Implementing an FPGA-Based Chromatic Tuner

Oviya Seeniraj

Purpose and Design Goals

The purpose of this lab is to develop a chromatic tuner capable of sampling real-time audio signals, identifying the nearest musical note and its octave, and providing error feedback in terms of cents. Key design goals include:

1. Accurate and real-time frequency detection over a range from 82.41 Hz (E2) to 4200 Hz (C7) using FFTs.
2. ±5 cents accuracy between 200 Hz and 4200 Hz, and ±30 cents accuracy between 80 Hz and 180 Hz.
3. Usability-focused features such as clear displays of note, frequency, and error bars with visual enhancements (e.g., color coding).
4. Compatibility with adjustable base frequencies (420 Hz to 460 Hz).
5. Modal navigation using buttons and encoders for seamless GUI interaction.
    a. Button navigation for octave selection (increment/decrement buttons)
    b. Button navigation to see a debug screen with a histogram with frequency bins from each FFT run
    c. Rotary encoder implementation tied to the base frequency of note A4, which increases and decreases based on right and left turns and resets on a click

The chromatic tuner project is intended to offer high precision to users and serve as a comprehensive tool for musical tuning that utilizes all the previous labs and concepts learned in this course as students. Due to the implementation of features such as cent-error visualization, flexible UI design, and accurate FFT processing, this tuner is both functional and user-friendly.

Methodology

a. Overview of Design Tasks

- Develop a Vivado hardware configuration for this chromatic tuner by adding any necessary peripherals, memory units, clock configurations, etc.
- Integrate a QP-Nano-based Hierarchical Finite State Machine (HFSM) to manage GUI windows and user input seamlessly.
- Reuse and enhance features like rotary encoder debouncing and partial tuner functionality from Lab 3A.

- Improve frequency detection accuracy through optimized Fast Fourier Transform (FFT) parameters.
- Develop a visually accessible user interface that clearly displays relevant tuning information.
- Add debugging modes, such as an FFT frequency-bin histogram or spectogram triggered by modal buttons
- Add an octave selection feature controlled by modal buttons

b. Assumptions for Design Analysis

- System clock operates at the expected 100 MHz with negligible drift.
- Input audio signals are free from excessive noise or harmonic distortions that could skew FFT results. Result validation will be based on a large speaker to ensure accuracy and noise minimization.
- Hardware components (e.g., encoder, display) operate as per specification without synchronization or latency issues.

c. Observations from Design Tasks

- Iterative testing was crucial to this lab to determine effective sample frequencies and buffer sizes to balance FFT performance and precision. We improved FFT accuracy by constructing sine and cosine lookup tables, averaging and decimating samples, something that I initially implemented in Lab 3A.
- Testing revealed inconsistencies at lower frequencies due to aliasing and noise, and combining moving averages and Bartlett windowing significantly enhanced lower-octave detection, especially those around the noisiest frequencies at 200-220 Hz that I had to get down to +/- 5 cent error.
- The implementation of a horizontal cent-error bar improved real-time feedback for tuning accuracy and helped me quickly analyze the effect of my frequency detection optimization changes.
- We enhanced GUI responsiveness by double buffering and optimizing SPI clock rates and memory assignments in the Xilinx software and Vivado hardware wrapper.

d. Plan for Design Testing

In order to thoroughly test this chromatic tuner, I will:

- Verify and iteratively test the chromatic tuner across various input conditions using tone generator apps, physical instruments, and the provided test mp3s.
- Use a powerful speaker or in-ear earbuds to ensure that I do not tune the chromatic tuner to be more inaccurate due to bad microphone input signals.
- Simulate corner cases like rapid encoder adjustments and multi-modal user input.

