FIR Filter Design and Implementation

1. Project Outline

The objective of this course project is to design and implement a low-pass FIR filter. This filter should have at least 100 taps, a transition region of 0.2pi-0.23pi rad/sample, and a stopband attenuation of at least 80 dB. To achieve performance gains, filter coefficients can be quantized, and the hardware implementations can include pipelining and/or parallelization.

2. MATLAB FIR Filter Design, Quantization, Frequency Response

i. FIR Filter Design

MATLAB's Filter Designer was used to design an unquantized 108-tap low-pass FIR Filter. I chose 108-tap because it exceeds the 100-tap requirement, while still divisible by two and three. This will be important when we look to add two, or three levels of parallelization.

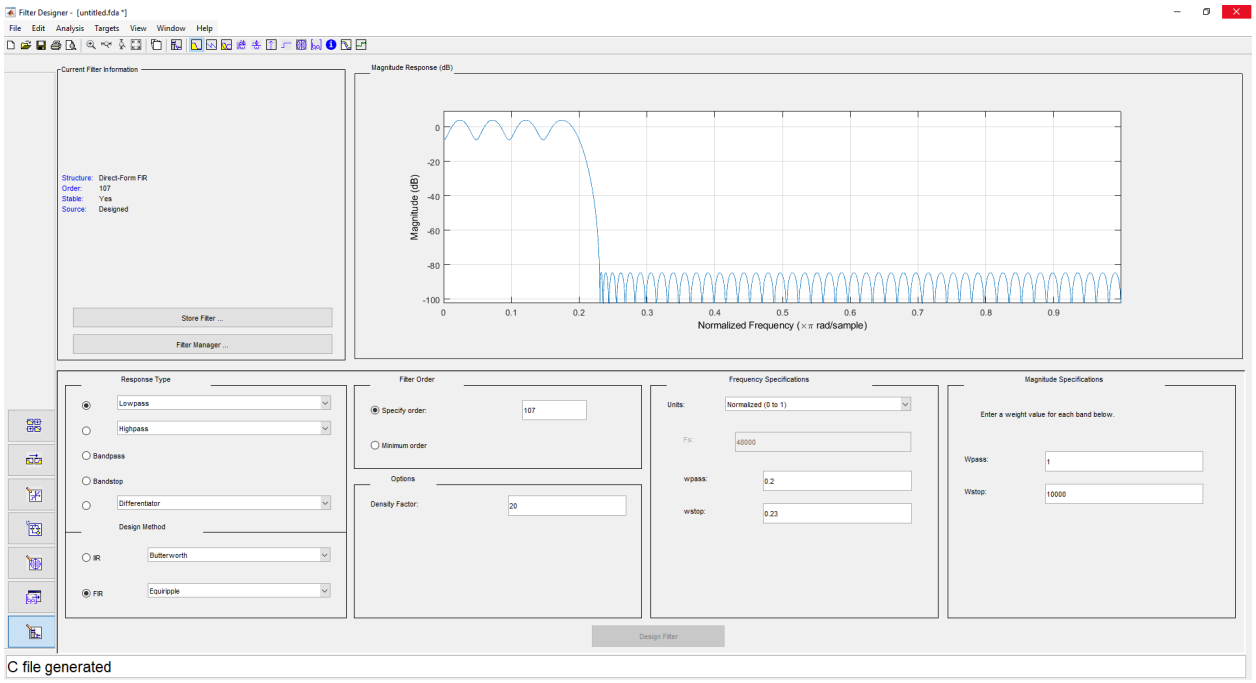

Figure 1: MATLAB Filter Designer

ii. Unquantized (Original) vs. Quantized

To determine the Filter's frequency response, we used another MATLAB script that allows us to view the filter's passband. Both an unquantized (original) and quantized filter were used in this test. My quantization method involved converting the filter taps (originally double-precision

floats) into a signed 24-bit representation. I also challenged the issue of arithmetic overflow by properly scaling inputs and using intermediate registers during MAC stages in the architecture. The figure below shows that the stopband attenuation is at least 80 dB, and that the transition region occurs at around 2.23pi rad/sample. The overlap between the Quantized and unquantized response is very large, indicating near identical performance.

