Polyphonic Music Transcription on an FPGA

Shuyi Chen
Lizi George
Kelly Ran
May 18, 2013

1 PROJECT OBJECTIVE

Our objective is to identify the tones in polyphonic music (i.e. music with multiple pitches playing simultaneously) in realtime. Monophonic pitch detection has good existing solutions, but polyphonic pitch detection implementations can have high error rates and slow processing times.

We aim to achieve an implementation that can be used in realtime transcription of music. In conjunction with a beat detection module, our pitch detection module would be able to engrave live or recorded music. We would be able to transcribe an improvised jazz session or engrave Youtube audio into sheet music for practicing. An FPGA-based solution is perfect for speeding up our pitch detection algorithm and achieving realtime results.

Our input music is sampled at 22.05kHz and chunked into frames of 2048 samples. Each sample is represented as a FixedPoint number with 16 integer bits and 24 fractional bits. Thus, the input bit rate is 882000 bits per second and our target frame rate is 11 frames per second. For each frame, our system outputs the frequencies of the detected notes, with a maximum of 4 notes detected simultaneously. The pitch frequencies are represented as FixedPoints with 16 integer bits and 16 fractional bits. Thus, the output bit rate is 1760 bits per second.

$$\text{frame rate} = \left(\frac{22.05\ \text{Ksamples/s}}{2048\ \text{samples/frame}}\right) \tag{1}$$

$$\text{bit rate} = \left(22.05\ \text{Ksamples/s}\right)\left(16 + 24\ \text{bits/sample}\right) \tag{2}$$

$$\text{output bit rate} = (11\ \text{frames/s})(5\ \text{pitch outputs/frame})(16 + 16\ \text{bits/output}) \tag{3}$$

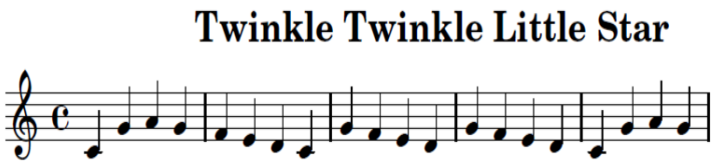

Figure 1: Sheet music for monophonic tune Twinkle Twinkle Little Star produced by Lilypond engraving editor. This output was the result of running the implementation of our algorithm in Matlab on a .wav file of a piano playing the tune. Rhythmic differentiation is not shown as the algorithm does not employ beat detection.

2 Background

Musicians, especially composers and those who arrange music must develop the ability to listen to music and write down or "transcribe" onto sheet music what they hear. However, this technique requires significant music theory knowledge and ear training, and becomes much more complex when multiple instruments and chords are introduced. It is especially difficult for those with no musical training to accomplish this task, which is becoming an increasingly desirable task as amateur musicians wish to play their favorite songs without having to buy sheet music that is often unavailable.

One might also want the ability to transcribe music in realtime for live-composition. In a test, commercially available music transcription software takes about 30 seconds to process a song of about 30 seconds and this cannot be done in real-time. With our Matlab implementation, it takes about 2 seconds to process a song of 30 seconds and 3-4 seconds to process a minute long file. For large audio files or small time-slice size for processing, this could become quite time consuming.

We define pitch as the fundamental frequency of a periodic sound. Humans perceive sounds on a logarithmic scale and recognize canonical frequencies in the Western music tradition. Research is currently ongoing to develop algorithms for polyphonic pitch detection. This goal is hindered by high computational loads, timbral effects, and inconsistent detection accuracy over large frequency ranges.

Many pitch detection algorithms use the Fourier Transform to analyze frequency components of audio. Detecting pitch in sounds can be complicated by the effects of timbre, or spectral composition of the sounds. To accomplish good timbre rejection, we use a technique called "spectral whitening" which is described later in this paper.

Some of the most current literature has blended different pitch detection algorithms to take advantage of their strengths and weaknesses. For example, researcher John Thomas combined algorithms that used the Auto-Correlation Function (ACF) and the Fast Fourier Transform (FFT). The resulting algorithm had the ACF's good accuracy at lower frequencies and the FFT's good accuracy at higher frequencies.

