TNShazam

An FPGA Based Song Recognizer

Eitan Kaplan
Jose Rubianes
Tomin Perea-Chamblee

Overview

- Shazam recognizes songs with a fingerprinting algorithm that involves taking an STFT of the songs and then processing the STFT to create a hashable, reduced representation of the song
- Our goal was to do the preprocessing (mainly the STFT) for that algorithm in hardware, and then do the remainder of the algorithm, including creating the database of hashed songs in software
- In addition to implementing a hardware STFT accelerator, and implementing the software portion of the algorithm, our project also required hooking up the board's microphone and configuring it with the I2C bus, as well as interfacing with the Avalon bus

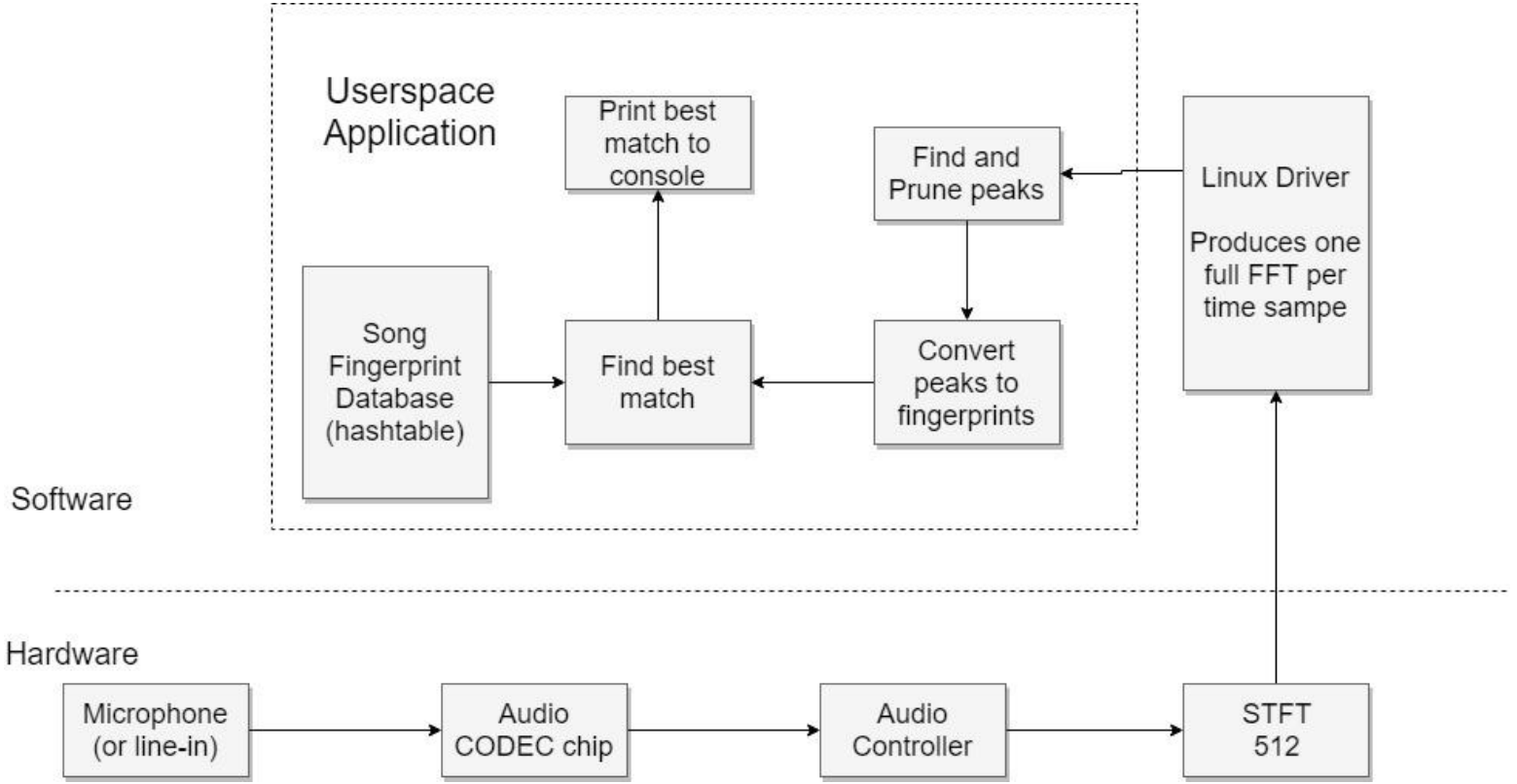
The Shazam Algorithm: Creating the Database

1. Take STFT of audio
2. Find amplitude peaks of the STFT
3. Prune amplitude peaks to retain only the most significant peaks. This pruning step makes the algorithm robust against noise. At this point, amplitude data can be discarded, and what remains is a “constellation map”
4. Create pairs of peaks that are in close time-proximity to each other. Form a fingerprint from each pair by concatenating the frequency of each peak, and the time delta between them
5. Put each fingerprint in a hashtable where the key is the fingerprint and the value is the song name and the absolute time of the first peak in the pair.

