Pitch Perfect

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

Sanjay Rajasekharan (sr3764), Maria Rice (mhr2154), Steven Winnick (shw2139) Embedded Systems Design (CSEE4840), Spring 2024

What is Pitch Perfect?





- Phase vocoder algorithm
- CORDIC algorithm
- Intel FFT block

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



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



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]



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



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

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





$\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet$

Block Diagram



$\bullet \bullet \bullet \bullet \bullet \bullet$

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 2⁸)

SAMP	PLING	MCLK FREQUENCY			SAMPLE RATE			DIGITAL
ADC	DAC			REGIS	STER SET	TINGS		TYPE
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	1
48	8	12.288	0 (256fs)	0	0	0	1	1
	1	18.432	1 (384fs)	0	0	0	1	1
8	48	12.288	0 (256fs)	0	0	1	0	1
		18.432	1 (384fs)	0	0	1	0	1
8	8	12.288	0 (256fs)	0	0	1	1	1
		18.432	1 (384fs)	0	0	1	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	1
44.1	44.1	11.2896	0 (256fs)	1	0	0	0	1
		16.9344	1 (384fs)	1	0	0	0	1
44.1	8	11.2896	0 (256fs)	1	0	0	1	1
	(Note 1)	16.9344	1 (384fs)	1	0	0	1	1
8	44.1	11.2896	0 (256fs)	1	0	1	0	1
Note 1)		16.9344	1 (384fs)	1	0	1	0	1
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

sector altera_up_avaion_adulo_and_video_coning aduro_and_video_coning aduro_and_video_coning_o		
urritedata[310] writedata readdata[310] readdata waitrequest waitrequest external_interface export [2C_SDAT export altera_up_avalon_audio_and_video_conf	Audio Out - Microphone Bypass Audio Out - Line In Bypass Data Format: Bit Length: Sampling Rate: 48 kHz ▼	

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
- sin²(2πn/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

ystem: fft Path: fft_ii_0 FFT altera_fft_ii Details Generate Example Design Basic Transform Length: A096 Direction: Bi-directional Field Point Data Flow: Variable Streaming Tota and Twiddle Representation: Fixed Point Data Input Width: 16 bits Twiddle Midth: 16 bits Twiddle Midth: 16 bits Calculation Latency: 4096 cycles Type Path Message f Message f 1 Info Message f ft.fft_ii_0 Radix-2 digit reverse implied	🧯 Paramete	ers 🖾			
Details altera_fft_ii Generate Example Design Basic Image: Construction in the image: Construction in th	ystem: fft	Path: fft_ii	_0		
altera_fft_ii Generate Example Design Basic Transform Length: 4096 ▼ Direction: Bi-directional ▼ Input Order: Natural ▼ Output Order: Natural ▼ Output Order: Natural ▼ Tota and Twiddle Representation: Fixed Point ▼ Data Input Width: 16 ♥ bits Twiddle Width: 16 ♥ bits Data Output Width: 16 ♥ bits Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles Type Path Message 1 Info Message I Info Message 1 Info Message	FFT				Details
Basic Transform Length: 4096 ▼ Direction: Bi-directional ▼ Input 0rder: Natural ▼ Output 0rder: Natural ▼ Data and Twiddle Representation: Fixed Point ▼ Data Input Width: 16 ♥ bits Twiddle Width: 16 ♥ bits Calculation Latency: 4096 _ cycles Throughput Latency: 8192 _ cycles Type Path Message 1 Info Message fft.fft_ii_0 Radix-2 digit reverse implied 	altera_fft_ii				Generate Example Design
Basic Transform Length: 4096 ▼ Direction: Bi-directional ▼ " /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 Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles * Messages -					
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 Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles ● 1 Info Message ● 1 Info Message ● 1 Info Message ● 1 Info Message	Basic				
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 Calculation Latency: 4096 Cycles Throughput Latency: 8192 Cycles	Transfor	rm			
Direction: Bi-directional ▼ VO 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 Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles • Messages ◎ • Messages ◎ • Info Message • 1 Info Message	Length:		4096	· ·	
▼ I/O Data Flow: Variable Streaming ▼ Input Order: Natural ▼ Output Order: Natural ▼ ● Data and Twiddle ● Representation: Fixed Point ▼ Data Input Width: 16 ♥ bits Data Output Width: 16 ♥ bits Data Output Width: 16 ♥ bits Calculation Latency: 4096 Cycles Throughput Latency: Type Path Y ● 1 Info Message ● 1 Info Message ● 1 Info Message	Direction:		Bi-dir	rectional 🔻	
Image: Stream ing Image: Stream in					
Data Flow: Variable Streaming ▼ Input Order: Natural ▼ Output Order: Natural ▼ Output Order: Natural ▼ Data and Twiddle Representation: Fixed Point ▼ Data Input Width: 16 ♥ bits Data Output Width: 16 ♥ bits Data Output Width: 16 ♥ bits Data Output Width: 16 ♥ bits Calculation Latency: 4096 Cycles Throughput Latency: 8192 cycles	* I/O				
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 Data Output Width: 16 ♥ bits Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles * Messages SCalculation Latency: 8192 cycles Type Path Message ¶ 1 Info Message 1 Info Message	Data Flow:	:	Varia	able Streaming 👻	
Output Order: Natural Data and Twiddle Representation: Fixed Point Data Input Width: 16 bits Data Output Width: 16 bits Data Output Width: 16 bits Calculation Latency: 4096 Cycles Throughput Latency: 8192 cycles Type Path Message Info Message Info Message	Input Orde	er:	Natu	ral 🗸	
● Data and Twiddle Representation: Fixed Point Data Input Width: 16 bits Twiddle Width: 16 bits Data Output Width: 16 bits Data Output Width: 16 bits Calculation Latency: 4096 Cycles Throughput Latency: 8192 Cycles Type Path Message Info Message Info Message Info Message	Output Or	der:	Natu	ral 🔻	
Representation: Fixed Point Data Input Width: 16 16 bits Twiddle Width: 16 16 bits Data Output Width: 16 16 bits Calculation Latency: 4096 Cycles Cycles Throughput Latency: 8192 Cycles - Type Path Message - Info Message - Info Message -	T Data an	d Twiddle			
Data Input Width: 16 → bits Twiddle Width: 16 → bits Data Output Width: 16 → bits Latency Estimates Calculation Latency: Calculation Latency: 8192 cycles Throughput Latency: 8192 cycles Type Path Message Y 1 Info Message Info Message Image: Stress of the str	Represent	tation:	Fiver	Roint	
Twiddle Width: 16 bits Data Output Width: 16 bits Latency Estimates Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles	Data Input	t Width	Inc		
IMiddle Width: 16 vits Data Output Width: 16 bits Image: Interpretation of the state	T delle me	- ht	10	bits	
Data Output Width: 16 bits * Latency Estimates Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles - 🗗 🗖 * Messages * - 🖆 🗖 • * Type Path Message • • * 1 Info Message • • • • * 1 Info Message • • • • * 1 Info Message • • • • • 1 Info Message	I WIDDIE WI	lath:	16	bits	
* Latency Estimates Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles * Messages	Data Outp	out Width:	16	bits	
Calculation Latency: 4096 cycles Throughput Latency: 8192 cycles Messages S Cycles Type Path Message 1 Info Message fft.fft_ii_0 Radix-2 digit reverse implied	* Latency	Estimates	;		
Throughput Latency: 8192 cycles	Calculation	n Latency:	4096	cycles	
Messages Type Path Message 1 Info Message fft.fft_ii_0 Radix-2 digit reverse implied	Throughpu	ut Latency:	8192	cycles	
Messages f Type Path Message ? ① 1 Info Message ③ fft.fft_ii_0 Radix-2 digit reverse implied					
Type Path Message ? (1) 1 Info Message (1) (1) fft.fft_ii_0 Radix-2 digit reverse implied	🗧 Message:	s 23			- 🗗 🗆
Ŷ ① 1 Info Message ⑥ fft.fft_ii_0 Radix-2 digit reverse implied	Туре	Path		M	essage
Ifft.fft_ii_0 Radix-2 digit reverse implied	9 🕕 1	Info Messa	ge		
	🚺 ff	ft.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(theta_k) = 2^-k, then an update rule can be established that doesn't involve multiplication
- X_(k+1) = X_k Y << 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