A first step to creating such a combined algorithm is to start with a single-method algorithm. From methods that used ACT, FFT, and non-negative matrix factorization (NMF), we chose the FFT-based Klapuri algorithm because of its superior accuracy when detecting 1-4 pitches in polyphonic music.

The Klapuri algorithm begins by processing each music input frame with Hanning windowing, zero-padding, and spectral whitening. Then, it uses a metric called Salience to estimate the strongest pitch in the frame. Salience is calculated as the sum of the FFT contributions from a pitch and up to 20 of its harmonics. Using Salience, the algorithm performs a modified binary search and returns the detected pitch. If polyphonic detection is desired, the Klapuri subtracts this pitch from the spectrum and repeats the process until up to four pitches per time-slice are detected.

3 Benefits of FPGA

The Klapuri algorithm is a good candidate to be implemented on a FPGA because many of the subcomponents of the algorithm can be done in parallel. For example, in the whitening module, there are 30 different coefficients to be computed from a set of 2048 input data points; if this computation was done serially, the total throughput would be 1/30 of the same computation done in parallel. The ML605 has enough DSP slices to provide high speed parallel computations needed for rapid input into the linear interpolation module.

On the FPGA, the streaming FFT module takes up the most real estate, and so having more than one module is not practical. However, by duplicating the computation modules (e.g. Absolute value) downstream from the FFT and staggering the FFT outputs, we can get a better throughput. This double buffering technique can also be implemented in software, but on an FPGA is much easier.