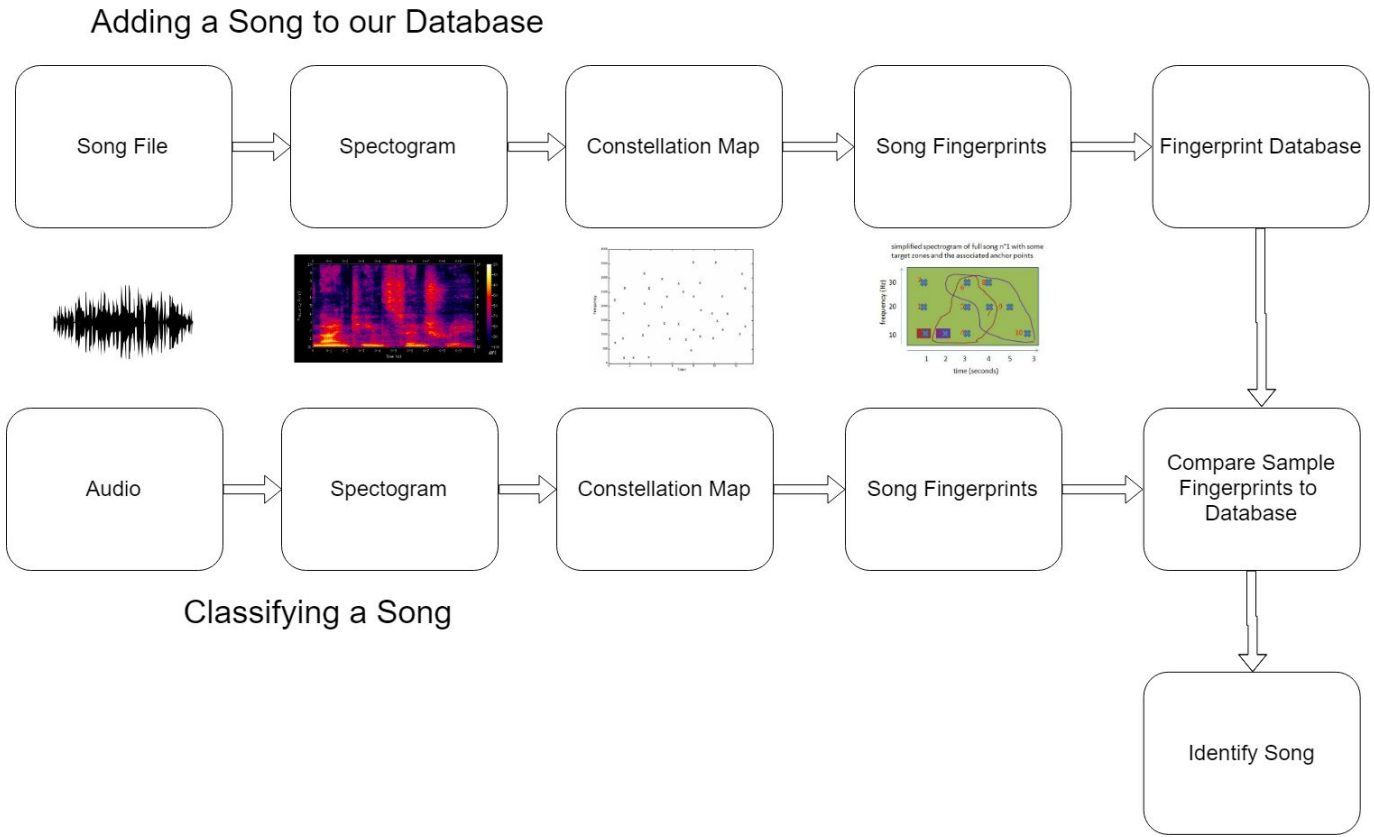
The Shazam Algorithm: Identifying Song Samples

1. Repeat steps 1-4 of the databasing algorithm (STFT, peaks, pruning, and fingerprints) on the incoming unknown song sample
2. Look up each resulting fingerprint in the database
3. For each song in the database, count the number of fingerprint matches that share both the first absolute time in the databased song and the absolute time in the unknown sample
4. Choose the song with the highest number of such matches (in case of a tie, choose the databased song with the smaller number of entries in the database).

