Autotune Design Document

April 15th, 2025

Amanda Jenkins (alj2155), Charlie Mei (jm5912), Millie Chen (sc5405), Meng Fan Wang (mw3751)

---

1. Introduction

This project implements a real-time pitch correction system on the DE1-SoC board, allowing users to sing into a digital microphone and hear their voice tuned to the correct musical notes. The microphone connects directly to the FPGA, capturing the user's voice in real time. The system is controlled through a keyboard interface: the user presses a key to start recording and begins singing, and presses another key to stop. Throughout the interaction, prompts and progress updates are displayed on a screen connected to the board, guiding the user through each stage.

Once recording is complete, the system processes the captured audio, identifying and correcting any off-pitch notes to match their nearest intended musical tone. During this pitch correction stage, the user sees live progress updates on the display. When processing is finished, the user is prompted to press a key to play back the tuned audio. The processed output is played through speakers connected to the board, and playback can be repeated as desired.

2. System Block Diagram

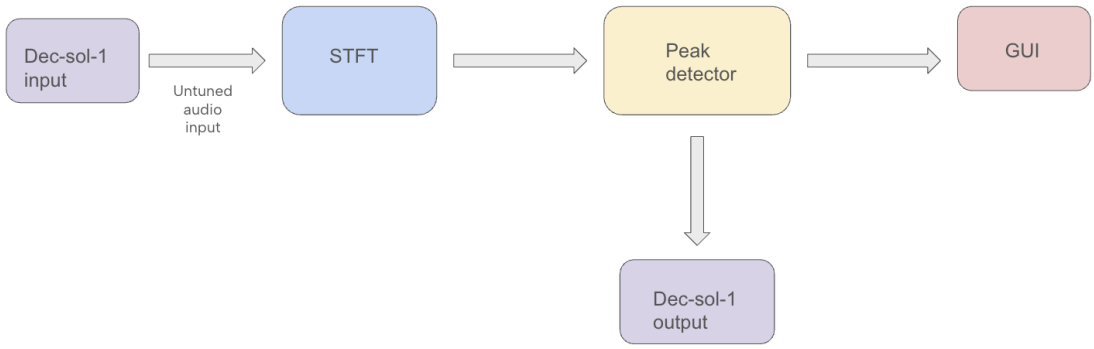

Figure 1: High-Level Block Diagram

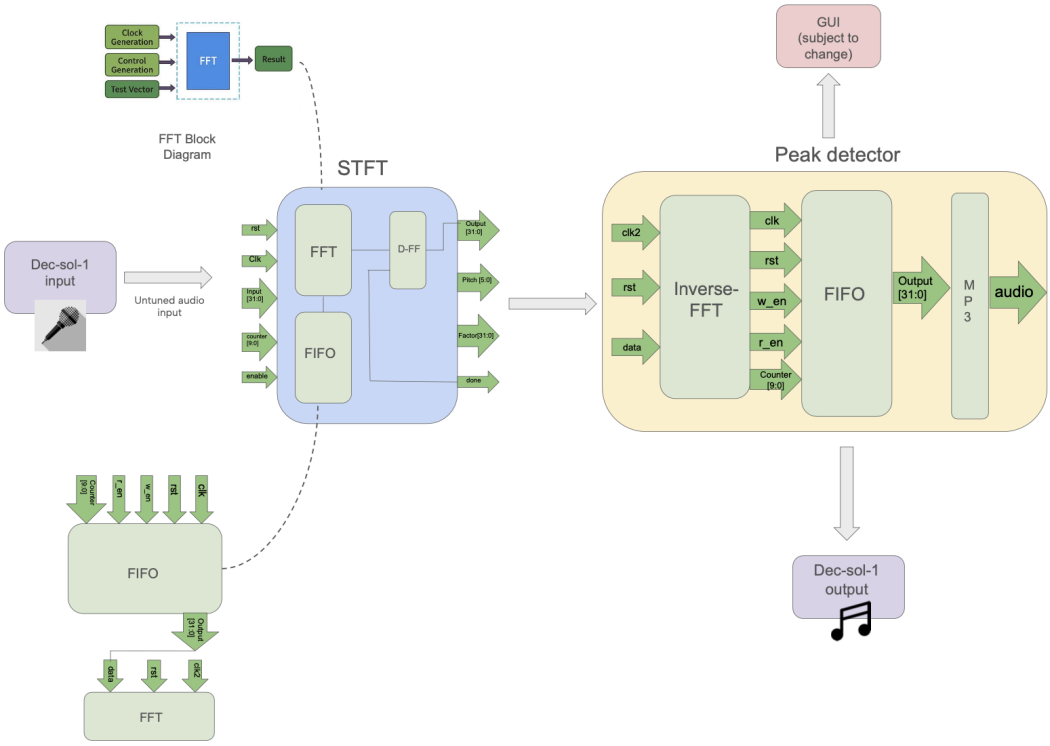

Figure 2: Total system block design showing the how the system is integrated

Our system is implemented on the DE1-SoC platform, leveraging both hardware and software components to achieve real-time pitch correction via autotuning. Figures 1 and 2 provide high-level and detailed block diagrams of the overall architecture, illustrating the flow of audio data through Short-Time Fourier Transform (STFT), pitch detection, and signal correction modules, as well as how the hardware and software components communicate.

- Dec-sol-1 Input (Audio Input via MIC-In)

The DE1-SoC board utilizes a 24-bit audio CODEC for capturing high-fidelity audio. The MIC-in port allows connection of an external microphone. Analog audio signals are digitized via the onboard CODEC and sent as digital audio streams to the FPGA. Communication is as follows: Analog input → CODEC → Digital stream to FPGA via audio bus interface.

- STFT Block (Hardware)