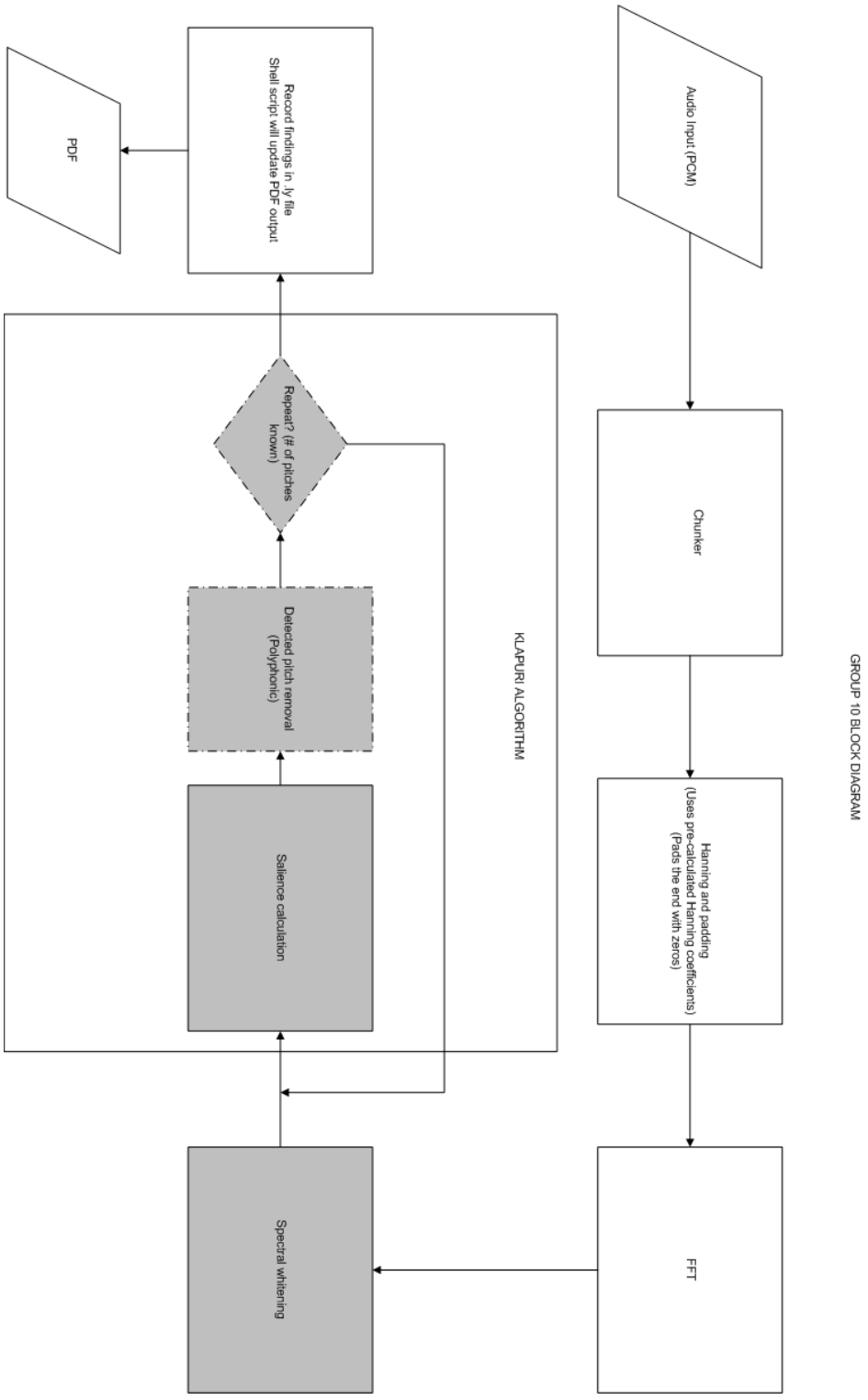

Figure 2: System Block Diagram.

4 High-Level Design and Test Plan

Our test bench was based on John Thomas' reference code in his 2012 paper. We ran the Klapuri algorithm in MATLAB, with .wav file inputs that we found on the Web (Twinkle Twinkle Little Star and a Bach chorale). With this system, we verified that the algorithm can identify the monophones in Twinkle and the polyphones in Bach Chorale.

Our Bluespec system starts when we feed a .pcm music file into the SceMi interface. The music's tempo and degree of polyphony are also provided via SceMi. The input music is chosen for its constant beat length, as our project does not include beat detection.

The pitch detection hardware chunks the data, windows and pads the data frames, applies whitening, generates the frame's FFT, and finds the strongest frequencies. These pitch frequencies, represented as FixedPoint numbers, are sent back to software.

Then, the software component matches the frequencies with actual pitches (i.e. 261.626Hz → C4). These pitches are written into a text file, and a shell script handles sheet music generation: the script sends the text file to LilyPond, which produces a sheet music PDF. The script reloads GhostViewer so that the displayed PDF always shows the cumulative engraved music.

Lilypond is a music engraving program that takes a text file similar to that of LaTeX and produces a professional-looking pdf of sheet music. This program is terminal-based, free and very simple to use with no GUI and seemed perfect for our application.

5 Microarchitectural Description

The time domain slices of audio samples coming from the .PCM file are first passed through the Hanning window function. The window function's purpose is to attenuate the signal at the edges of the slice and make sure the fourier transform of the signal does not extend to infinity (similar to sync function). 2048 Hanning coefficients were generated in MATLAB and loaded into a big look up table for multiplication with the incoming data points. The lookup table was later replaced with BRAM to decrease LUT utilization. This module takes in the data samples as signed integers and multiplies them with appropriate Hanning coefficients and output to the FFT module.

$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \tag{4}$$

The cube root module was based partially on the square root module we were given. The goal was to use an efficient algorithm that would not take up much hardware on the FPGA. The algorithm

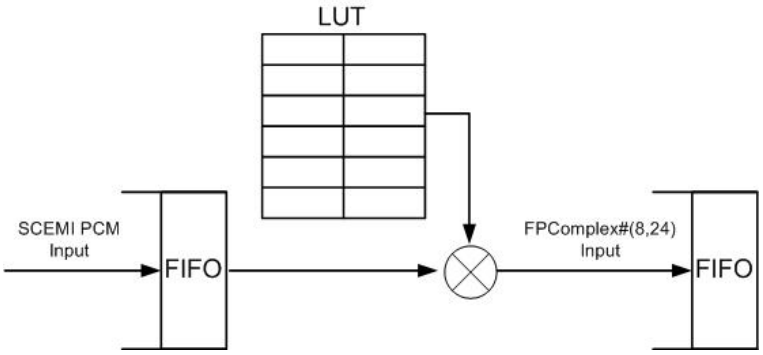

Figure 3: Hanning window stage microarchitecture.

