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

What is Pitch Perfect?

OUR GOAL

- A phase vocoder for real time pitch scaling primarily through hardware

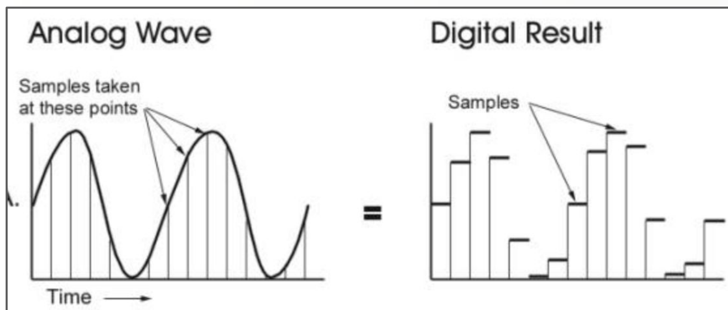
THE TOOLS

- Phase vocoder algorithm
- CORDIC algorithm
- Intel FFT block

Phase Vocoder Algorithm (⅕)

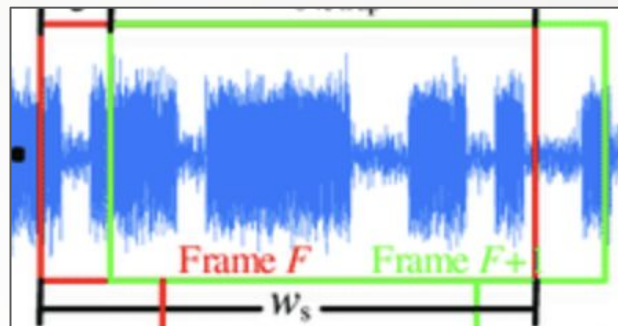
Input

- Audio sample stream
- 16-bit integer values [0, 65,535)
- 48kHz sample rate



Windowing

- Input stream parsed into overlapping "windows"
- 4096 samples each, 1024 "hop length"
- Process each window separately, then recombine at the end



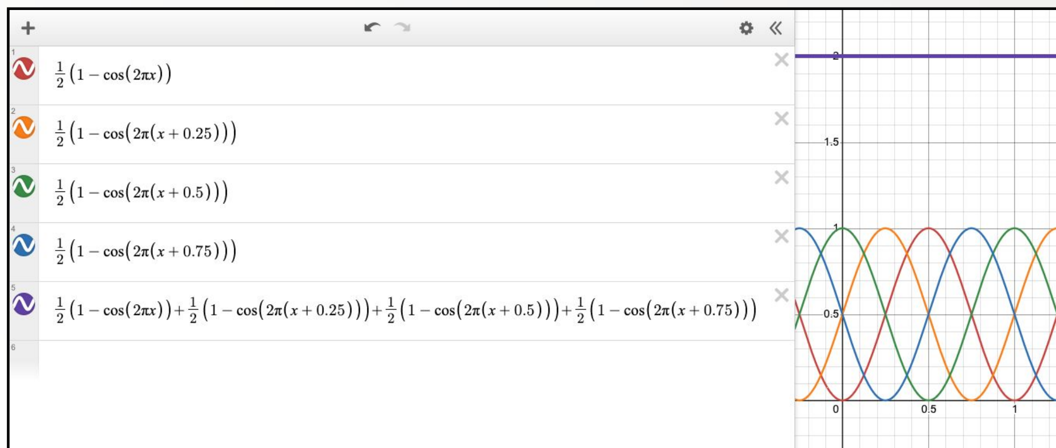
Phase Vocoder Algorithm (2/5)

First Hann Window Scaling

- Samples in each window scaled according to the Hann Function
- Zero-indexed nth sample in a window will be scaled by a factor of $\sin^2(2\pi n/4095)$.

