a finished embedded system. We will create organized C code by incorporating the FFT code into the QP-Nano based HFSM GUI built in previous labs. Lab 2B provides the basic structure for The Chromatic Tuner. 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. Our 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 Required 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.

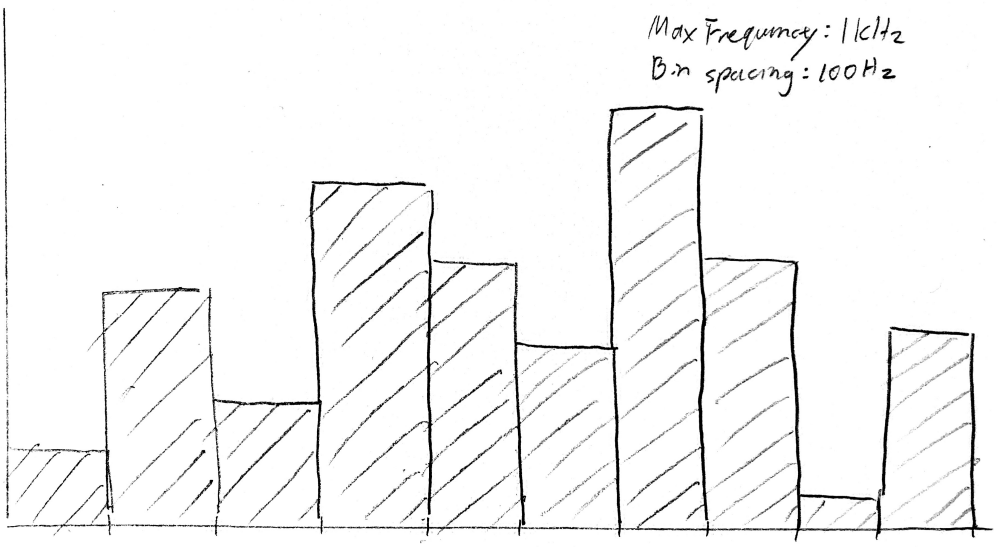
2. Also display the error estimate of input tone in geometric cents. Generally, equal-tempered notes are spaced at geometric spacings of $2^{1/12}$ with 100 geometric divisions between each successive note (thus called 'cents'). This error must be displayed as a horizontal graph, with the note in the center and a bar to the right if the played frequency is too high, and to the left if it is too low.

3. The tuner makes the assumption that A4 = 440 Hz which does not allow for relative tuning with other fixed instruments such as old church organs or harps which often have Italian or Pythagorean tuning standards. It was decided that the

equivalent tuning of A4 should be variable from 420 Hz to 460 Hz in 1 Hz intervals. For this feature, tie the value of A4 to your encoder. As soon as the encoder is turned display the current frequency value for A4 and increment/decrement it by 1Hz. Once the encoder has stopped moving for 2 seconds remove the frequency of A4 from the display and update your octave ranges based on the new value (I.e. if A4 is switched to 430 Hz, then the 4th Octave ranges from 430 Hz - 860 Hz)

4. You will be required to make 2 test modes to help incrementally code and test your design. We recommend implementing them in the following order. (we will be asking for both during the final demo)

   FFT histogram You will create a function mode that shows a histogram of the FFT. Each vertical bar should be a single bin and organized from left to right. (lowest to highest number bin) The histogram should also show what the highest frequency is and the bin spacing. This way you can get an idea of what the received sound waveforms look like in frequency space. Remember, the FFT is mirrored across the Nyquist frequency. Is it useful and/or necessary to show the mirrored half? Should the Y axis be logarithmic? Make sure you can justify your histogram choices. A basic diagram of what we expect is below. (do not just make what is below. It is just a guideline on what it might look like.)



   Spectrogram (Optional)

   The FFT histogram is useful only for instantaneous results. In the real world, we use spectrograms to analyze waveforms over time. If you want, you can create a spectrogram in place of the FFT histogram. This can be beneficial for debugging and seeing the behavior of your FFT over time. Look online for examples of a spectogram plot.

   Octave selection You will create a menu that selects between octaves 2-7. Once you've selected an octave the tuner will work as described previously, however you can guarantee only frequencies within that octave range will be played. This will be very helpful is getting your code started before implementing the full tuner.

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

5. You will have a default menu that allows you to select between the full chromatic tuner functionality and some of the test functions. 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 rotary encoder should function in real time. We already solved a part of this problem in Lab3A by improving the FFT response time. Your target is to integrate your FFT code with a stable Chromatic Tuner GUI so as to have a fluid interface and response with reasonably high accuracy. (I.e. be able to see the 1 Hz errors in A4 setting)

3 Recap on Octaves, Improving FFT Accuracy and Stability

The frequencies detectable by our microphone range from tens of hertz to the kilohertz range. 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:

