ECE Lab 3A: FFT

Varun Iyer · *Rajan Saini*

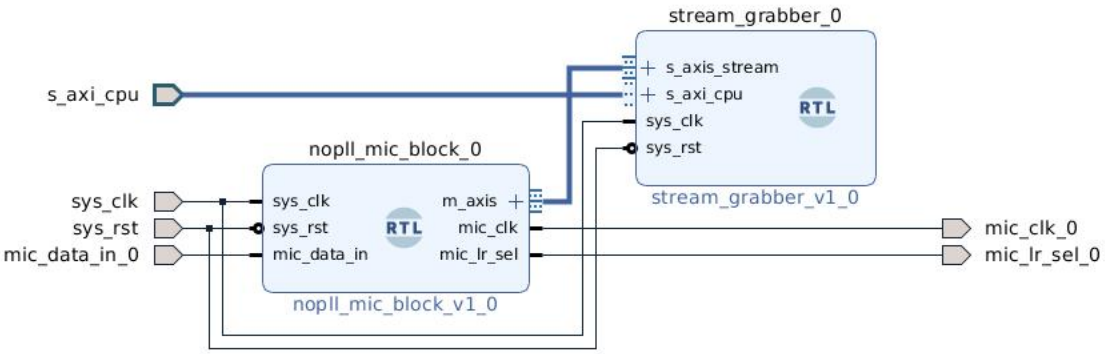
2019-11-25

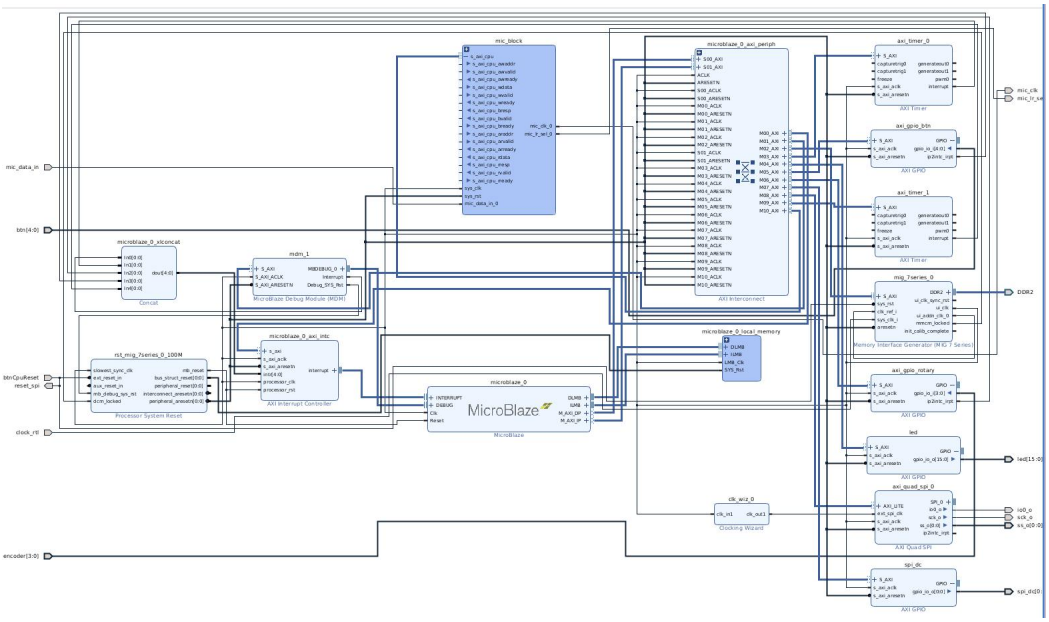
1 Purpose

Our primary goal in this lab was to determine the frequency of an audio signal in under 50 milliseconds. There are two necessary tasks to complete this goal. First, we must incorporate the Nexsys board's on-board microphone into our hardware design. Second, we must optimize the Fast Fourier Transform algorithm provided to us to operate within this time limit.

2 Methodology

2.1 Microphone Incorporation into Vivado





2.2 FFT Optimization

2.2.1 Twiddle Coefficients

Initially, we noticed that the slowest aspect of the program was the calculation of the twiddle coefficients, which involve multiple sin and cos calculations and complex multiplication. The first step we took was to save the results of sin and cos computation for the same values of `k` and `b` so we didn't have to compute them repeatedly. For further improvements, we

decided to avoid using high-order and computationally expensive Taylor expansions for trigonometric computations. We replaced these with a lookup table and used trigonometric identities and low-order, small-angle Taylor approximations for interpolation. We saw substantial improvements from this (we saw 75% of the computation time melt away), but realized that we could make even more time gains by minimizing the amount of floating point operations and needless π multiplications throughout this calculation. Instead of passing in an angle as a floating point value, we entered it as a ratio between two integers, `k` and `b`. We could now perform all 2π and $\pi/2$ modulo using integer ratios and save floating point operations for the final multiplications.

2.2.2 Division Optimization

We also saw that floating-point divisions were very computationally expensive, despite using an FPU.

We looked for instances of float division by a number of the form 2^n and subtracted the exponential bits by n instead. This resulted in a 10 ms speedup. Whenever this did not hold, we did our division using fixed-point arithmetic so that we could divide by a power of 2 when converting back to a float.

2.2.3 Imaginary Removal

We noticed that our sampling is entirely in the real numbers and imaginary numbers are passed in as zeroes. For this reason, we removed all reordering and unnecessary computation of the imaginary component of the sample until the calculation of $N = 2$ DFTs, when twiddle multiplication results in non-zero imaginary components.

3 Results

3.1 Low vs High Frequencies

We noticed that our FFT was significantly less precise at lower frequencies. This is because the Fourier transform's "certainty" grows with the number of input cycles increases (this can be visualized as the maximum of the frequency-domain graph increasing with a steeper drop at neighboring frequencies).

Since we are sampling over a constant amount of time, the number of cycles within the sample buffer increases as the frequency increases. It follows that the FFT grows more precise at higher frequencies.

The performance was similar to that of the sample code.

3.2 Profiling

To do our profiling, we had a second timer trigger an interrupt at a high rate. We then counted the number of interrupts triggered at each return address and maintained a hashmap of each count. Finally, we printed the return address with the highest number of "hits" per grand loop cycle. The profiler reported a high amount of sampling calls compared to all others, which makes sense: the computation of the FFT was split amongst multiple functions, spreading their counts over the address map. Specifically, there were an average 12.5 such hits out of an average of 80.0 total, from which we can say that 15.6% of the time was spent sampling. Possible sources of error include time spent computing the hash function within the interrupt handler and a high interrupt rate (which could block a function from executing, artificially increasing the count).