System Architecture



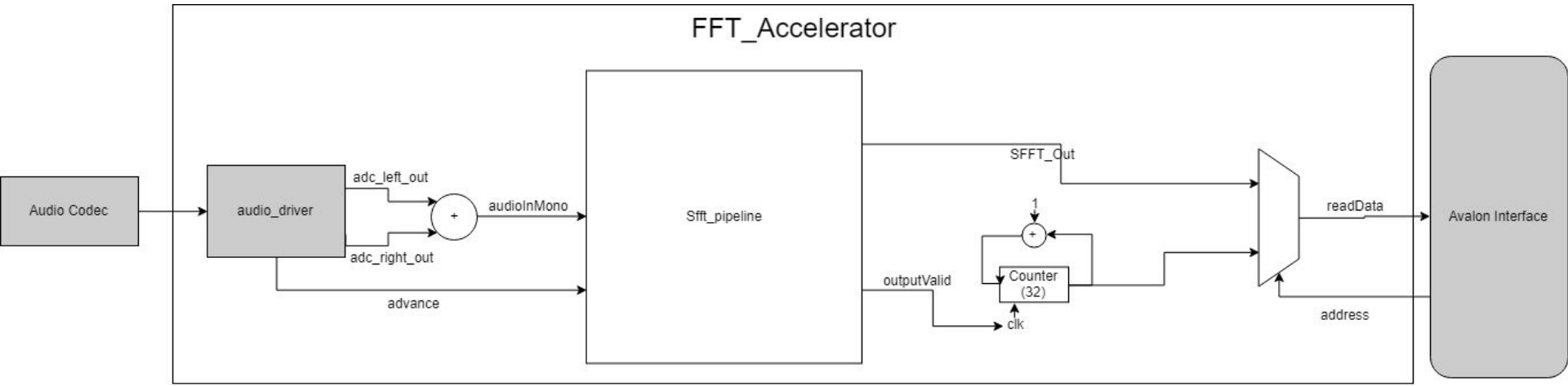
How it works...


Adding a Song to our Database
Song File
Spectogram
Constellation Map
Song Fingerprints
Fingerprint Database
simplified spectrogram of full song n°1 with some target zones and the associated anchor points
Audio
Spectogram
Constellation Map
Song Fingerprints
Compare Sample Fingerprints to Database
Classifying a Song
Identify Song

Hardware Overview

1. Our design calculates the STFT of the incoming audio.
2. After being passed through the audio codec and the audio driver, the input to the Sfft module is a mono representation of the audio and the ‘advance’ signal which switches on the sampling of the input signal.
3. The Sfft module in turn calculate the STFT by implementing the Cooley-Tukey algorithm, so as to avoid large matrix multiplications.
4. The base hardware block needed to accomplish this would be a module (ButterflyModule) that performs a single radix-2 FFT calculation. Then, this module can be replicated and pipelined to calculate any N-point FFT as shown below (next slide).