Short-Time Fourier Transform

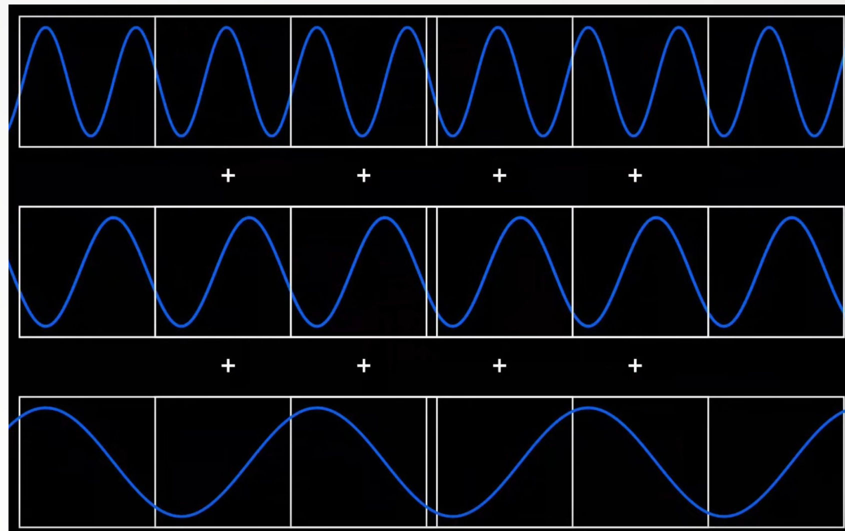
- Fourier transform is applied to each window



Phase Vocoder Algorithm (⅗)

Pitch Shifting, pt. 1

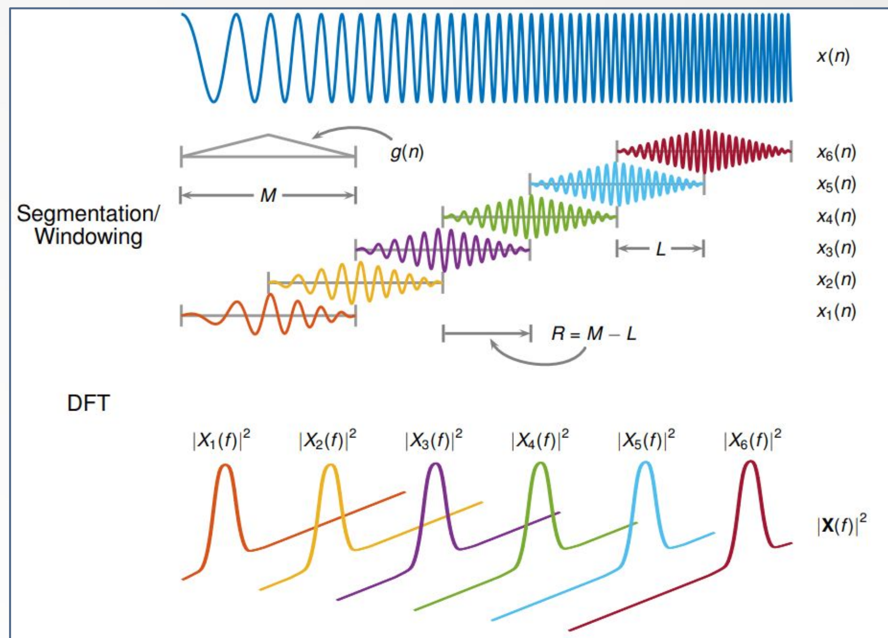
- Each FFT bin has an expected phase difference between frames based on its center frequency
- Observed phase difference gives us a “fractional bin deviation” from center
- Scale fractional bin number based on scale amount (0, 4]



Phase Vocoder Algorithm (3/5)

Pitch Shifting, pt. 2

- Shift phase in new bin by combining expected shift with fractional bin deviation
- Add magnitudes of all inputs in the same output bin



Phase Vocoder Algorithm (4/5)

Inverse Fourier Transform

- An inverse fourier transform is applied to each of the windows to return them to the time domain

Second Hann Window Scaling

- Repeat same Hann Function a second time (to minimize artifacts in synthesized sound)
- Allows for smoother blend between discontinuous time-domain waveforms

.

Phase Vocoder Algorithm (5/5)

Window Stitching

- De-transformed phase-adjusted windows of samples are stitched back together into a main audio stream
- Done by adding half of each sample's windowed value for all 4 windows it appears in.