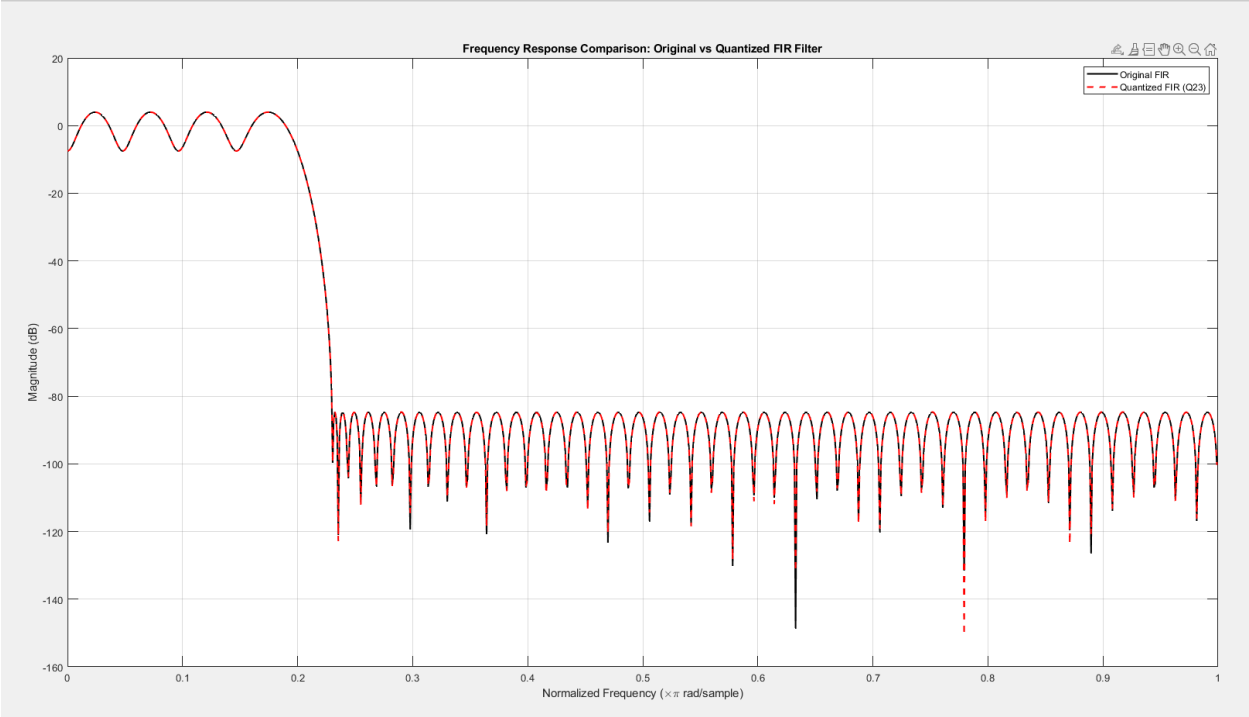

Figure 2: Unquantized (original) vs. Quantized

iii. Benchmark Input Signal

For simulation setup, and to determine the FIR filter's effectiveness, I passed in a sine wave (1 kHz) containing noise. This signal was generated through MATLAB and saved as a .data or .txt file. This allows us to determine the FIR filter's ability to remove noise and preserve key signal shaping. This sine wave will be simplified to 16-bit integer inputs.

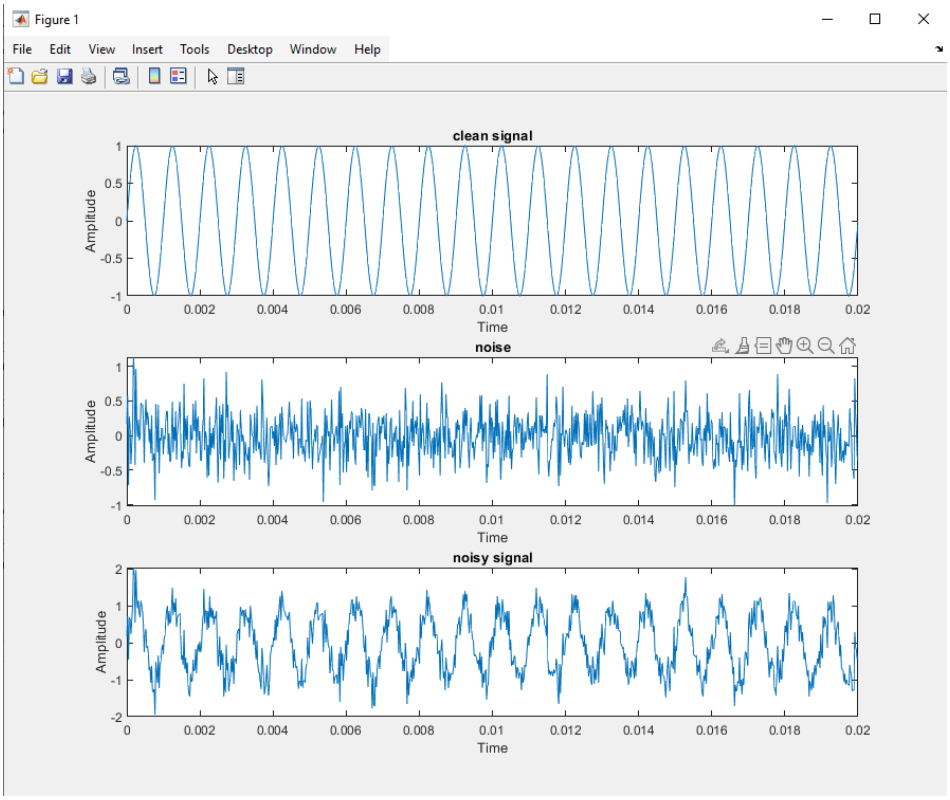

Figure 3: Noisy Sine Wave Input Generation

3. FIR Hardware Architecture

i. Baseline Design

Four hardware implementations for this lab were created. This includes a pipelined variant, N=2 level parallelization, N=3 level parallelization, and pipelined N=3 level parallelization. The first image below highlights the baseline requirement for implementing an FIR filter. This approach uses Multiply-Accumulate (MAC) blocks to delay sample progression. This architecture has a very long critical path, which means there is significant room for improvement.

