ECE 153A Final Project - The Chromatic Tuner

By: Ananth Pilaka (4200580) and Snehith Nayak (3961497)

Goals and Purpose

The goal of this project was to create a chromatic tuner, similar to ones used by musicians to determine frequency and the note being played. Along with other features, the key functionality of the tuner is to accurately display the musical note and octave of the frequency being played down to the cent. Primary focus of the tuner implementation was directed towards the accuracy of the tuner. By setting different input parameters, we aim to determine the optimal sample rate for our Fast Fourier Transforms for each octave present in our frequency range, and subsequently, the correct musical note being played. Some key functionality, like the debouncing of the rotary encoder and a partially-optimized chromatic tuner built in Lab 3A were repurposed. Other features such as a frequency-bin histogram from each FFT run and an octave selection mode were also implemented to improve the debugging capabilities of our design. To meet the accuracy goal of our chromatic tuner, a horizontal bar displaying frequency cent error from the nearest note/note frequency was designed to enhance the functionality of the tuner. Additionally, we worked to create an overall UI menu in which we can switch between different modes (tuner, octave select, settings, and histogram).

Design and Modeling

The most fundamental element to our design was its structure. The core of the GUI LCD display is contained in gui_hsm.c, which contains a QP-nano based Hierarchical State Machine. This HSM takes inputs from an

interrupt-based program control, which is configured and initialized in bsp.c (an extension board support package file). This C file configures the peripherals (rotary encoder, onboard pushbuttons, and AXI Timer) to fire interrupt signals defined in the Vivado hardware configuration, and run custom interrupt execution code. Each of these custom interrupt execution functions dispatches inputs to the QP-nano framework, which is then handled with the appropriate context within the HSM.

There are 5 main functional states within our state machine (and more specifically within LCD_on, the super state): LCD_Menu, LCD_Tuner, LCD_Hist, LCD_Settings, LCD_Octave. On an ENCODER_CLICK signal, a transition either made to the main menu (LCD_menu) or to the state selected in the menu. A key element of our design was that in the QF_onIdle function, we performed our FFT with the set FFT parameters, and then determined the frequency of the signal as well as the cent error and note octave. Using global static flags set by entry and exit actions in the appropriate states/transitions, we were able to determine when it was necessary to print these values to the LCD. While the context for actions in each state was generally intuitive, it is worth noting that in the LCD_octave state, a global static flag, Auto, is set to 0 (false) on its entry action to indicate that the auto function, which automatically sets the FFT parameters based on the measured octave, should not automatically scale the FFT parameters. Instead, we allowed the octave to be chosen in the LCD_Octave state, and held the FFT parameters constant.

Optimizing

Since the default FFT code provided was extremely inefficient, we had to optimize the FFT to provide a result in < 30ms. To achieve this we had

constructed a sine and cosine lookup table and included rolling averages of the raw samples in Lab3A. With the goal of the having the played frequency's octave fall in the upper half of the Nyquist bandwidth, we were presented with the problem of trying to dynamically change the sample size, the decimation and averaging factor to have the octave/frequency fall within the most accurate portion of the FFT.

To this end, ideally, we had to have the effective sample frequency as such: s_f > 2*(the highest frequency in the octave). We had to optimize this with the raw sample size, as we also had to achieve smaller frequency bin widths to be able to distinguish frequencies/notes in the lower octaves. A higher decimation/ averaging rate lowers the effective sampling rate and the number of samples by the same factor, which does not truly improve our frequency bin spacing. Therefore, we had to optimize the utility of increasing the number of raw samples in the buffer with the effect this would have on the speed of our FFT. Utilizing the octave select test function, we determined (through trial and error) the optimal set of FFT parameters that yielded the most accurate FFT in that octave, while still providing a reasonably efficient FFT. We then modified the auto function to utilize the determined FFT parameters for frequencies converging to that octave, thus yielding the most accurate frequency identification by the chromatic tuner.

Testing

To verify that our design worked reliably, the first thing we did was procure a moderately powerful speaker that was capable of playing all of the frequencies in the range 65 Hz to 4500 Hz (similar to the speaker used in demo). We used the recommended sine tone generator to produce note

frequencies and the off-note frequencies in the given range for octaves 2 - 7. We always began testing the tones in the histogram mode, to determine that the correct frequency was being picked up by the microphone, the noise was not ruining the samples, and that the frequencies were not being clipped. We then would test the frequency in octave select mode, to first test that the frequency was not being aliased (related to the testing done to determine FFT parameters for an octave) and subsequently test that the frequency was being read by the chromatic tuner accurately. Finally, we tested the frequency in the general tuner mode, to ensure that the auto function would converge to the correct octave and frequency. We repeated this process for around 4-5 notes per octave (as shown in the notes table in the recommended tone generator).