	oconor
clock	10
çik	clk
from cost to polor	L.I.C.
run_cart_to_polar	18
do in	data
do in	go
r_pre_scaler_mag_buf_0	
mag_in_buf_0_addr[110]	addr
mag_in_buf_0_data[150]	data
r pre scaler mag buf 1	
mag in buf 1 addr[11.0]	
mag in buf 1 data[150]	addr
r pro cooler phase buf o	Data
pre_scaler_phase_bui_o	(
phase in buf 0 data[15 0]	addr
phase in build data[15.0]	data
r_pre_scaler_phase_buf_1	
phase_in_buf_1_addr[110]	addr
phase in buf 1 data[150]	data
ny post scaler phase buf 0	
phase out buf 0 raddr[11.0]	
phase out buf 0 rdata[15 0]	raddr
phase out buf 0 wraddr[11.0]	rdata
phase out buf 0 wrdata(150)	wraddr
phase out buf 0 wren	wrdata
an and and a share but I	wren
rw_post_scaler_phase_bui_t	
phase out bur 1 raddr[11.0]	raddr
phase out but 1 ruada[13.0]	rdata
phase out but 1 wraddr[11.0]	wraddr
nhase out buf 1 wren	wrdata
	wren
rw_post_scaler_mag_buf_1	
mag_out_buf_1_raddr[11.0]	raddr
mag out buf 1 rdata[150]	rdata
mag_out_buf_1_wraddr[11.0]	wraddr
mag out but 1 wrdata[150]	wrdata
mag_out_pur_1_wren	wren
rw_post_scaler_mag_buf_0	
mag out buf 0 raddr[11.0]	raddr
mag out our o rdata[15.0]	rdata
man out buf 0 wrdata[15_0]	wraddr
mag out buf 0 wren	wrdata
nu cunth dava hud	Iwren
rw_synth_devs_but	(
south days readd[11.0]	raddr
synth devs wraddrill 01	rdata
synth devs wrdata[150]	wraddr
synth_devs_wren	wrdata
py susth mags buf	Incen
soft made raddr[1]_01	
synth mags rdata[15.0]	raddr
synth mags wraddr[110]	rdata
synth mags wrdata[150]	wraddr
synth_mags_wren	wren
to polar to cart	
cur_buf	data
co_out	90
from software interface	
scale amt(7.0)	
	data

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