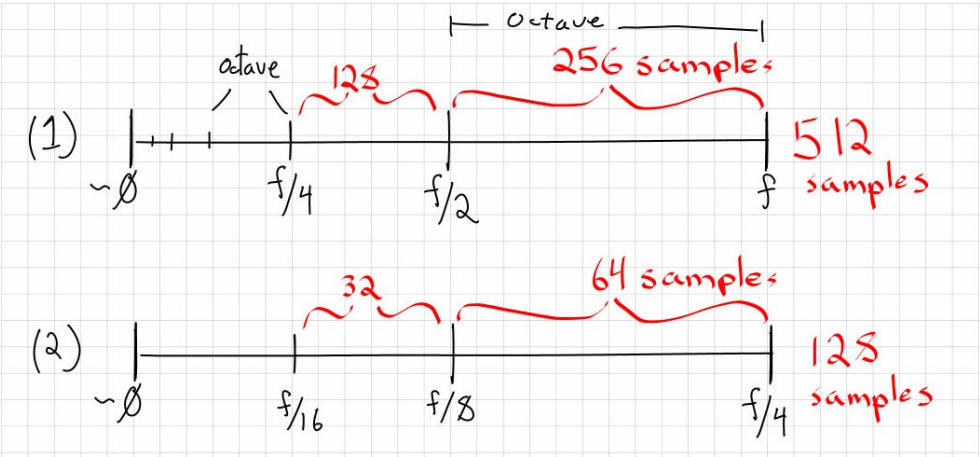
$$f = 440\ Hz * 2^{\frac{(n-9)}{12}+k-4}$$

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 octaves to identify the note is very 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.

3.1 Fast Fourier Transform

The fast Fourier transform places recorded samples into frequency bins. The FFT is calculated up to the Nyquist frequency (half the sampling frequency) which here is 24 kHz. The resolution of the frequencies placed in the upper half of the range is greater than those placed in the lower half of the range. This is shown in the figure below. Looking at Fig. 3.1:plot(1) If we are analyzing the raw samples from the microphone, the upper half would contain frequencies ranging from 12 kHz to 24 kHz. This means that if we are trying to detect a lower frequency, like 100 Hz, only a few samples are used.



There are two ways we could fix our resolution, and potentially increase the speed of our FFT. 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 decimation or averaging makes an 128 samples at 48 kHz/4 = 12 kHz effective sample rate so it will have high resolution for signals from 3 kHz to 6 kHz. **Averaging also acts as a low-pass filter to lower the noise floor by removing higher frequency components.**

On the other hand we could grab, say, 2048 samples at 48 kHz and decimate/average down to 512 samples at 12 kHz to get really good resolution at 3-6 kHz, or even 128 samples at 3 kHz for a computationally efficient look at the 750 Hz-1.5 kHz range. In any case, 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 – possibly running to seconds of sampling. It is 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.)

Lab 3A provides default source code to verify functionality of the microphone. The demo code tells the stream grabber to start grabbing data from the microphone, waits for it to fill its buffer, then copies that into a software buffer. This occurs in microblaze_bread_datafsl(). The sample code provided in **src/fft.c** computes the Fast Fourier Transform (FFT) and xil_printf() outputs the guessed base frequency. The FFT response time for the default code is long and must be improved. To improve the code it is necessary to to conduct a performance analysis. There are three sections of code in particular to focus on.

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 existing code starts the sample grabber, waits until it gets 512 values into the grabber's buffer, then copies them into a software buffer. In fact the grabber keeps going up to 4096 samples but the software doesn't need to check this. The hardware is fixed at the microphone's sampling rate so lower effective sampling rates must be achieved in software.

   (b) The existing code gets 512 values into the grabber (by default – the buffer can be extended to 4096), then reads them into the microblaze. Look into modifying the process of reading from the buffer. Try reading less values, or decimating the buffer data. Experiment with the conversion to floating point. Moving a single line of code can result in dramatic changes to your code performance. Equivalently for accuracy a longer capture can be used.

   (c) 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 octave falls in the upper half of the FFT.

   (d) 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. Study the FFT code, and focus on finding small changes which result in noticeable speed-ups of your code. The original code uses floating-point arithmetic. Rewriting the FFT to use a fixed-point FFT (which would remove the need for floating point number's altogether) would allow for a faster FFT. Fixed point has the difficulty of requiring being careful about round-off errors and overflow of the values.

   This class of optimization is particularly important since if you can run a 512-point FFT in the time it would take someone else to run a 128-point FFT you can have that extra speed without the accuracy trade-off of a smaller FFT.

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

4 Grading/Evaluation

To achieve a high grade, The Chromatic Tuner must perform flawlessly, despite a user who might push buttons mid-update, or excessively twist the rotary encoder. The Chromatic Tuner should function over a range of frequencies from 110Hz to 4,000Hz and will be expected to perform continuously in real time. The +/- 10cent accuracy of the Chromatic Tuner should hold over the range of frequencies. 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 or displays 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, and the cheaper imports– a little bit of thought can 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. 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 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.) Be sure to test your tuner in octaves 2-7. 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. **It is strongly recommended that you use one of these methods to test your design. While there are many other phone and web-based apps available to generate tones, their accuracy is highly questionable.**

4.2 In Person Demo (50/100 pts)

The final demo procedure has not yet been finalized, however visual appeal, nicer fonts, color choice, ease of use, fluidity and stability of the interface, good performance, accuracy, range and completeness of the project all contribute to the final grade. We hope that working on this project is an enjoyable experience and we look forward to seeing your final projects!