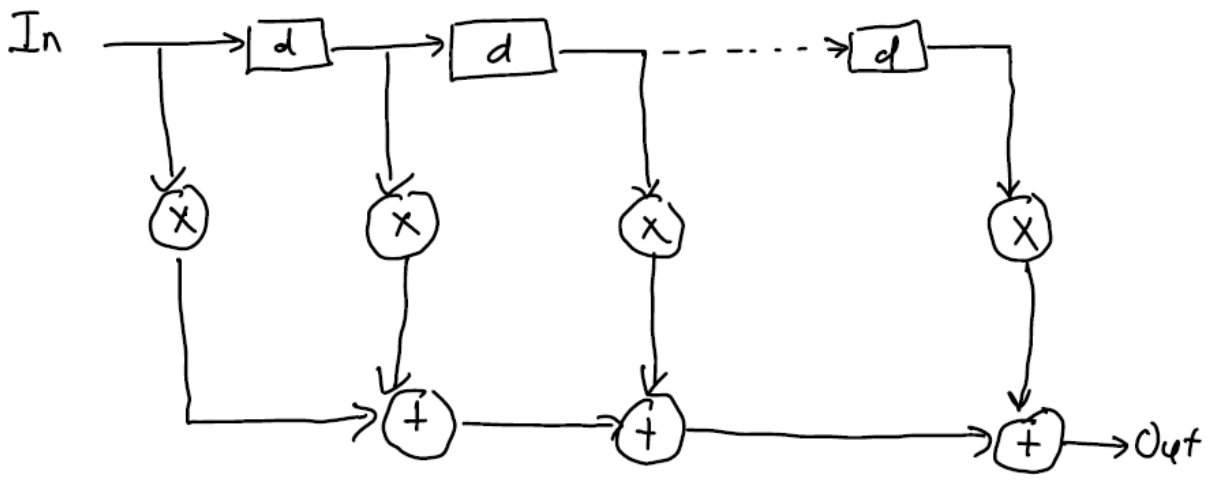

Figure 4: Baseline Implementation

ii. Fully Pipeline Design

The next image below illustrates the ideal full pipelining. Using vertical (MAC) and horizontal (Adder) pipelining limits the critical path to a single multiplier or adder. This allows for throughput to be drastically increased, compared to waiting for one pass at a time.

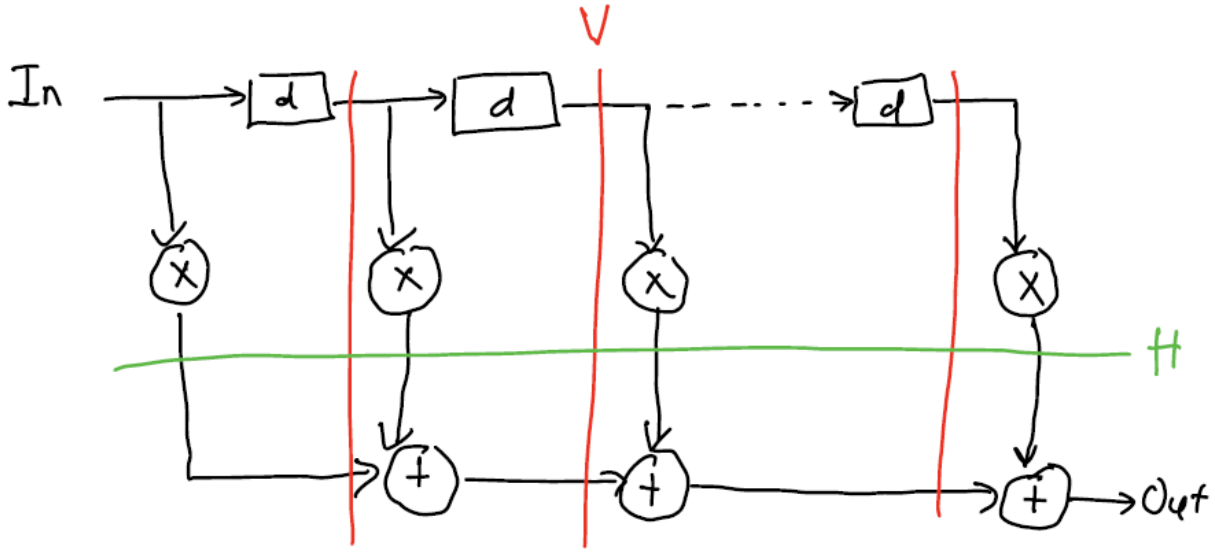

Figure 5: Fully Pipelined Implementation

iii. Simple Pipeline Design

Implementation issues with vertical pipelining (affecting output smoothing clarity) led to a simplified pipelining variation, specifically using horizontal pipelining only. While this still improves the performance, the critical path is still high (up to a “tap count” number of adders.