Output

- Outputs a stream of audio
- In our implementation, these will be signed 16-bit integer values at a sample rate of 48kHz

C-implementation

Python Simulation

- Python simulation that allowed us to easily fine-tune details of our algorithm, such as the window size and hop length, before implementing them in C.

C Real-time Streaming Algorithm

- Reads in samples from standard input and emits the scaled stream to standard output
- Allows it to connect to programs to stream live audio to it and playback its output as live audio

C Fourier Transform

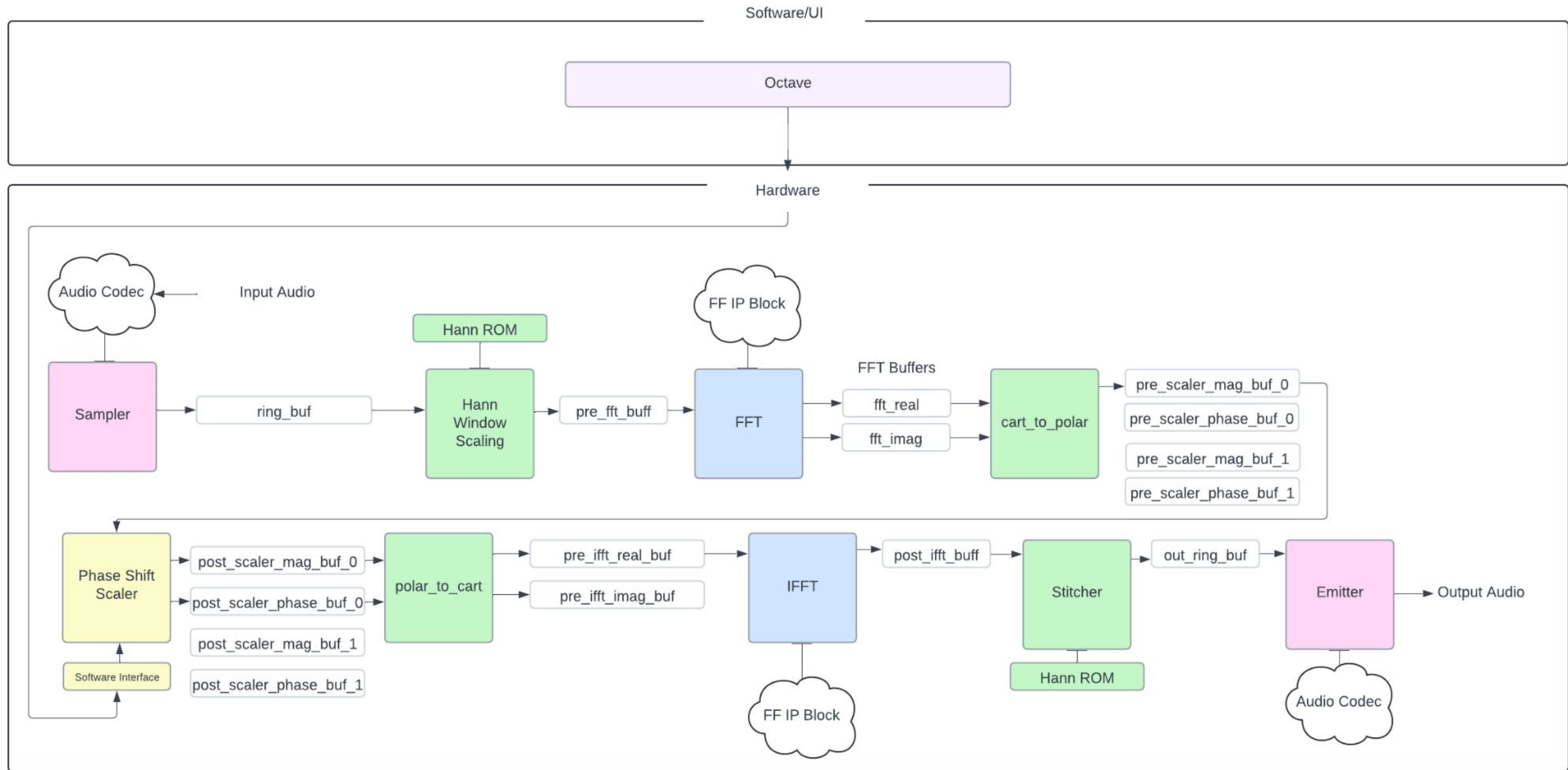
- 3 key functions
  - rearrange()
  - compute()
  - inverse()

