Hardware-Accelerated Real-Time Phase Vocoder for Pitch Scaling*

*Title in progress

Embedded Systems Design (CSEE4840)

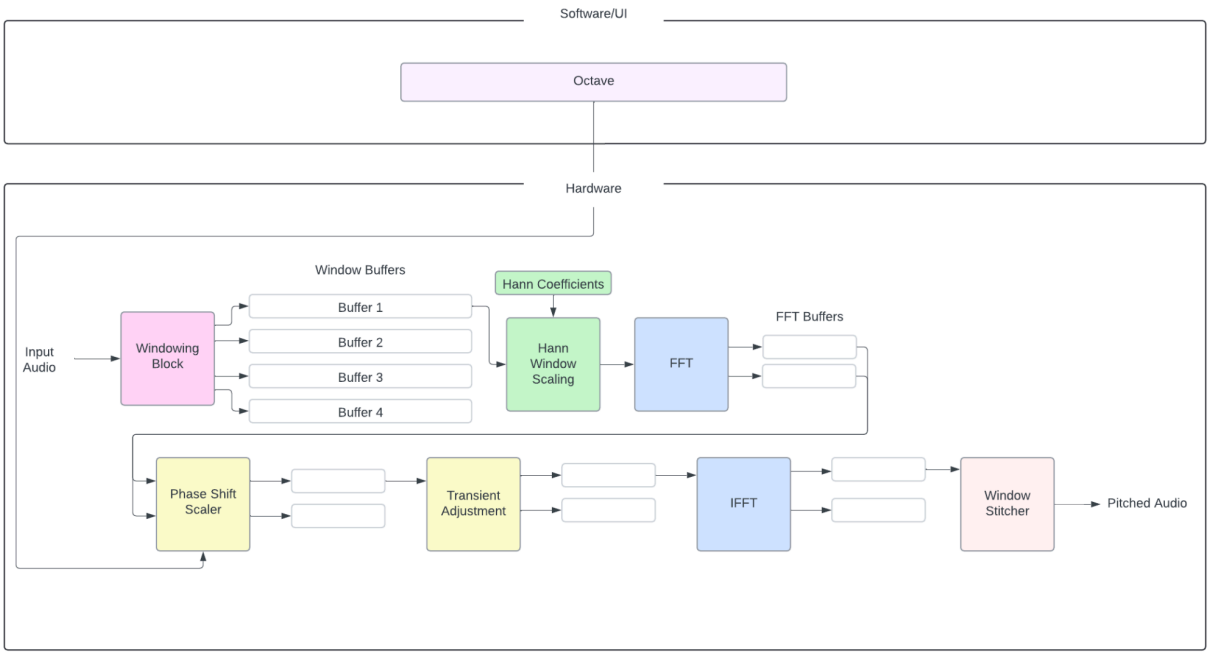
Spring 2024

Sanjay Rajasekharan (sr3764), Maria Rice (mhr2154), Steven Winnick (shw2139)

Introduction

Our project aims to build a phase vocoder for real time pitch scaling primarily through hardware. We utilize an FFT block from the Intel IP Core to perform a Short Term Fourier Transform on the input audio stream. Combined with a series of other hardware-based transformations, the input audio stream will be pitched up/pitched down by a configurable amount, either through software or some simple user interface which is still to be determined.

System Block Diagram



Quad-Output FFT Engine

At the core of our implementation are two Fourier Transform hardware blocks. For these, we've decided to use Intel's Quad-Output FFT Engine rather than attempting to implement our own Fast Fourier Transform module, as we have faith that the good people at Intel were able to implement an efficient FFT module. We decided on the Quad-Output engine because it has a higher throughput than the Single-Output counterpart. The cost of this higher throughput is an increased use of our limited customizable hardware fabric, but based on our initial estimates, we don't expect this to be an issue. We're also using a dual-engine to perform both the initial FFT and the inverse FFT more efficiently than using two separate FFT engines. From the Intel FFT IP Core User Guide, we've surmised that it works in the following way:

- The FFT engine reads complex data samples in parallel from internal memory
- The samples are reordered and processed by a radix-4 butterfly processor to generate complex outputs
- Three complex multipliers perform the necessary twiddle-factor multiplications on the butterfly processor outputs, leveraging the inherent mathematics of radix-4 DIF (Decimation in Frequency) decomposition
- Block-floating point units (BFPU) evaluate the four outputs in parallel to discern the maximum dynamic range of the samples
- The FFT engine discards appropriate LSBs, rounds the values, and reorders the complex outputs before writing them back to internal memory

Data for Quad Output Engine

Window length = 1024 samples (~ 23 ms)
Hop length = 400 samples (~ 9 ms)

The window length was selected to be 1024 based on the convention of using this window length in audio processing. The hop length was calculated using the same ratio used in this Python pitch-shift example, which we've found helpful is understanding the phase vocoder algorithm.

Our implementation maps to the following row in the FFT IP Core user guide.

Device	Parameters			ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
	Type	Length	Engines			M10K	M20K	Primary	Secondary	
Cyclone V	Burst Quad Output	1,024	2	2,501	12	14	--	5,972	196	231

Memory Requirement Estimation

From the table entry above, we see that our chosen FFT engine uses 14 of our 397 available M10K memory blocks.

We will also need 12 buffers to store adjacent windows of samples and frequency-domain vectors. Each of these will hold 1024 16-bit samples, thus requiring two M10K blocks each, putting us up to 36 total blocks, far below the 397 we have available.

Pitch Scaling Algorithm

Input

The algorithm takes as input a stream of audio. In our implementation, these samples are 16-bit fixed point values at a sample rate of 48kHz.

Windowing

The input audio stream will be split into overlapping "windows," or groups of samples. Each window will contain 1024 samples, and be offset by a "hop length" of 400 samples from the previous window.

Hann Window Scaling

The samples in each window will be scaled according to the Hann Function, so the n^{th} sample in a window will be scaled by a factor of $\sin^2(n/1024)$

Short-Time Fourier Transform

A Fourier Transform will be applied to each scaled window. In our implementation, this will be done using the Fast Fourier Transform

Phase Shift Scaling

This is the point in the algorithm where the pitch scaling actually occurs. For each frequency bin in a given window, we will compare the phase to the corresponding bin in the previous window to determine the amount it has shifted by. We will then scale these phase differences by a constant factor, determined by the pitch shift amount. By scaling the phase differences between matching frequency bins in sequential windows by a constant factor, we can essentially change the "speed" at which the composite frequencies in each bin playback, thereby scaling the pitch.

Transient Adjustment

Because we are scaling phase shifts, over time, the shifted phases of frequency bins will become misaligned relative to one another (a problem with what is known as vertical coherence). This is mostly fine, but does cause "transient smearing," which causes transients (fast changes in time-domain wave amplitude at the start of a sound) to sound muffled because phases across frequency bins no longer "line up" at the transient. To account for this, we will perform basic transient detection for each bin by checking for sudden increases in magnitude between subsequent bins. At these points, we will reset phases back to their initially-computed values.

Inverse Fourier Transform

An inverse Fourier Transform will be done to each window to return it to an array of samples.

Window Stitching

The de-transformed phase-adjusted windows of samples are then stitched back together into a main audio stream.

Output

The algorithm outputs a stream of audio. Like the input, in our implementation these will be 16-bit fixed point values at a sample rate of 48kHz.

Milestones (revised)

1. C Simulation for fixed length audio input
2. C Simulation for a real-time audio stream
3. Implement and test each phase of the vocoder algorithm in hardware
4. Successfully perform our full algorithm on an audio stream passed by software, and output to software
5. Connect to external audio input and output streams

References

https://www.youtube.com/watch?v=PjKlMXhxtTM
https://github.com/JentGent/pitch-shift/blob/main/audios.iptheynb
https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling
https://en.wikipedia.org/wiki/Phase_vocoder
https://www.guitarpitchshifter.com/algorithm.html
https://cdrdv2-public.intel.com/667064/ug_fft-683374-667064.pdf&ved=2ahUKEwiR1YTHi5uFAxWCkIkEHUhhBCMQFnoECBMQAQ&usg=AOvVaw1FMRMwxHpeg2l39DPhM4wI