The Short-Time Fourier Transform (STFT) block consists of FIFO Buffers which buffer incoming audio frames and help manage timing mismatches. An FFT Core which converts

time-domain audio frames into frequency-domain representations using a streaming FFT IP core. Also, D-Flip Flops which synchronize intermediate signals and manage timing for downstream logic.

- Peak Detector (Hardware)

The peak detector identifies the dominant frequency in each FFT frame using a magnitude-based peak detection algorithm, which compares the detected pitch with the closest musical note frequency and determines the necessary frequency shift.The Inverse FFT (IFFT) transforms the corrected frequency-domain signal back to the time domain and the FIFO Buffers continue to regulate timing and data transfer between processing stages. The MP3 Encoder (optional) converts audio to compressed format for output.

These blocks communicate using internal data buses. The FIFOs use handshake signals to control when data moves between parts of the system.

On the software side, the HPS (Hard Processor System) runs Linux and controls the hardware. A system controller program, written in C, sets up the hardware, starts the process, and checks system status. It talks to the FPGA through memory-mapped I/O using the AXI-Lite bridge. A graphical user interface (GUI) lets users upload audio, pick scales, turn effects on or off, and see pitch tracking.

- Dec-sol-1 Output (Audio Playback via Speaker)

The corrected time-domain audio is sent from the FPGA back through the DE1-SoC's CODEC to the Line-Out port. Users can listen to the pitch-corrected audio in real time through the speaker.

3. Algorithms

The core algorithm for this project is Fourier Transform with the given audio file. Specifically, we are going to apply STFT (Short-time Fourier Transform) and FFT (Fast Fourier Transform). In other words, we will divide the audio file into frame buffers, apply FFT to determine the frequency, and make proper adjustments to it.

Specifically, each buffer will have:

1. 1024 data points
2. Sampling rate of 8kHz.

Besides, we will utilize Intel FPGA core package to conduct FFT and IFFT algorithms for the signals, therefore we do not have to worry about the detailed implementation when transferring the signals in between time-domain and frequency domain.