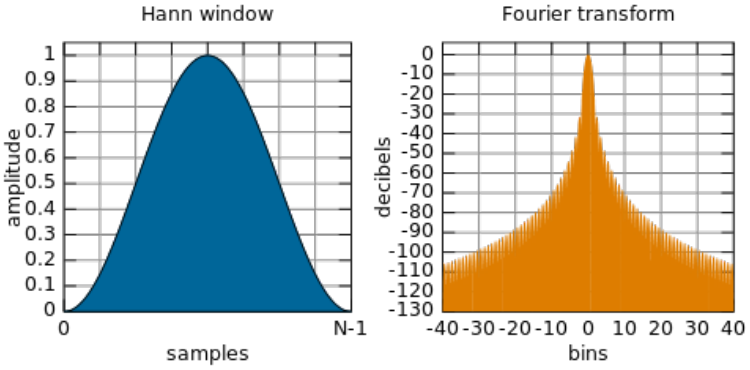

Figure 4: Hanning window stage microarchitecture.

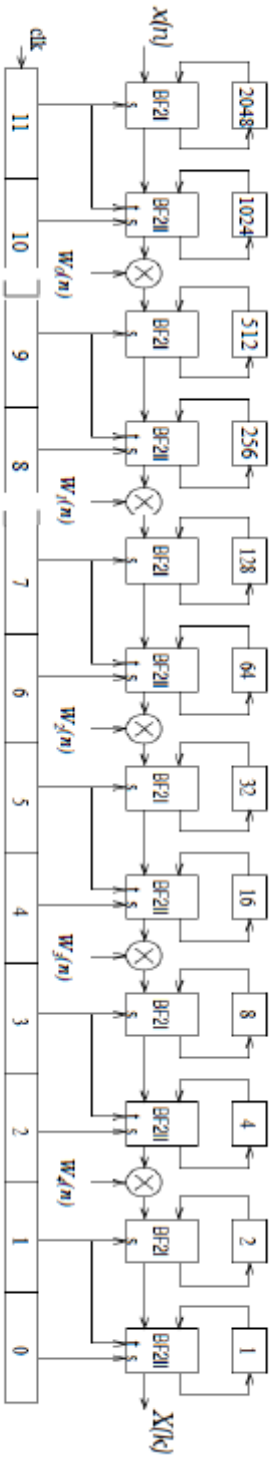

Figure 5: 4096-pt streaming FFT implementation by Abhinav Agarwal of MIT CSAIL.

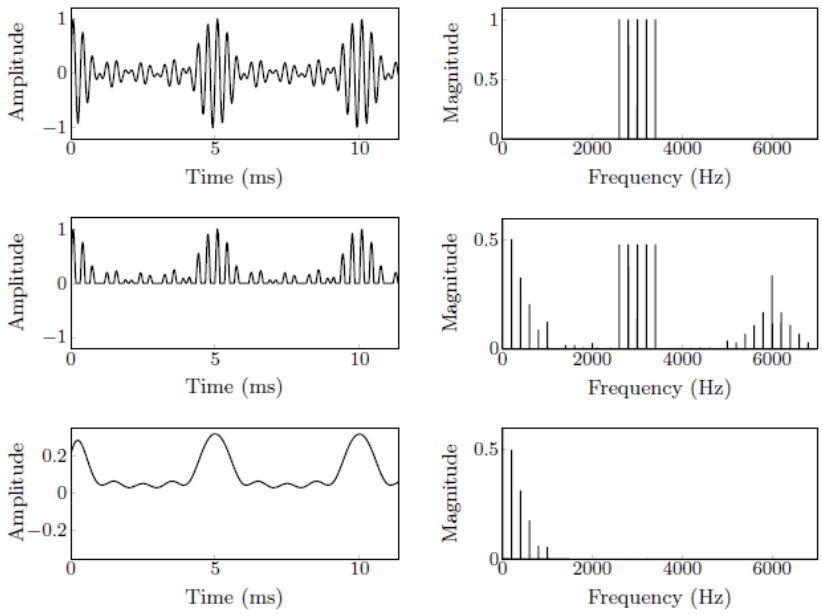

Figure 6: Effects of half-wave rectification at the output of the absolute value module.

