Anthony Guzman
Blake Russell
ECE 153A
Professor Forrest Brewer
November 12, 2021

Lab 3A - Microphone: Computing FFT

Purpose and expected goals of the lab

The purpose of this lab is to create a Chromatic Tuner by using FFT to compute the frequency of a note being played, captured by the embedded microphone on the FPGA. The goal of this lab is to conduct a better method to compute the FFT quicker while retaining accuracy.

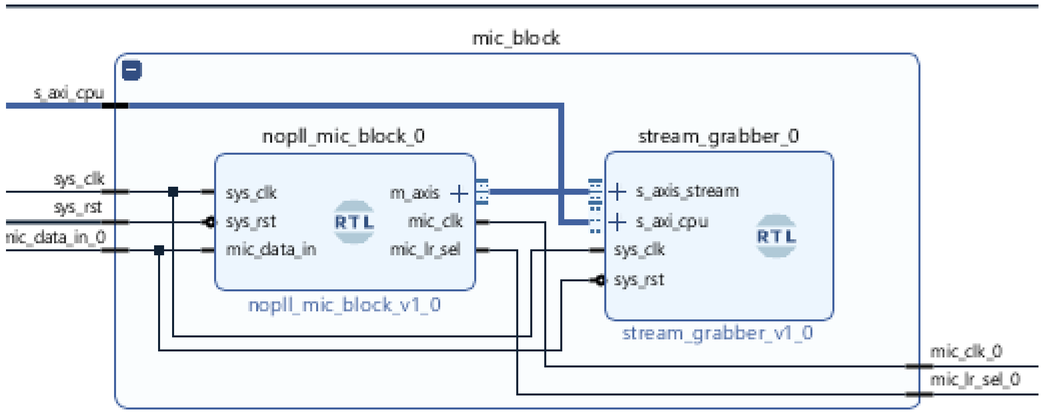
Methodology

Initially we added a microphone block and a stream grabber to our Vivado block design, directly connected to the CPU clock, listening from an onboard FPGA microphone. In order to optimize the FFT processing time, instead of recalculating the sine and cosine values, we initialized those values and stored them into a table, a method considered Look Up Table. To optimize the computation time further, we reduced the sample size from 512 to 128 samples, while maintaining high accuracy. A performance analyzer was implemented and ran every millisecond through the timer interrupt handler, counting the time it takes for each major function in the program and accurately calculating total time.

Results

1. Include a Block Design Schematic of your Vivado Project.
Block Schematic Diagram will be attached in the file named: system.pdf

2. Include a list of the Mic Block internal wiring

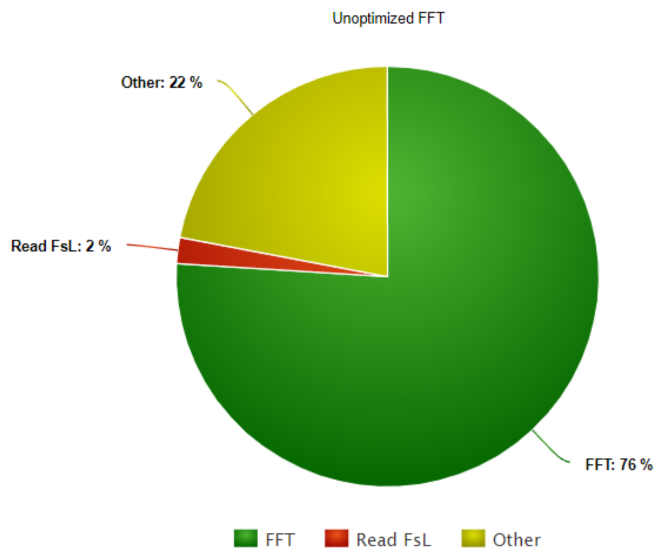


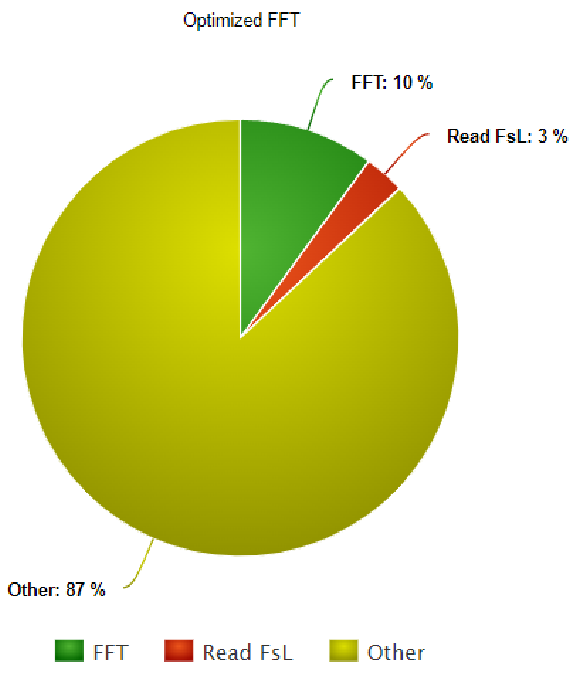
3. Include a description of the steps taken for optimizing the original FFT code.
The first thing we noted when we initially ran the code was the heavy delay within each frequency calculation. The long wait time was due to the heavy computations of running the cosine and sine functions. Instead of calculating the cosine and sine values each time fft was run, we pre-computed them and stored them into a matrix, and used the matrix as a lookup table. We managed to decrease the computing time from approximately 1300ms to 70ms. The FFT was then optimized even further by cutting down the sampling size from 512 to 128. As a result from cutting down the sampling size, we were able to get a computing time of approximately 30ms, with very minimal loss in accuracy.

4. Was there a difference in the FFT computation for low and high frequencies?
For our setup, we noticed that as the frequency was lower, the accuracy loss was closer to ~15-20Hz, but as we reached near the 5KHz, the accuracy was pretty spot on. Any more than 6kHz, will result in discreferency.

5. How does the performance of your FFT code compare to that of the default code?
The new FFT method has severely increased the computational speed, at a cost of some accuracy. In order to keep the same computational speed but with greater accuracy, we'd have to have a manual toggle method that switches between detecting high vs low frequency.

6. Include results from your performance analyzer. The performance analysis should consist of an estimate of the fraction of total execution time spent in each program code module. Estimate the average time you spend in your profile code per sample interrupt (times the size of its code segment). This is the 'goodness' metric for your profiler.


Unoptimized FFT
Other: 22 %
Read FsL: 2 %
FFT: 76 %
FFT
Read FsL
Other


Optimized FFT
FFT: 10 %
Read FsL: 3 %
Other: 87 %
FFT
Read FsL
Other