FFT Intel FPGA IP Core:
https://www.intel.com/content/www/us/en/products/details/fpga/intellectual-property/dsp/fft.html

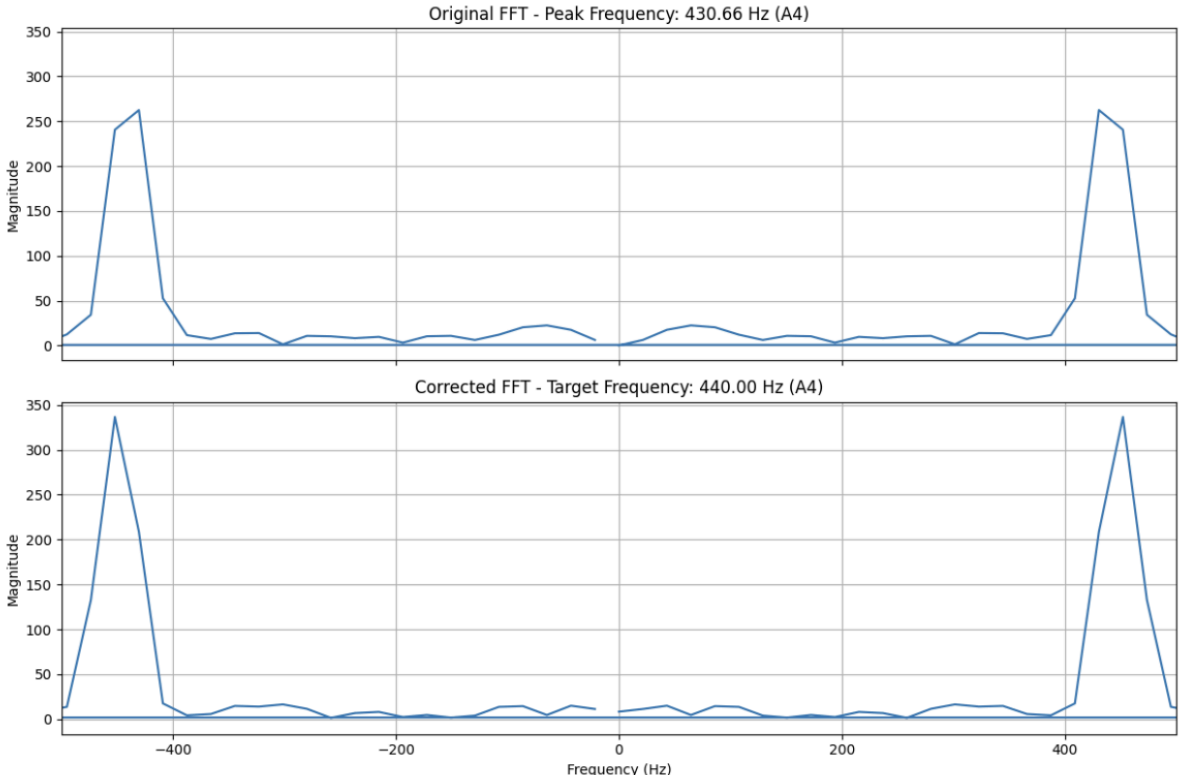

Figure 3: Python demonstration of pitch detection and tuning.

For every FFT generated, we will apply a peak detector algorithm to find out the dominated pitch in that frame buffer. Then we will match this frequency with the closest note on piano (assume you sing not perfectly in tune and we assume you try to sing to the closest note on piano). Before we apply IFFT algorithms, we will first stretch the FFT graph proportionally to tune it correctly, while maintaining the noise to keep the voice texture unchanged.

The graph above demonstrates how the pitch is tuned in frequency-domain before applying IFFT. We will use IFFT to convert the signal back to time-domain, which will be used as output audio.