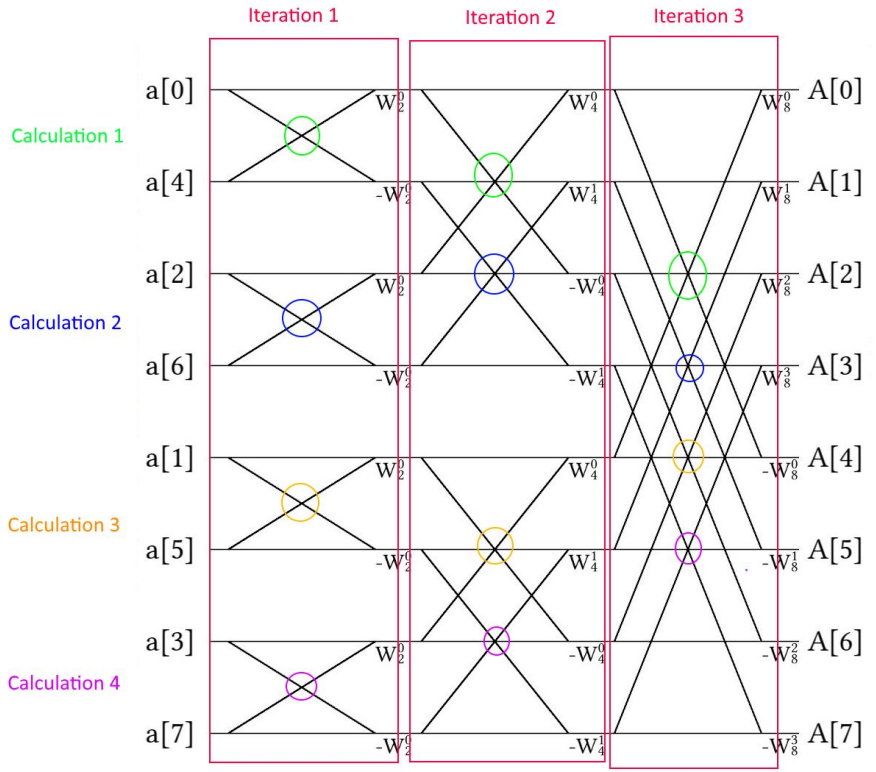
Hardware: FFT Accelerator Module Block Diagram



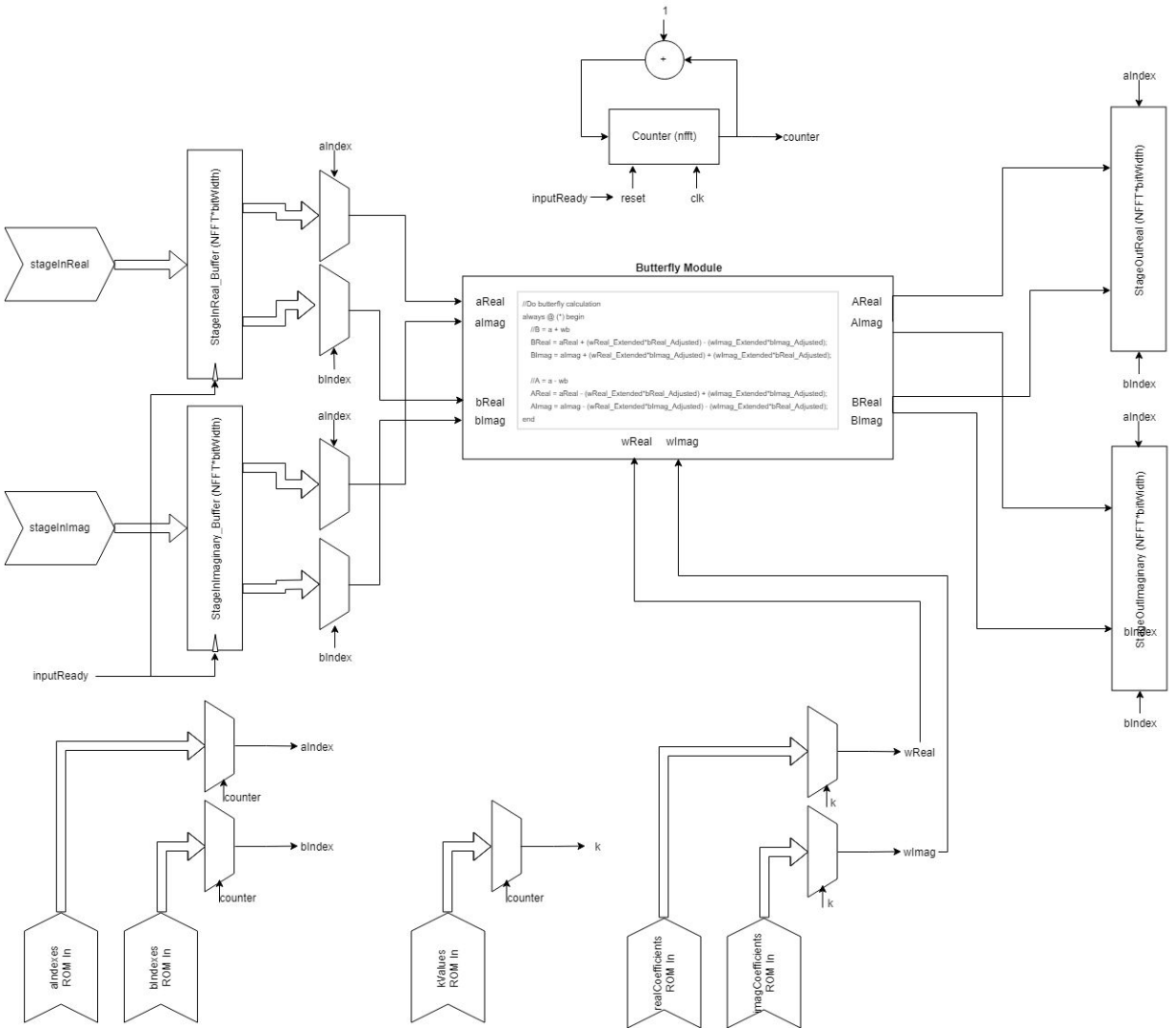
Iterations = log2(N)