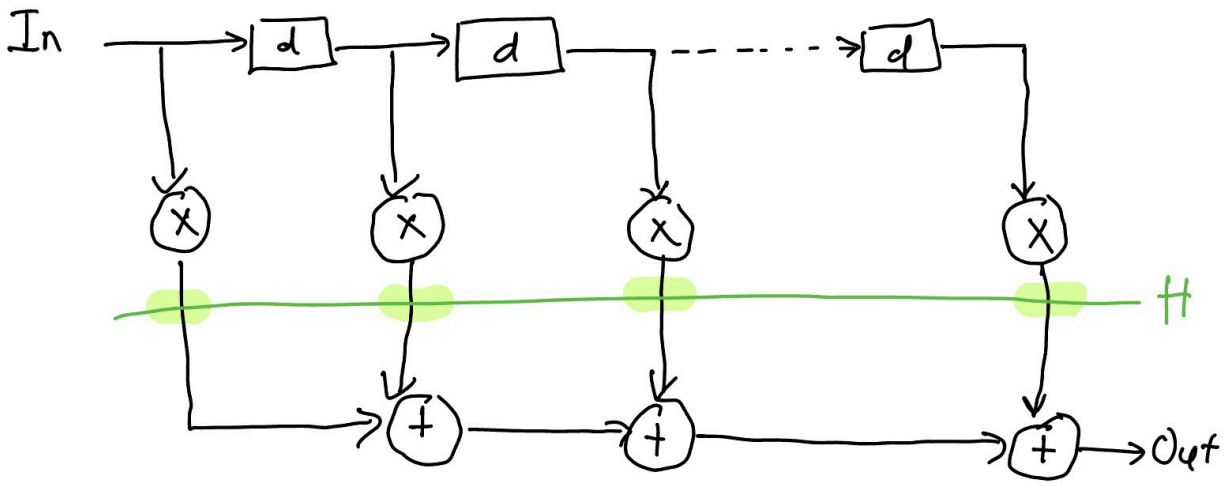

Figure 6: Horizontal Pipeline Implementation

iv. Parallelization (N=2 and N=3)

Parallelization allows for the ability to split the workload across multiple lines, allowing for a multi-faceted hardware approach for this filter. The two images below show an N=2 and N=3 parallelization implementation. The 108 taps (coefficients) were split across each line through a process known as polyphase decomposition.

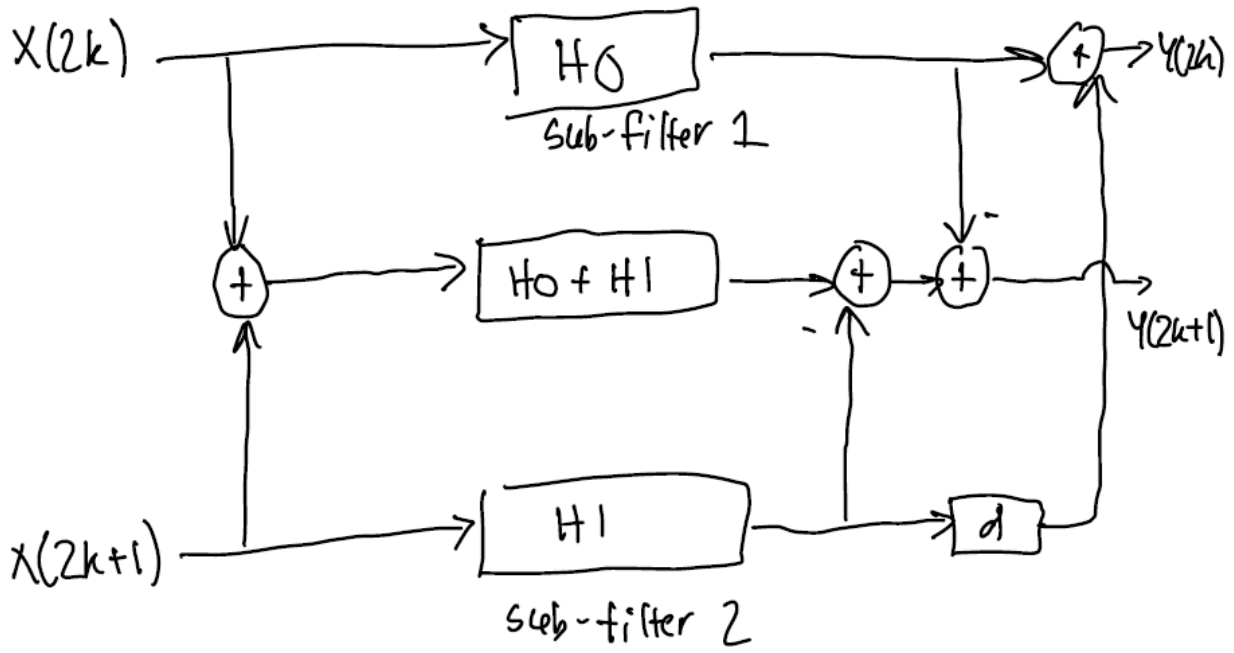

Figure 7: Parallelization (N=2)