Sampling rate: we want to reach the frequency resolution of 8Hz. With 1024-point FFT and the formula: frequency resolution = sampling rate / (# of points), we will have a sampling rate of approximately 8kHz. With the sampling rate, the Nyquist frequency will be half of that, which is 4kHz. Given the highest pitch is A6 (1760Hz), which is well below the Nyquist frequency, the module will function correctly.

4. Resource Budgets

Raw Audio Storage:
2 RAM blocks for storing audio captured from the digital microphone, each storing ~2048 16-bit samples: ~8 KB total

Processing Buffers:
2 RAM blocks for storing intermediate FFT inputs and outputs:
~8 KB

Pitch-Corrected Output Buffer:
1 RAM block to hold tuned samples before playback:
~4 KB

VGA Display Text Buffer:
Stored in a ~2.5 KB on-chip memory buffer

So total expected RAM usage: ~22.5 KB

5. The Hardware/Software Interface

Userspace program:

1. Handling Keyboard Input for Control Signals
   The user controls the system using keyboard input (e.g., 'S' to start recording, 'E' to stop, 'P' to play output). A dedicated keyboard control driver captures keypresses and communicates user commands to hardware by writing to memory-mapped control registers via the ctrl_dev driver.

2. Displaying Prompts and Progress on Screen
   A display driver (text_disp_dev) interfaces with a hardware VGA module to show text-based prompts and progress messages (e.g., "Recording...", "Processing 50%..."). The userspace program writes to this device to update instructions based on system state.

3. Initiating and Monitoring Pitch Correction
   After recording is stopped, the program sends a signal to start processing. During correction, it monitors a hardware busy or progress flag via the ctrl_dev driver and updates the display accordingly.

4. Controlling Playback of the Processed Audio
   When pitch correction is complete, the user is prompted to press a key to play the

tuned output. The program sends a playback trigger via the control driver and may monitor playback progress or loop on request.

Device Drivers:

1. kbd_ctrl – Keyboard Driver
   Captures input from a PS/2 or USB keyboard and forwards user commands to the control register space. It may also directly interface with the control device for writing flags like start_record or play_output.

2. ctrl_dev – Control & Status Driver
   Exposes memory-mapped control signals for the hardware FSM:

   Control: start_record, stop_record, start_processing, play_output

   Status: recording, processing_done, playing, busy, progress. This driver enables clean synchronization between the user interface and hardware state.

3. stream_dev – Audio Streaming Driver (Optional)
   Interfaces with the RAM buffers storing raw and processed audio. Useful for future enhancements such as:
   - Saving the audio clip to file
   - Visualizing the waveform
   - Inspecting/debugging buffer content from userspace

4. text_disp_dev – VGA Display Driver
   Enables writing strings and basic visual elements to the on-board VGA display. The hardware display controller reads characters from a video buffer or character RAM to show prompts and progress.

FPGA Hardware Modules

1. Digital Microphone Interface
   Captures PCM data from a digital I²S microphone at 48 kHz and stores the incoming audio stream into on-chip RAM or SDRAM when recording is active.

2. Audio RAM Buffers
   1. One buffer stores raw audio recorded from the microphone
   2. Another stores the pitch-corrected audio for playback

3. Dual-port RAM is used to allow concurrent read/write access

3. Pitch Correction Processor

   Reads audio samples from the raw input buffer and applies pitch detection and correction. It adjusts each note to the nearest musical pitch using techniques like autocorrelation or FFT-based analysis and writes corrected samples to the output buffer. The module signals completion via status flags and optionally updates progress indicators.

4. Audio Output Module (I^2S Transmitter)
   Streams the corrected audio from RAM to the WM8731 audio codec via I^2S. Audio is played through connected headphones or speakers via the board's line-out or headphone jack.

5. FSM Controller
   A finite state machine coordinates the transitions between system states:
   Idle → Recording → Processing → Playback → Idle
   The FSM listens to control flags from the software and updates hardware modules accordingly. It also exposes real-time status signals.

CODEC interface:

We will initialize the I2C module and create the configuration table for the 10 registers we need to program.