Butterfly Calculations

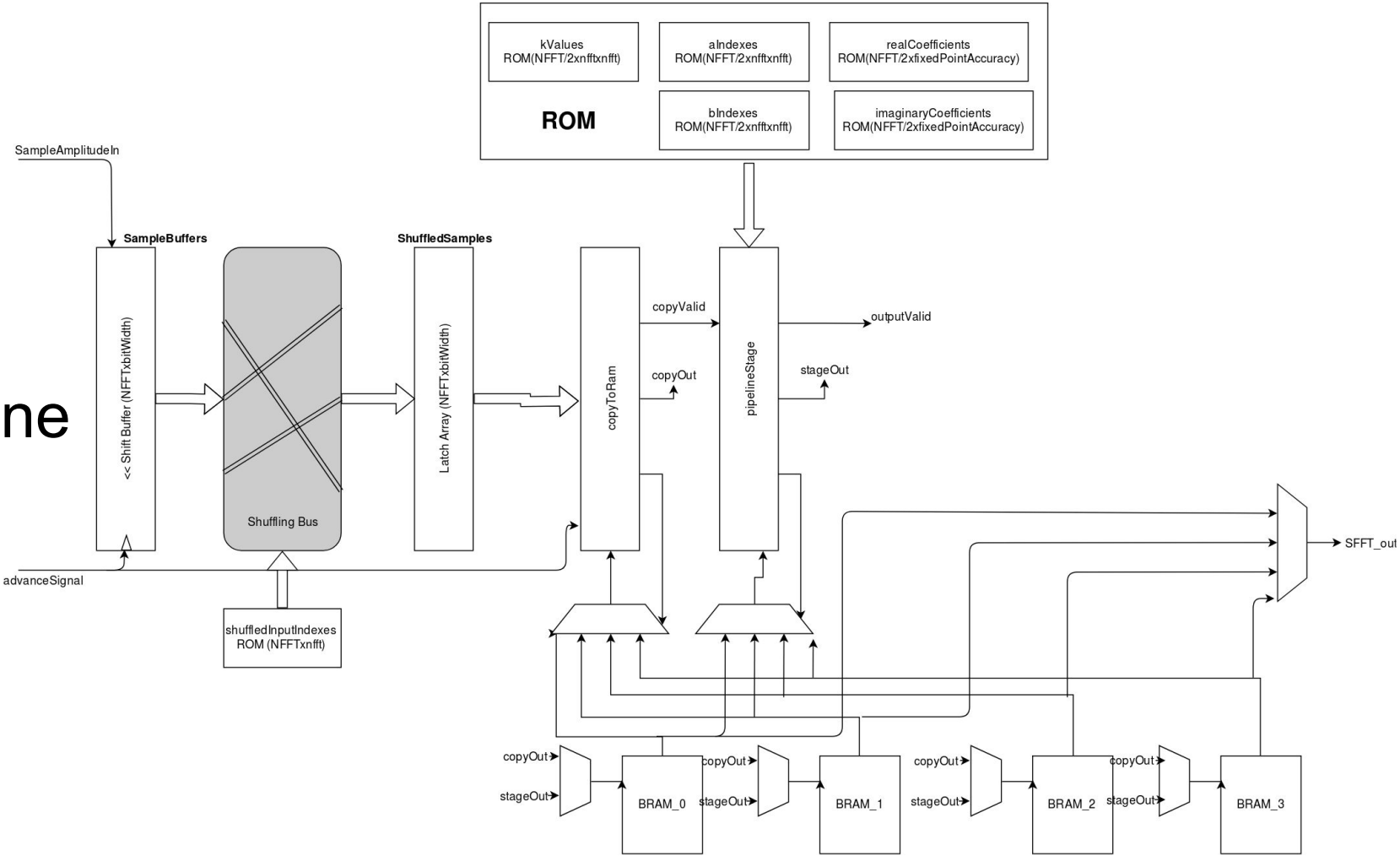
Butterflies Per Iteration = N/2



Sfft Stage


1
+
Counter (nfft)
counter
inputReady
reset
clk
aIndex
bIndex
stageInReal
StageInReal_Buffer (NFFT*bitWidth)
stageInImag
StageInImaginary_Buffer (NFFT*bitWidth)
inputReady
Butterfly Module
aReal
aImag
bReal
bImag
wReal
wImag
AReal
AImag
BReal
BImag
//Do butterfly calculation
always @ (*) begin
//B = a + wb
BReal = aReal + (wReal_Extended*bReal_Adjusted) - (wImag_Extended*bImag_Adjusted);
BImag = aImag + (wReal_Extended*bImag_Adjusted) + (wImag_Extended*bReal_Adjusted);
//A = a - wb
AReal = aReal - (wReal_Extended*bReal_Adjusted) + (wImag_Extended*bImag_Adjusted);
AImag = aImag - (wReal_Extended*bImag_Adjusted) - (wImag_Extended*bReal_Adjusted);
end
StageOutReal (NFFT*bitWidth)
StageOutImaginary (NFFT*bitWidth)
aIndex
bIndex
counter
k
wReal
wImag
aIndexes ROM In
bIndexes ROM In
kValues ROM In
realCoefficients ROM In
imagCoefficients ROM In

Sfft Pipeline

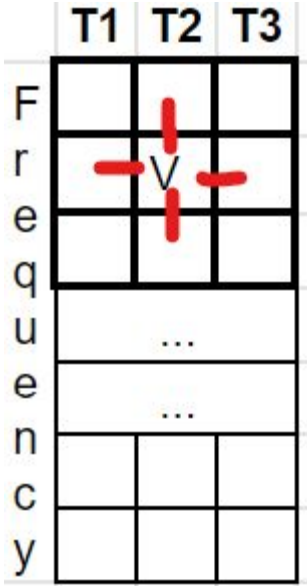


Hardware / Software Interface

1. Data produced by hardware module is stored in a memory mapped buffer.
2. The data that is read from the bus is the value of a counter that tallies the number of STFT results thus far computed is placed in the buffer.
3. The driver reads from the buffer to retrieve the data. Once the driver begins reading, the buffer will not be overwritten until the data retrieval is finished.
4. However, the hardware makes no guarantee that the next sample seen by the driver will be the next sample sequentially (i.e., if the driver is too slow retrieving data, some data may be dropped).
5. Our algorithm is robust against occasional missed sample (as long as timing data is preserved -- hence the need for the sample counter).

Software - Amplitudinal Peak Finding