Shell Script

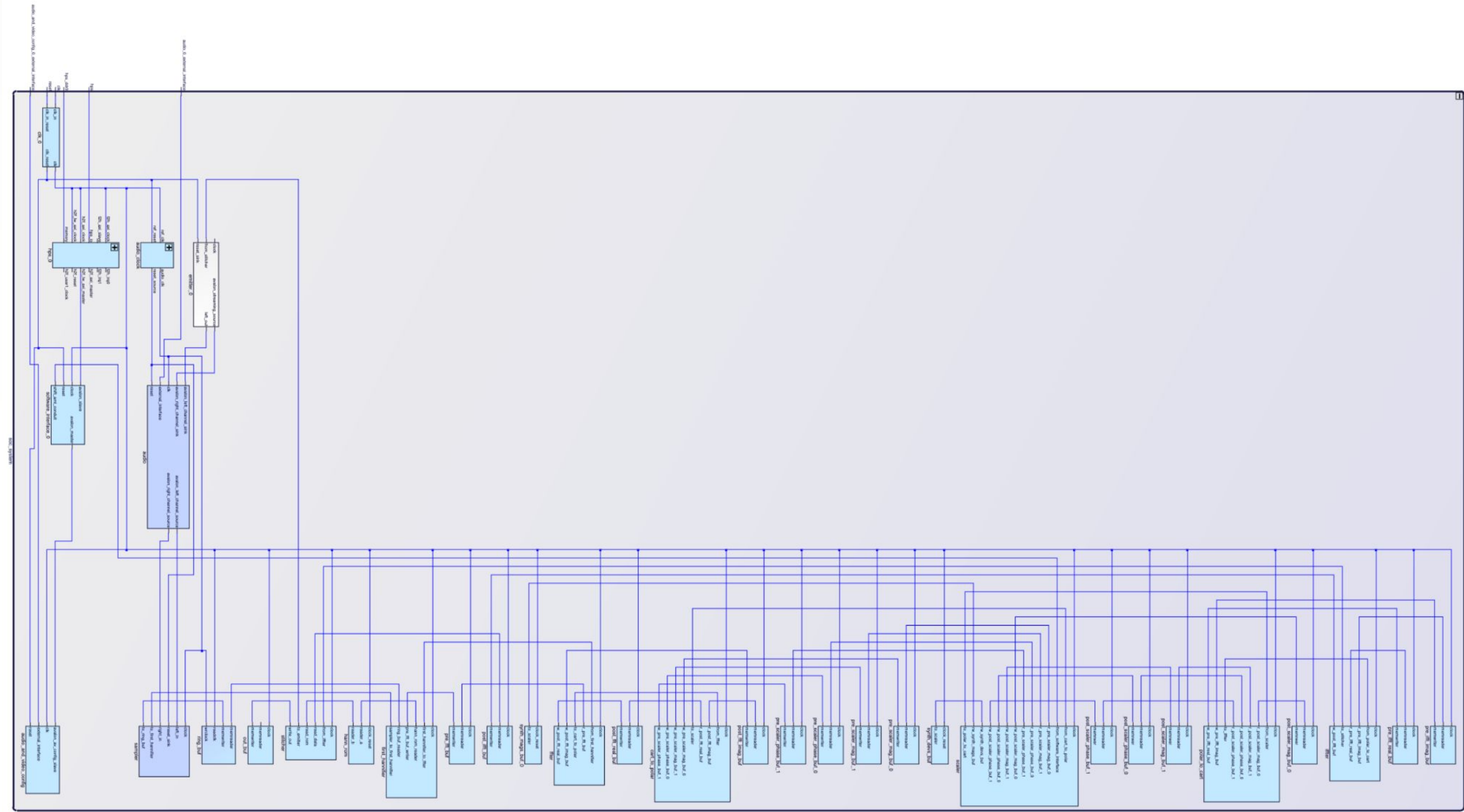
- Simple shell script that creates an end to end pipeline for the software simulation