first considered was the iterative Newton-Raphson search method, however this algorithm is quite computationally expensive because it involves picking "likely" numbers and cubing each one until the original value is reached. The convergence of this algorithm is at least quadratic which intuitively means that the number of correct digits roughly at least doubles in every step.

The cube root module that we implemented was based on an algorithm published in a 1981 paper detailing algorithms for square and cube roots. It uses clever bitwise operations to cut down the amount of computation needed for this operation. This turns a potentially expensive block into one of manageable size.

In order for the algorithm to effectively detect pitch with different sound sources, we need to isolate the fundamental frequency while attenuate the harmonics. The spectral whitening module implements equation to suppress the timbre of the sound in the sample slice. The operation is done in the frequency domain, after the sound has been zero padded and went through FFT and absolute value.

The whitening module is made up of several interconnected components, and each component performs an operation on the input and passes the result onto the next one. As the input leaves the input FIFO to get squared by the squaring component, a copy of it is stored in a register file. Once every value has gone through all the component stages, the whitening coefficient can

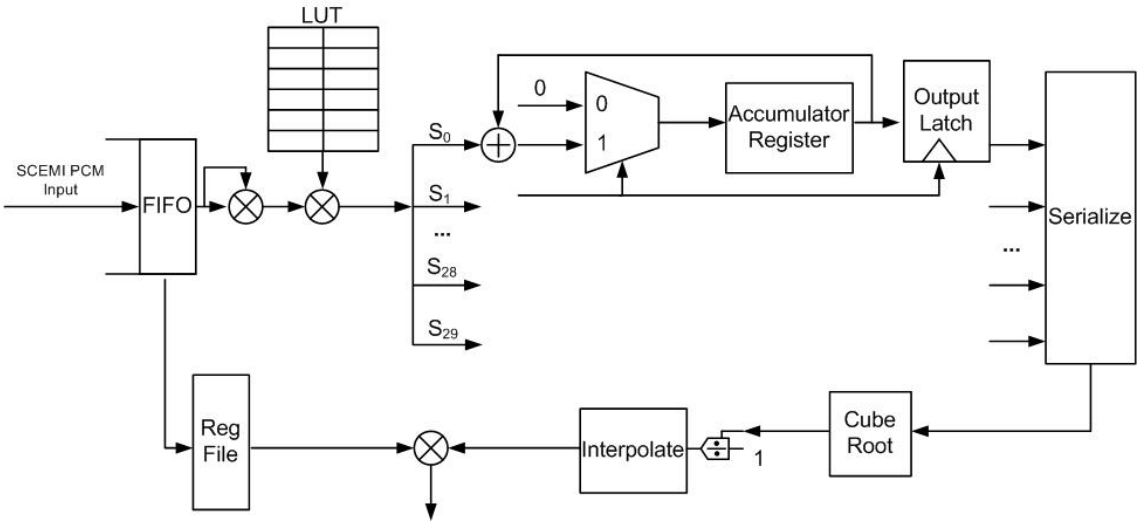

Figure 7: Spectral whitening stage microarchitecture.

finally be calculated and multiplied to the original inputs in the register file. The interpolation submodule receives 32 x,y pairs and outputs 2048 linearly interpolated x,y pairs. The y values are the whitening coefficients.