- The STFT data comes in from the accelerator as a two-dimensional array (see diagram to the right).
- In software, we divide the 256 frequencies into 6 logarithmic bins. In each bin, we find all peak candidates -- all entries in the array whose amplitudes are greater than all of their four neighbors. Then, for each bin, at each time slice, we keep only the peak with the highest amplitude of the peak candidates.



Software - Pruning and Fingerprinting

- At this point peaks are represented as 3-tuples consisting of the time, frequency, and amplitude.
- In order to further reduce the number of peaks we use, we prune in the following manner. We look at a time chunk time slices at a time, and take the average and the standard deviation of all peak amplitudes in that time chunk. Then we throw away all peaks with amplitudes that are less than the average plus a the standard deviation times a coefficient.
- Fingerprints, as aforementioned, amount to pairs of these pruned peaks.
- Checking for equality between fingerprints is quick and computationally inexpensive

Challenges and Lessons Learned

Know your specifications

1. Specifying memory in an interpretable way (lest the *synthesizer* punish you)
2. Know hardware default input sampling frequency.
3. Certain considerations were based on erroneous assumptions about bus throughput

Acknowledgements

- Cohen, Danny. (1977). Simplified control of FFT hardware. Acoustics, Speech and Signal Processing, IEEE Transactions on. 24. 577 - 579. 10.1109/TASSP.1976.1162854.
- Christophe. How does Shazam work. http://coding-geek.com/how-shazam-works/
- Froitzheim, Simon. (2017). A Short Introduction to Audio Fingerprinting with a Focus on Shazam. http://hpac.rwth-aachen.de/teaching/sem-mus-17/Reports/Froitzheim.pdf
- Intel (for Quartus)
- C++ (for pre-implemented containers)
- Martha! Thanks for all your help!