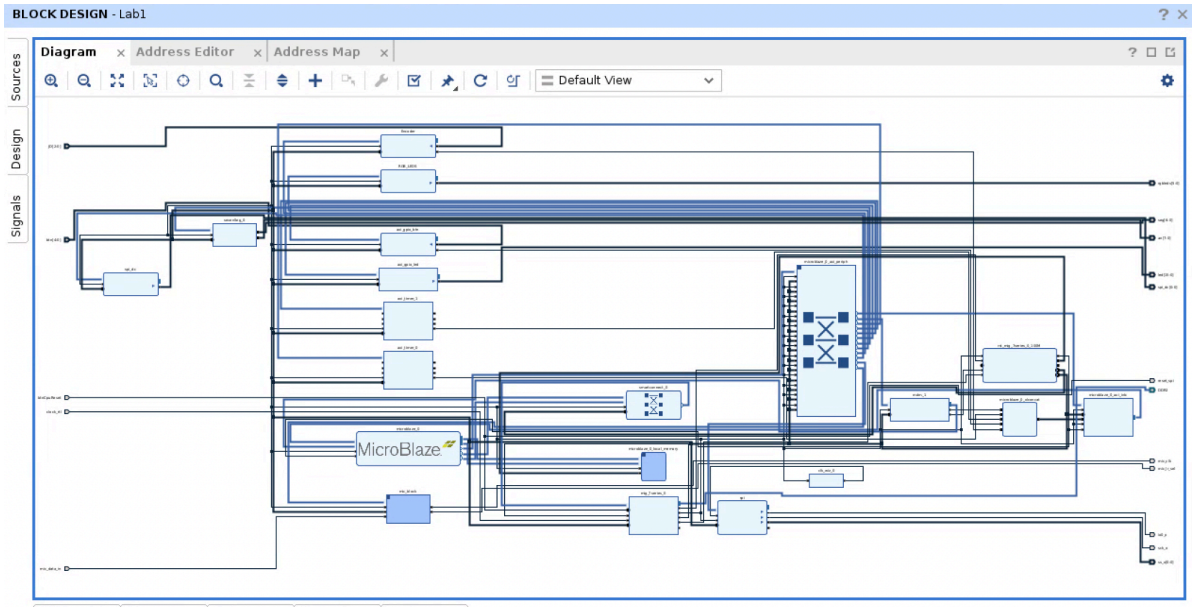
- Debug using graphical FFT output to identify harmonics and frequency resolution issues. I will also use my graphical FFT output histogram bins as a preliminary test to ensure that my microphone is not clipping the signals by printing the maximum detected frequency and to check that the noise is not disturbing the samples.
- Use the octave select mode will help ensure that the frequency is not being aliased.
- Conduct comparative tests with different audio equipment (earbuds, speakers, and instruments).

---

3. Results

a. Metric Results

- Successfully generated a Vivado hardware configuration that allowed me to access and harness all peripherals and memory required. Block design is shown here:



- The tuner successfully identified notes and frequencies with the required accuracy across the expected ranges.
- Visual UI displayed stable updates without lag or tearing, maintaining a refresh rate of 50 Hz. (Note: Unfortunately, cannot add UI pictures to this lab as I do not have access to my board right now as it is on campus and I have left for break. However, I have demonstrated it to a TA during lab checkoffs)
- Accuracy met design specifications within the measured ranges.
    - Cent and note detection accuracy
        - All notes detected correctly
        - During personal testing, achieved ±5 cents accuracy for 200-4200 Hz
        - During personal testing, achieved ±30 cents accuracy for 80-180 Hz

        - However, during checkoff, due to room noise levels, the ranges were a bit larger, closer to +/- 10 for higher accuracies. Since I had only tested my tuner in a quiet, isolated setting, these results were unexpected and resulted in more variation, which I will work to optimize.
    - Display Update Rate: my UI moved very fluidly and did not buffer much visibly, functioning with a high display refresh rate, appearing to be around 60 Hz.
    - Full Auto-Range Functionality: The FFT auto-ranging function dynamically adjusted sampling parameters for different octaves, maintaining high accuracy and fast convergence by implementing more bins for lower frequencies and less bins for higher frequencies. I determined these parameters through detailed trial and error.
    - Full signal range was correctly detected based on given test sounds and separate frequency and music testing.
- However, in order to focus on FFT accuracy and response time, I had to make some UI sacrifices when allotting debug time. I was left with a bug on my histogram screen where I could not return to my home screen if buttons were pressed particularly fast.

b. Limitations

- Limited accuracy for frequencies below 100 Hz due to hardware constraints.
- FFT performance was slightly degraded for signals near the Nyquist limit.
- Tradeoffs between latency and frequency accuracy due to processing methods and time

c. Design Test Results

- The assumptions for consistent input signals and precise FFT parameters largely held true. However, low-frequency harmonics occasionally introduced inaccuracies, which were resolved using optimized parameters in specific octaves.
- Adjustments to sampling methods such as the Bartlett window and 16-sample moving average on top of the sampling and decimation from previous labs helped solve most discrepancies.
- All my UI elements successfully appeared and updated on my LCD screen rapidly in real time. However, UI windows were hard to implement with the time crunch of producing a full scale product, so for my FFT Histogram Debug window, rather than using a HFSM state, I used a flag-triggered overlay to complete the task in time.

