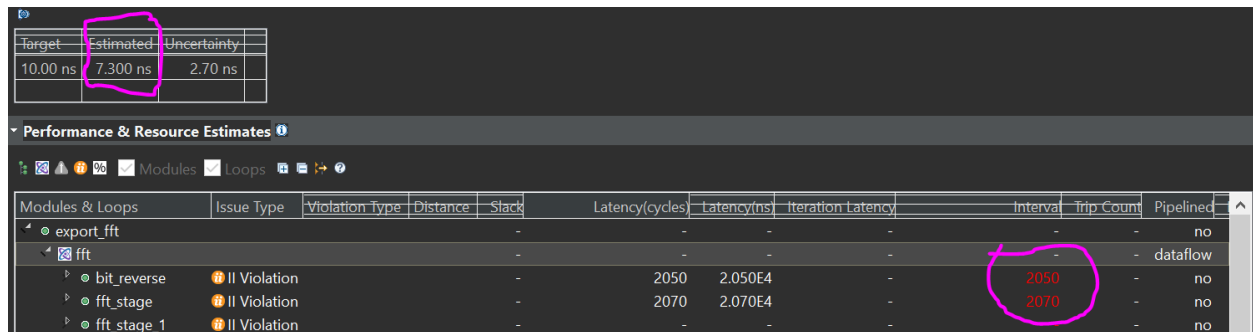


FFT Report - Nate Mosk

The Fast Fourier Transform is an algorithm that optimizes the Discrete Fourier Transform by taking advantage of the symmetry of the DFT transfer matrix.

Because we can pass the multiplication through different sets of stages, it allows us to unfold and pipeline or dataflow to a greater extent, thereby giving better performance.



Target	Estimated	Uncertainty
10.00 ns	7.300 ns	2.70 ns

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined
export_fft				-	-	-	-	-	-	no
fft				-	-	-	-	-	-	dataflow
bit_reverse	II Violation			-	2050	2.050E4	-	2050	-	no
fft_stage	II Violation			-	2070	2.070E4	-	2070	-	no
fft_stage_1	II Violation			-	-	-	-	-	-	no

From my C-synthesis report, I can see that the initiation interval of the bit reverse and the fft_stage functions was a little under 2100. Along with an estimated clock rate of 7.30 ns, that gives us a throughput of about 66 kHz, which is in line with our goal of > 50 kHz.

I encountered a lot of challenging problems putting this together, and it was rewarding to get to the end.

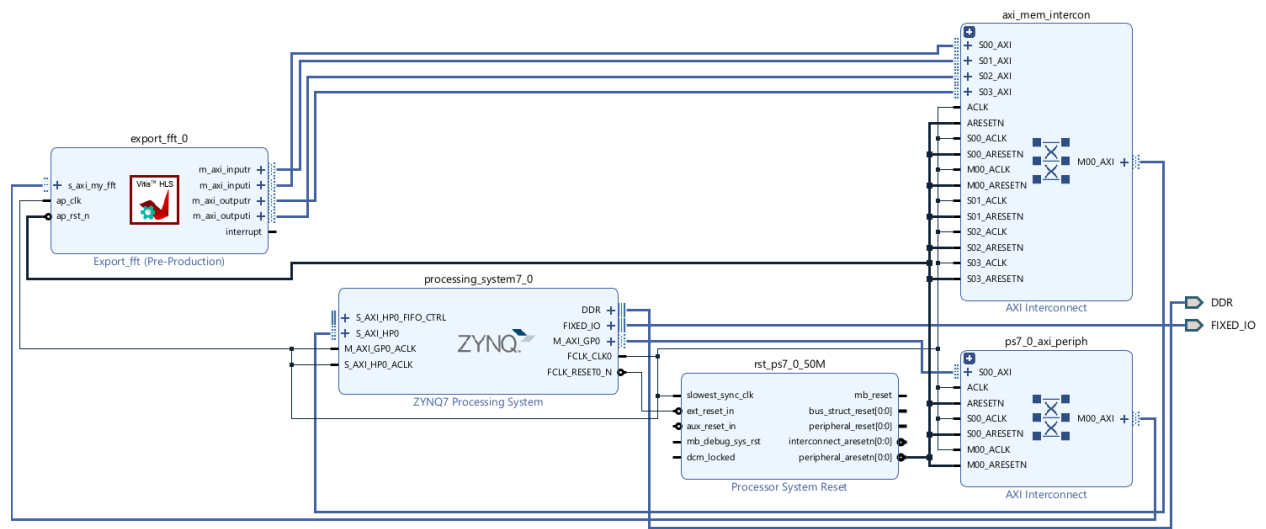
The book was the biggest guide for my design, I essentially followed the construction in the book starting with the baseline implementation and making progressive changes.

The first improvement was the addition of the look up table. That took a lot of examination of the butterflies and stages to get the indexing correct for the look up table. I found that the index increment by increasingly smaller steps each time we progressed to a new stage.

The other big improvement was to isolate the loop so that we could send through different stages in sequential order. That way we could take advantage of the directives to Dataflow and unroll the loop.

My biggest challenge was to implement the FFT on the board. I had previously failed to figure out how to implement streaming on the DFT project, so I was very nervous about this project.

By following the axiburst example on the pp4fpga website, I got a basic understanding of how to use the m_axi directive. Then I extrapolated from that design to add more inputs and outputs as well as reworking the testbench and the jupyter notebook to fit our FFT design.



I ended up with the design above, and it worked on the board, giving me an output that matched our golden output.