C-implementation: example

Block Diagram



Hardware Schematic



Hardware Implementation (1/10)

Audio Clock

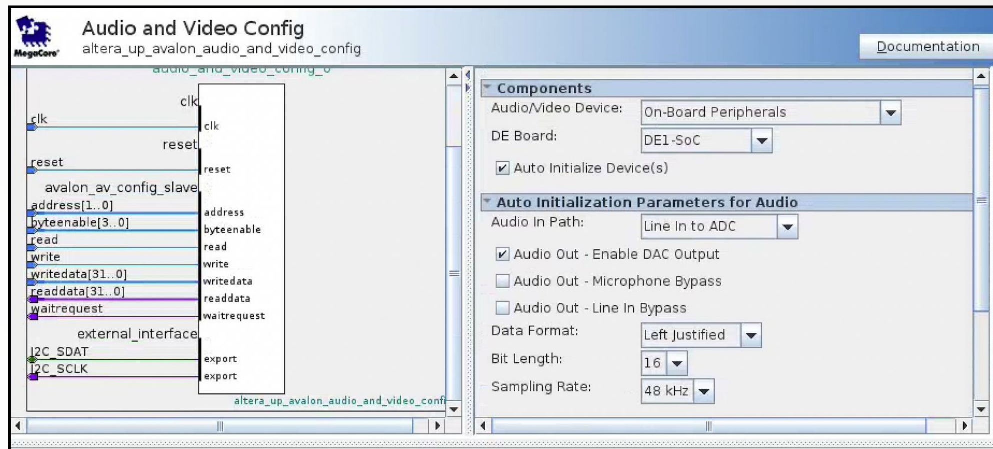
- Clock for all components that interface with audio
- Configured at 12.288 MHz to work with 48 kHz ADC and DAC
- Frequency is efficiently divisible by audio sampling rate (factor of 2^8)

SAMPLING RATE		MCLK FREQUENCY	SAMPLE RATE REGISTER SETTINGS					DIGITAL FILTER TYPE
ADC	DAC							
kHz	kHz	MHz	BOSR	SR3	SR2	SR1	SR0	
48	48	12.288	0 (256fs)	0	0	0	0	1
		18.432	1 (384fs)	0	0	0	0	
48	8	12.288	0 (256fs)	0	0	0	1	1
		18.432	1 (384fs)	0	0	0	1	
8	48	12.288	0 (256fs)	0	0	1	0	1
		18.432	1 (384fs)	0	0	1	0	
8	8	12.288	0 (256fs)	0	0	1	1	1
		18.432	1 (384fs)	0	0	1	1	
32	32	12.288	0 (256fs)	0	1	1	0	1
		18.432	1 (384fs)	0	1	1	0	
96	96	12.288	0 (128fs)	0	1	1	1	2
		18.432	1 (192fs)	0	1	1	1	
44.1	44.1	11.2896	0 (256fs)	1	0	0	0	1
		16.9344	1 (384fs)	1	0	0	0	
44.1	8	11.2896	0 (256fs)	1	0	0	1	1
	(Note 1)	16.9344	1 (384fs)	1	0	0	1	
8	44.1	11.2896	0 (256fs)	1	0	1	0	1
(Note 1)		16.9344	1 (384fs)	1	0	1	0	
8	8	11.2896	0 (256fs)	1	0	1	1	1
(Note 1)	(Note 1)	16.9344	1 (384fs)	1	0	1	1	
88.2	88.2	11.2896	0 (128fs)	1	1	1	1	2
		16.9344	1 (192fs)	1	1	1	1	

Hardware Implementation (2/10)