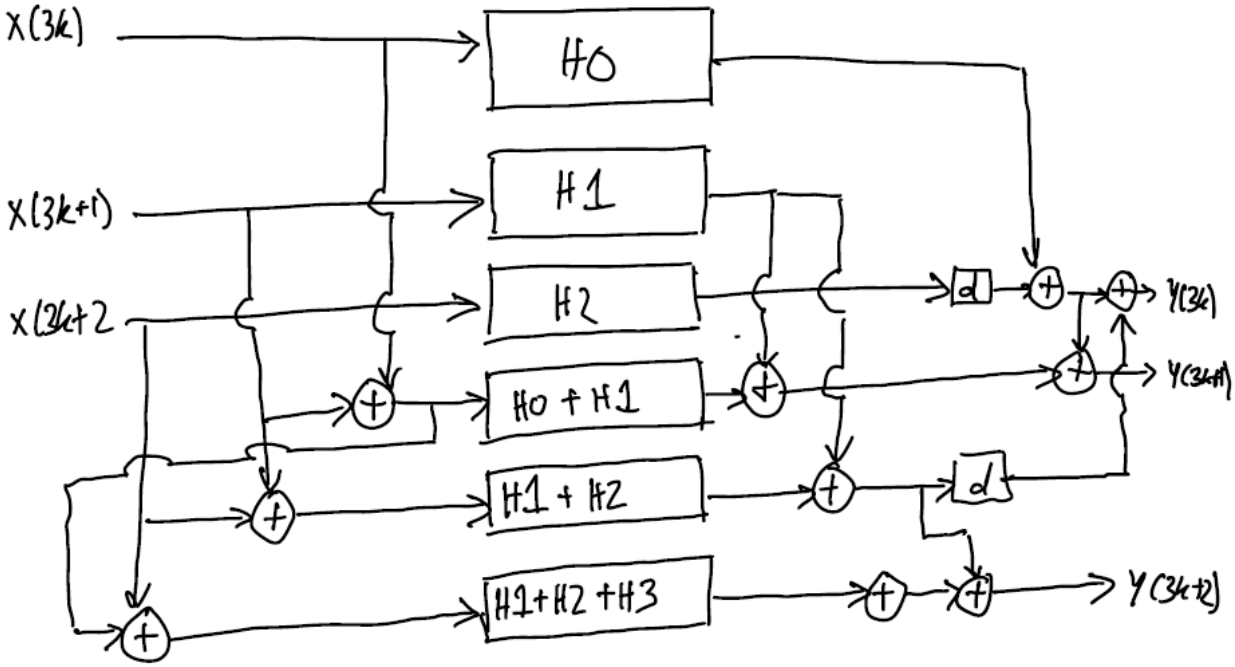

Figure 8: Parallelization (N=3)

v. Pipelining and Parallelization (N=3)

Combining pipelining and L=3 parallel allows for even higher performance. Implementing pipelining into three subfilters allows for increased pipelining throughput across a split workload.

4. HDL Code Structure

The SystemVerilog structure for this project (for each of the 4 filter implementations) includes the filter design file (and sometimes an adjacent top.sv), a testbench file (ran for simulation), a sine wave input signal file coupled with binary signal data (generated from the noisy sine wave MATLAB file), and a counter file to continually increment and pass time. The measured values were input and output data, showing the original noisy sine wave input compared to a smoother, refined output. The results for each of the 4 implementations can be seen below. For the examples containing parallelization, the input and output are split into multiple groups, based on the parallelization level.