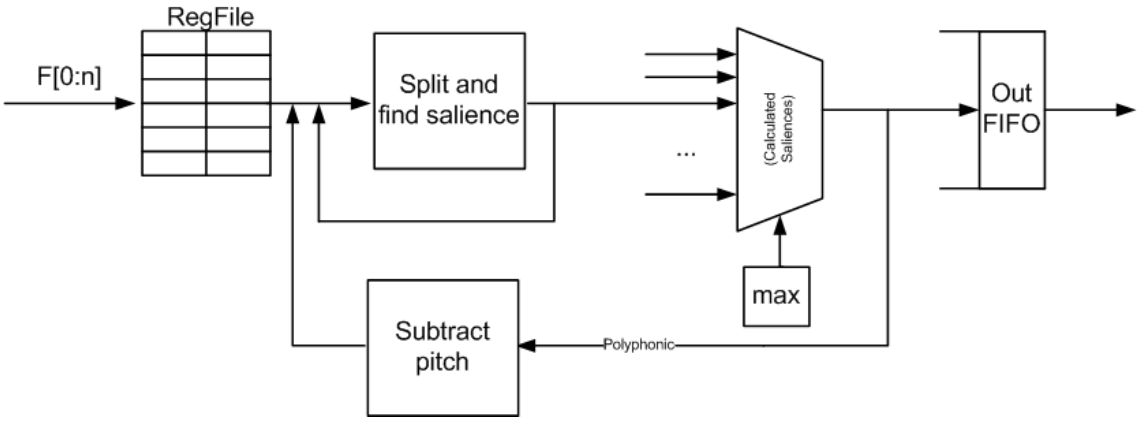

Figure 8: Salience stage microarchitecture.

$$\sigma_b = \sqrt{\frac{1}{K}\sum_k H_b(k) \mid X(k) \mid^2} \tag{5}$$

The Salience module requires the FFT output to be indexed and addressable, so we used RegFiles to contain this data. In addition, the Salience calculation cannot be performed until a full window of FFT output has been received. (Footnote: We only use the first 2048 FFT outputs – this approximates a positive FFT.) Two RegFiles were used so that at any given time, one can be used for calculations and the other can be used to stream in the next window of data.

The module uses Registers of Bits to keep track of indices of interest. The Salience sorting algorithm begins by "splitting" the Regfile into two blocks, by using the minimum, maximum, and midpoint indices in its calculations. Each block represents its center frequency, whose associated FFT value gets summed with up to 20 of its harmonics' FFT values. This sum is the Salience. The module choses the block with the largest Salience and repeats the algorithm: split, calculate Salience for the newly split half-blocks, find the block with the global maximum Salience, and iterate with that block.

When the block size becomes small enough, the detected pitch with the largest Salience gets enqueued into the output FIFO. The user-specified number of polyphonic pitches determines whether the Salience module starts anew with a new data frame or finds more pitches in the current window.

To find additional pitches in a window, the module first updates the data RegFile to remove the contributions from the detected pitch. Then, the module repeats its pitch detection on the modified data. Detecting a single pitch takes roughly 50,000 cycles. At a clocking rate of 50MHz, this means that each pitch takes 1ms to detect. If we use 4-tone chords as our input, each window takes 4ms to process through Salience. The 50,000 cycles number comes from code that completely unfolded for-loops. We planned to change that after our timing analysis by completing multiple for-loop iterations per rule instead of just one iteration per rule.

6 Implementation Evaluation

We encountered some standard coding challenges when writing our system. For example, in Salience, we replaced many Vectors with RegFiles and unfolded our for-loops. We also changed our microarchitecture in Salience – instead of estimating whether to continue polyphony detection (using energy), we depend the user to input the number of pitches in the music.

Whitening was very difficult to understand and implement. It required interpolation and a lot of Vector manipulation. Implementing Cube Root was also difficult because most resources show cube root iterative estimation for decimal numbers, not binary representations. Sorter and Salience are bottle-necks because each one requires a full window to be streamed in before it can start outputting.

Module	Lines of code
Absolute value	48
Cube root	200
Hanning multiplication	40
Lily (combines other modules)	150
Salience	400
Sorter	40
Whitening	200
FFT files combined	~600

Figure 9: A listing of number of lines of code for our implementation.