Audio and Video Config Core

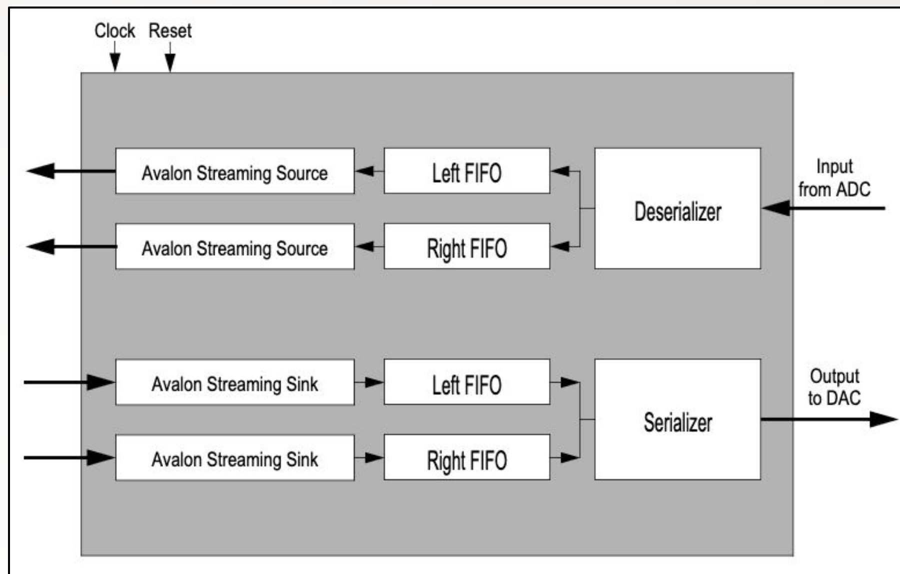
- Initializes Wolfson Audio CODEC
- Communicates via 2-wire I2C serial bus: I2C_SDAT and I2C_SCLK to board's FPGA_I2C_SDAT and FPGA_I2C_SCLK
- 16 bit, left justified, 48kHz
- We interpret as Q'8.8



Hardware Implementation (3/10)

Audio Core

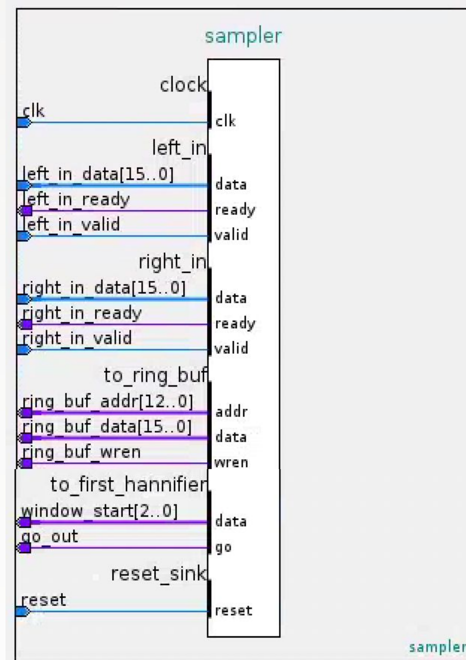
- Interface to Audio Codec
- “Streaming Mode” - we stream values to/from its FIFO buffers, and it handles timing them with the ADC/DAC



Hardware Implementation (4/10)

Sampler

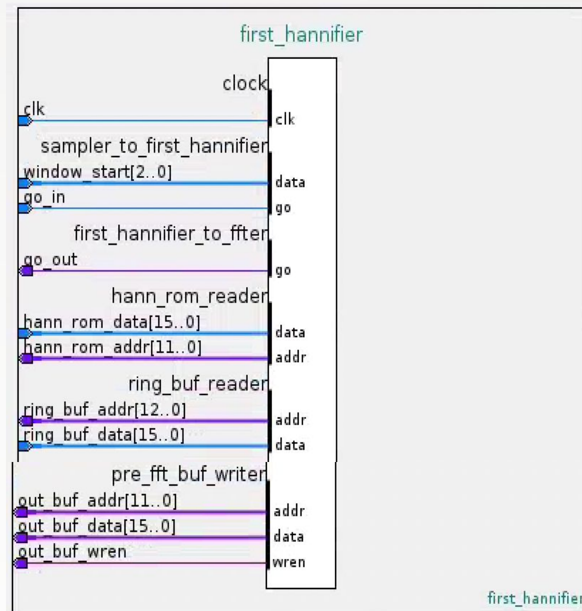
- Read samples of left from_adc to a ring buffer (mono audio)
- Ring buffer holds 1 window + 1 hop of samples (prevent overwriting before use)
- Track start of current window to tell next component
- Every hop, tell next component to start applying Hann Window