i. Pipelining

Figure 9: Pipeline ModelSim

ii. Parallelization, N=2

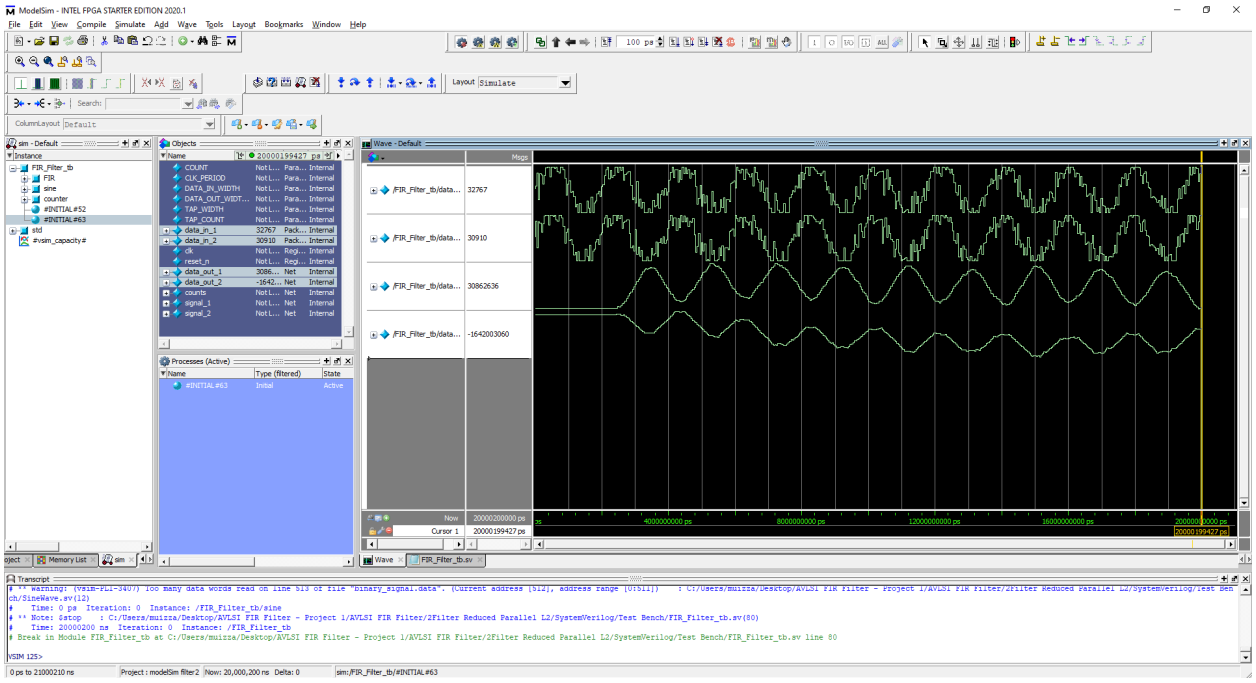

Figure 10: Parallelization (N=2) ModelSim

iii. Parallelization, N=3

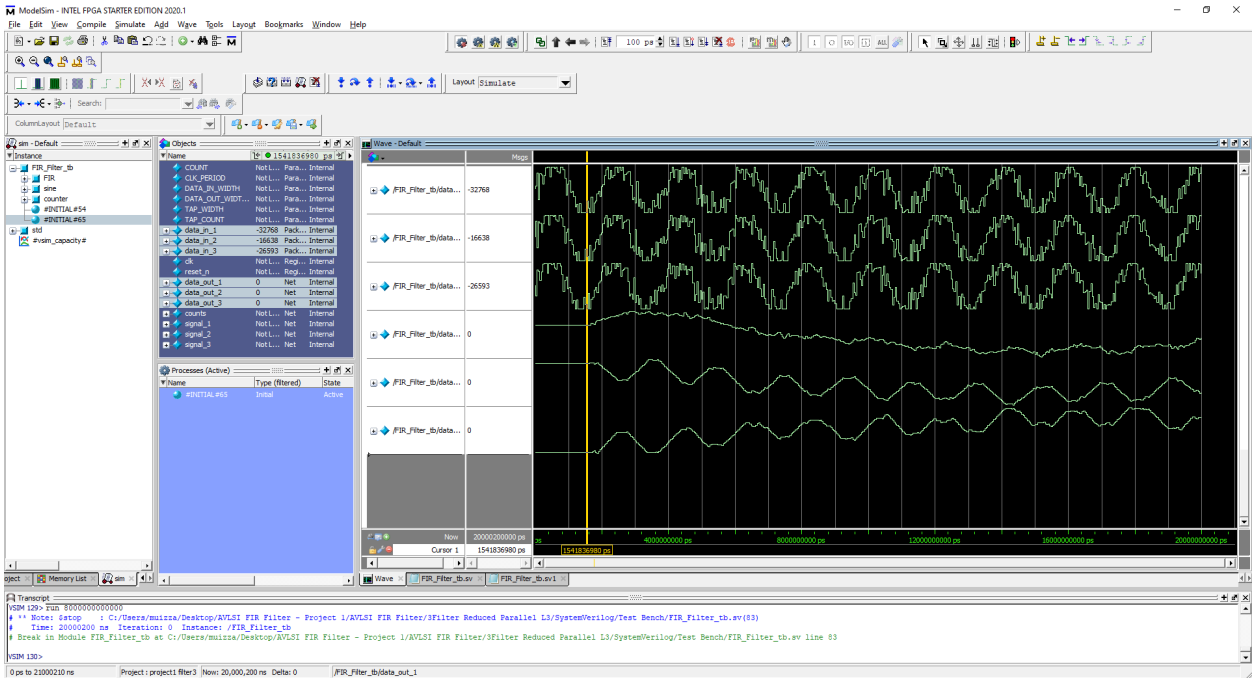

Figure 11: Parallelization (N=3) ModelSim

iv. Pipelining & Parallelization, N=3

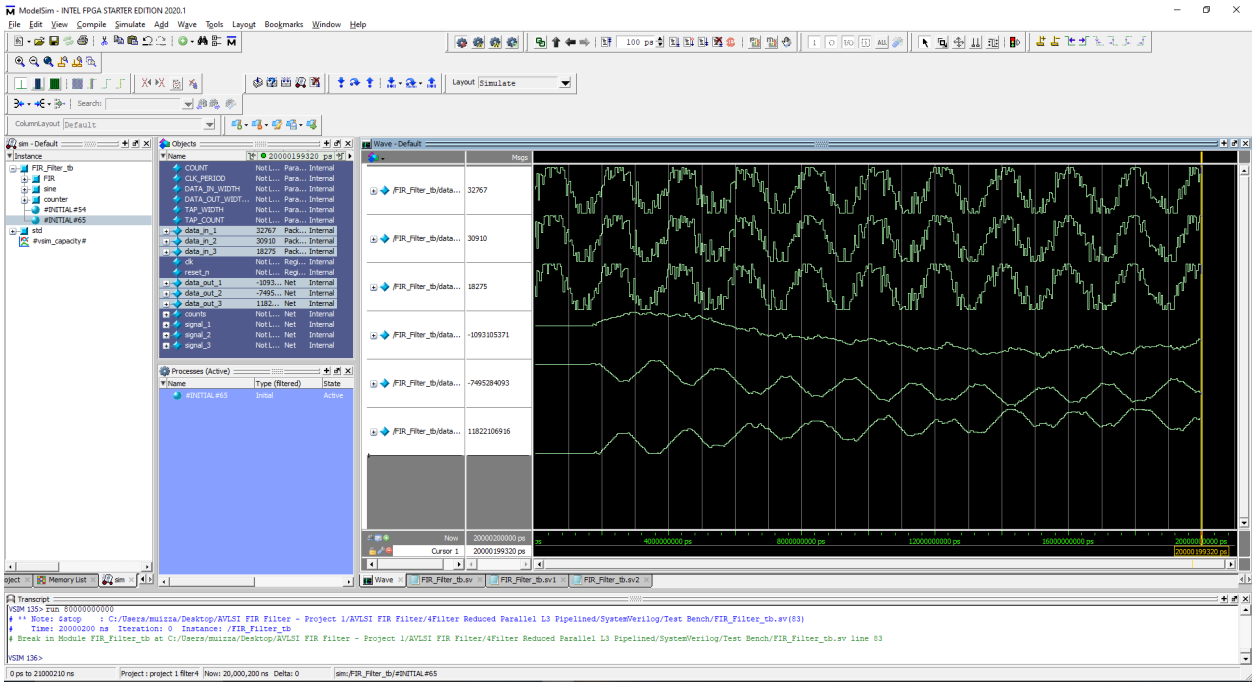

Figure 12: Pipelining & Parallelization (N=3) ModelSim

5. Hardware Implementation Results

Hardware Implementation Results were carried out through AMD's Vivado software. The implementations were all run on device ID "xc7k70tfbv676-1", which was chosen for its higher port/pin count. The implementations were analyzed for area (cell count), power consumption, and timing (speed and performance). Screenshots of in-depth results can be found below and in the "Analysis" subfolder under each implementation folder (1Filter Pipelined, 2Filter Reduced Parallel L2, 3Filter Reduced Parallel L3, 4Filter Reduced Parallel L3 Pipelined). At the end, there is a table summarizing all of the data.