In moving from simulation to FPGA, we encountered some difficulties. First, the FFT module requires enough hardware that we must use the Virtex 6. In addition, we got access to zakota@csail.mit.edu, a server that had enough memory to synthesize our system. It is computationally equivalent to the purity system managed by Nirav.

We synthesized our system and found that our Slice LUT utilization was far too high as summarized below.

```
Slice Logic Utilization:
 Number of Slice Registers:              115688  out of  301440  38%
 Number of Slice LUTs:                   384498  out of  150720  255%
   Number used as Logic:                 167106  out of  150720  110%
   Number used as Memory:                217392  out of  58400   372%
   Number used as RAM:                   209120
   Number used as SRL:                   8272

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     480837
   Number with an unused Flip Flop:      365149  out of  480837  75%
   Number with an unused LUT:            96339   out of  480837  20%
   Number of fully used LUT-FF pairs:    19349   out of  480837  4%
   Number of unique control sets:        3259

IO Utilization:
 Number of IOs:                          45
 Number of bonded IOBs:                  45      out of  600     7%

Specific Feature Utilization:
 Number of Block RAM/FIFO:               22      out of  416     5%
   Number using Block RAM only:          22
 Number of BUFG/BUFGCTRLs:               15      out of 32       46%
 Number of DSP48E1s:                     611     out of 768      79%
```

To mitigate this problem, we began by transferring all of our Hanning and Whitening coefficients to Block RAM. For example, we used BRAM_Configure and LoadFormat to configure a Hanning BRAM with a Hanning text file.

Our timing report did not complete, as our Slice LUT problems prevented mapping from finishing. If we had gotten timing results, we would have modified our critical paths and also improved latency by using multiple for-loop iterations in our unfolded rules.

Part of the power of Bluespec lies in the ease of reusing IP blocks due to modularity and polymorphism. To this end, we made sure to reuse as much IP as possible. We were able to use an existing implementation of a streaming 4096-pt FFT and a square root module. Though there was a slight issue with overflow in the FFT module given our inputs and we were not quite able to produce the correct output as a result, this FFT module fit perfectly into our streaming audio implementation.

7 Design Exploration

We had correctness issues in simulation: some test windows of music data gave correct results, but when we started streaming data through the system, we found that was an error in either the Hanning multiplication or the FFT output. Future work would explore these errors and correct them.

We were not able to perform a timing analysis and optimize our code for timing. A future exploration that would enable true music transcription would be combining our project with a beat detection module, so we can engrave music with varying beat lengths and tempos.

We could also make our window size smaller so the system would work with extremely fast tempos. There would be less latency with smaller windows, because some of our modules cannot begin computation until they have whole windows streamed in.

Another improvement would be increasing the FFT resolution in response to smaller differences between pitches as the frequency lowers. Finally, we could combine the Klapuri algorithm with an autocorrelation function-based algorithm (like Toleinen) as some current researchers are doing. Blending different algorithms can help with accuracy throughout low and high frequencies.

References

[1] Tadokoro, Y.; Morita, T.; Yamaguchi, M., "Pitch detection of musical sounds noticing minimum output of parallel connected comb filters," TENCON 2003. Conference on Convergent Technologies for the Asia-Pacific Region, vol.1, no., pp.380,383 Vol.1, 15-17 Oct. 2003

[2] P. de la Cuadra, A. Master, C. Sapp, "Efficient Pitch Detection Techniques for Interactive Music." Proceedings of the 2001 International Computer Music Conference.

[3] Dziubinski M. and Kostek B. (2004). "High Accuracy and Octave Error Immune Pitch Detection Algorithms." Archives of Acoustics.

[4] Peng, Hong, "Algorithms for extracting square roots and cube roots," Computer Arithmetic (ARITH), 1981 IEEE 5th Symposium on, pp. 121-126, 16-19 May 1981.

[5] Thomas, John M. "Robust Realtime Polyphonic Pitch Detection." Thesis. George Mason University, 2012. Print.