Hardware Implementation (5/10)

First Hannifier

- Apply Hann Windowing function to a window of samples
- $\sin^2(2\pi n/4095)$
- Values stored in ROM
- Runs at base clock speed, so ring buffer uses 2 clocks



Hardware Implementation (6/10)

FFT-er / IFFT-er

- Implemented Intel's FFT Engine for an efficient Fourier Transform
- Parameters:
  - See image
- Testing:
  - Initial testing with example file in QuestaSim.
- Wrapper Modules:
  - fft_wrapper.v: Manages input/output signals and instantiates FFT module
  - control_for_fft.v: Generates control signals and handles configuration
  - testbench.v: Verifies functionality

Parameters

System: fft Path: fft_ii_0

FFT
altera_fft_ii

Details
Generate Example Design...

Basic

Transform
Length: 4096
Direction: Bi-directional

I/O
Data Flow: Variable Streaming
Input Order: Natural
Output Order: Natural

Data and Twiddle
Representation: Fixed Point
Data Input Width: 16 bits
Twiddle Width: 16 bits
Data Output Width: 16 bits

Latency Estimates
Calculation Latency: 4096 cycles
Throughput Latency: 8192 cycles

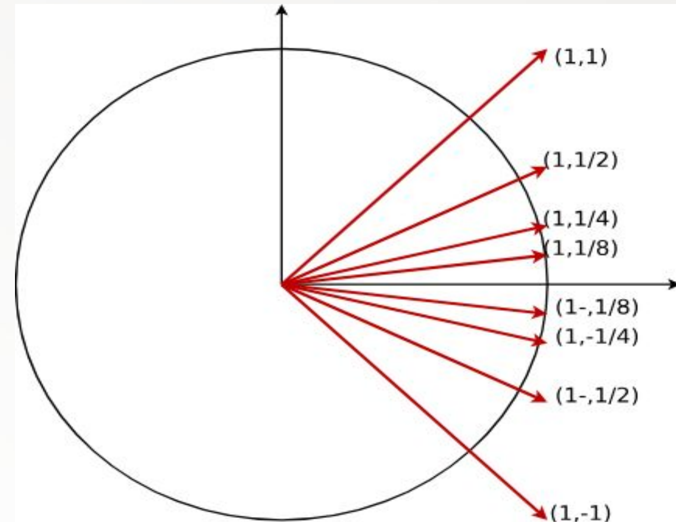
Messages

Type	Path	Message
	1 Info Message	
	fft.fft_ii_0	Radix-2 digit reverse implied

Hardware Implementation (7/10)

Polar Conversion:
CORDIC (Co-ordinate Rational Digital Computer)

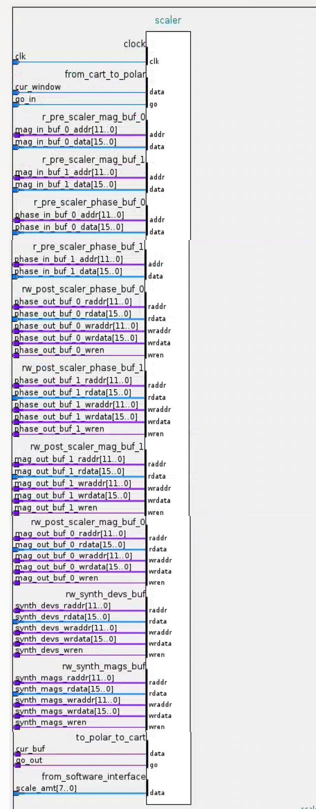
- Cost effective way of computing trig functions on hardware
- Algorithm is based on iteratively rotating a point by an angle theta_k until the point reaches 0
- if $\tan(\text{theta}_k) = 2^{-k}$, then an update rule can be established that doesn't involve multiplication
- $X_{(k+1)} = X_k - Y \ll k$
- $Y_{(k+1)} = X_k + Y \ll k$