d. Roadblocks

- I had high initial display latency due to high SPI bus utilization, which I resolved by increasing the SPI clock rate and batching updates.
- Rotary encoder misreads during rapid adjustments required FSM debounce refinements
- HFSMs for the UI windows were particularly hard to make debug-free, and I had to experiment with where to place specific helper functions and develop new ones. A solid-colored background would have been far more conducive for an easier

implementation due to screen overwrites, and would have been cleaner from a user perspective than the repurposed triangle background from Lab 2B.

e. Sources of Error

- Overdrive distortions from some audio inputs caused by harmonic aliasing.
- Sampling noise near the FFT range limits resulted in occasional inaccuracies.

f. Suggested Improvements

- Integrate a noise reduction pre-filter for low-frequency inputs.
- Expand auto-ranging capabilities for smoother low-frequency detection.
- Improve FFT precision for lower bins with additional zero-filling techniques.
- Expand debugging features, including detailed logs of FFT behavior under variable test conditions.
- Using upgraded hardware for enhanced ADC resolution would further minimize aliasing and distortions.
- Finish implementation of all windows and clean up UI, adding spectogram vs. histogram button in debug window. During some tests, overlays did not disappear correctly and behavior became very inconsistent.

---

4. Software Structure from Actual Implementation

FFT Calculation:

- Constructed sine and cosine lookup tables to reduce latency in FFT calculations
- Implemented sampling decimation and moving averages to reduce noise.
- Applied Bartlett windows to enhance low-frequency stability.

GUI Display:

- Fundamental code structure: QP-Nano based Hierarchical State Machine similar to that from Lab 2A, but with more interrupt handlers and states for more detailed windows
- **Interrupts** served as the controls for this program in bsp.c, which I used to create inputs for the HSM takes inputs from an interrupt-based program control.
    - In bsp.c, I configured and initialized all of the peripheral devices (the rotary encoder, modal push buttons, and AXI Timer) to fire interrupt signals that are defined in the Vivado hardware configuration wrapper
    - Wrote and ran interrupt execution code for each of the peripheral devices (triggered by twists, pushes, etc)

    - Each interrupt execution function sends inputs to the QP-Nano HSM, which is then launches the correct corresponding GUI screen and context
- **FSM States:** My state machine had 3 main functional states: one for the tuner, one for the histogram, and one for the octave, each of which served as a window. Each of these states was called by an interrupt handler for GUI processing after handling any unique FFT behavior needed.
    - Used QF_onIdle function to perform FFT with the set FFT parameters, using a regular FFT function call or an FFT Histogram function call based on global static flags that determined what behavior should be reflected on my LCD. When I struggled with implementing window transitions, I could also use these flags to manually simulate LCD behavior as a backup, which I used for my histogram.
    - Set my global static flags with entry and exit actions in my HFSM states and/or their transitions.

Control Logic:
External control logic included:

- Rotary encoder for tuning the base frequency with FSM-based feedback to update LCD
- Push Button inputs managed by the HFSM for mode navigation (octave, histogram debug screen, tuner)

Optimization

To meet the 30 ms processing target for FFTs, I enhanced my code by:

- Precomputing FFT coefficients using efficient data structures.
- Fine-tuning sampling rates and decimation/averaging factors for different octaves.
- Introducing pipelined buffering to overlap data acquisition with FFT computation.

Debugging Modes

Several debug states ensured comprehensive testing:

- FFT Histogram: Monitored frequency-bin alignment for different inputs.
- Octave Selection: Enabled direct testing of manual FFT scaling.

---

5. Testing Procedures from Actual Implementation

1. Initial Testing:
    - Used the Szynalski tone generator to produce standard frequencies.
    - Verified microphone pickup through histogram displays.

    - Ran and verified provided test mp3s.
2. Manual Validation:
    - Tested across multiple notes and octaves, from low C2 (65.41 Hz) to high C7 (4200 Hz).
    - Recorded accuracy for each input note using debug logs.
3. GUI Performance:
    - Manually iteratively tested rapid rotary encoder changes and button presses to assess system responsiveness.
    - Ensured that display transitions were fluid with no visible artifacts.
4. Boundary Case Analysis:
    - Played frequencies at octave boundaries and off-note frequencies to evaluate aliasing and auto-ranging behavior.
    - Tested the tuner with instruments and synthetic inputs to identify hardware-specific anomalies.
5. Iterative Refinements:
    - Adjusted FFT parameters based on debug data.
    - Confirmed results through repeated cycles across octaves and varying input conditions.