Hardware Implementation (8/10)

Scaler

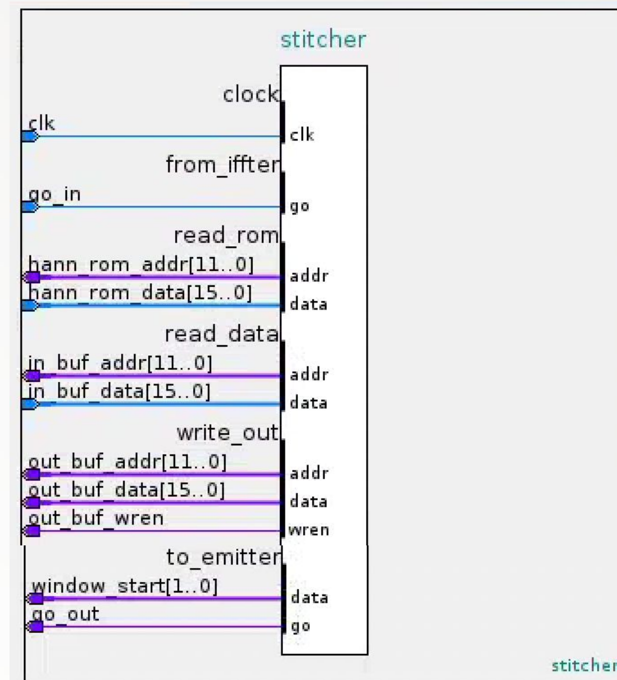
- Performs FFT bin movement described earlier
- State machine: analysis read, analysis write, synthesis
- Custom fixed-point multiplication of different sizes
- Integer rounding of fractional bits
- Avoid modulus



Hardware Implementation (9/10)

Stitcher

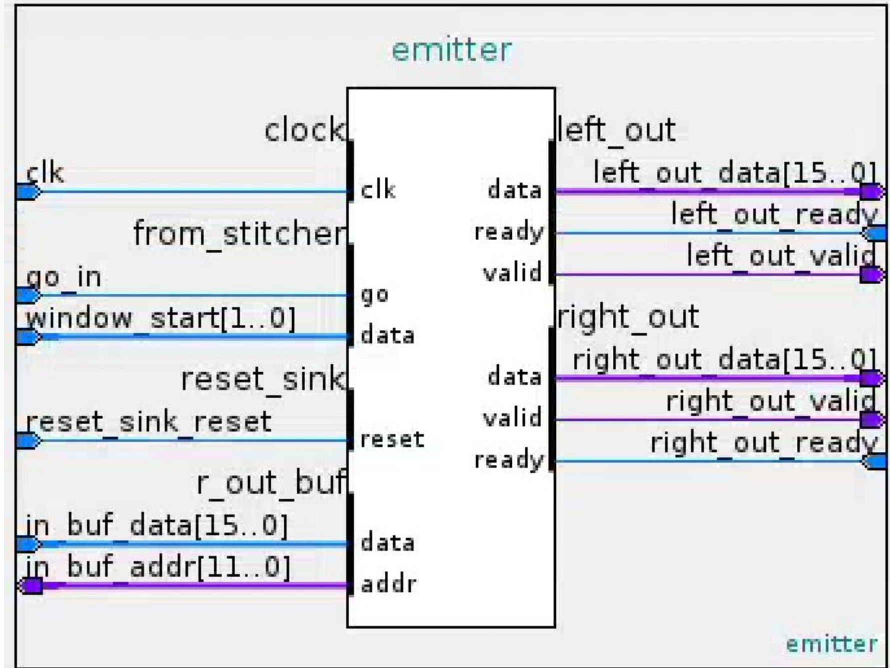
- Apply second Hann Window
- Removes artifacts from discontinuities
- First ¾: add ½ of windowed value
- Last ¼: overwrite with ½ of value



Hardware Implementation (10/10)

Emitter

- Write samples to both to_dac (blocks otherwise)
- One hop at a time
- Runs at audio clock speed, so prior buffer uses 2 clocks



What we learned