i. Pipelining Hardware Implementation Results

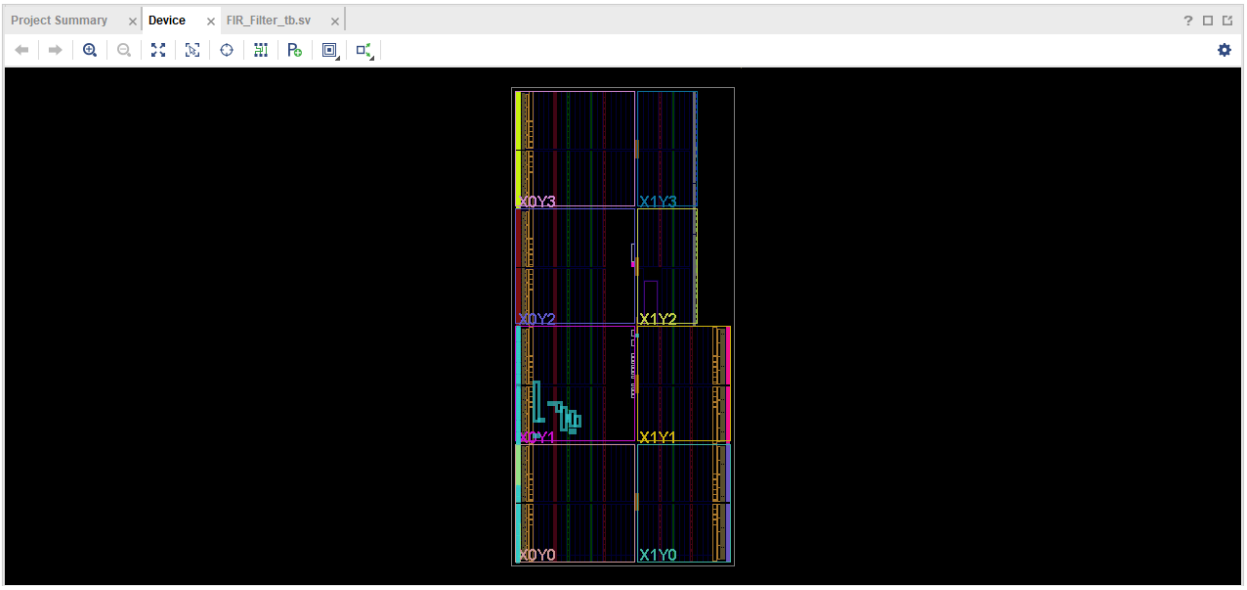

Figure 13: Pipelining Device Layout

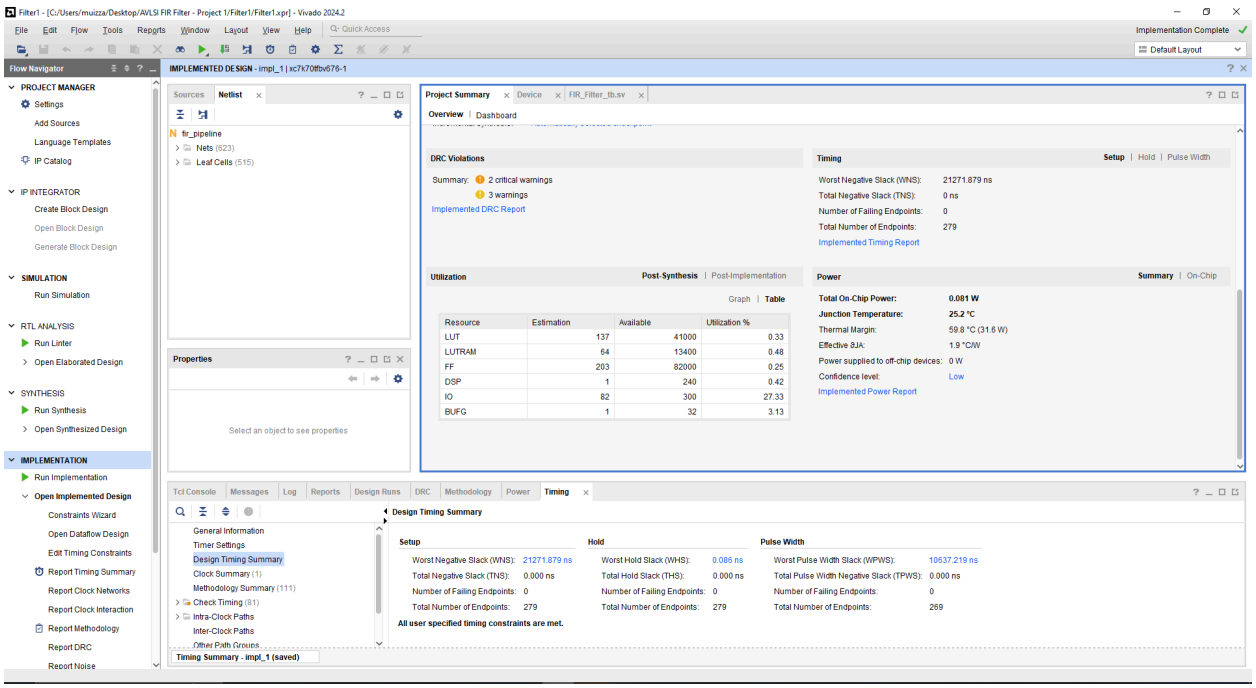

Figure 14: Pipelining Cell Count & Summary

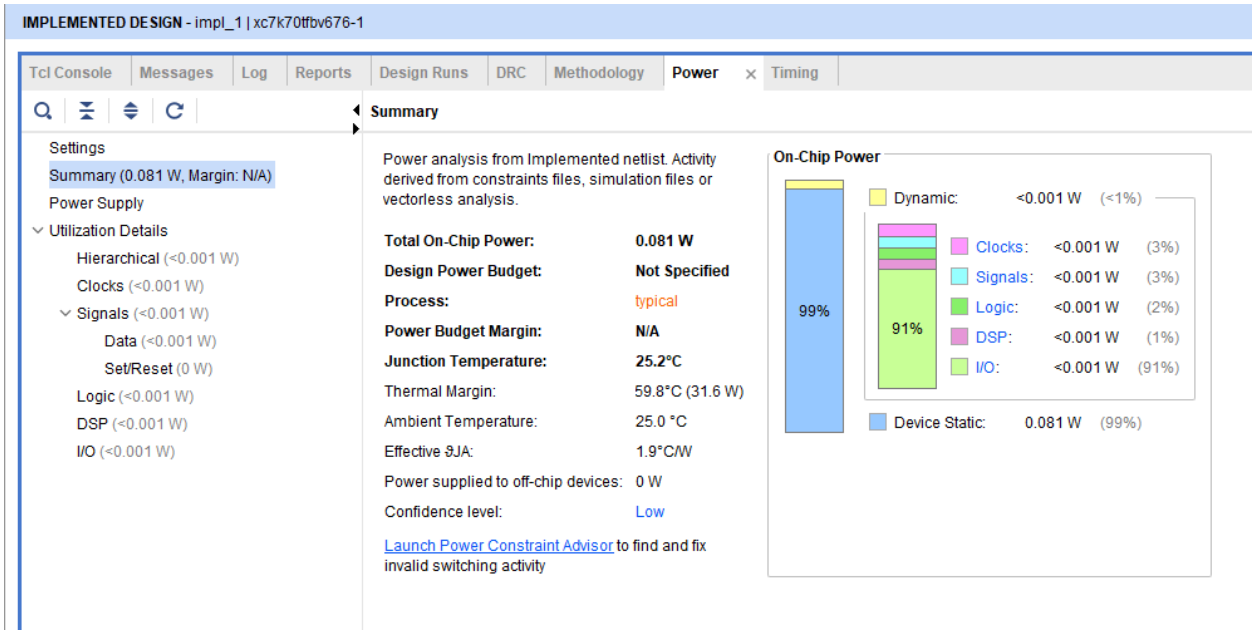

Figure 15: Pipelining Power

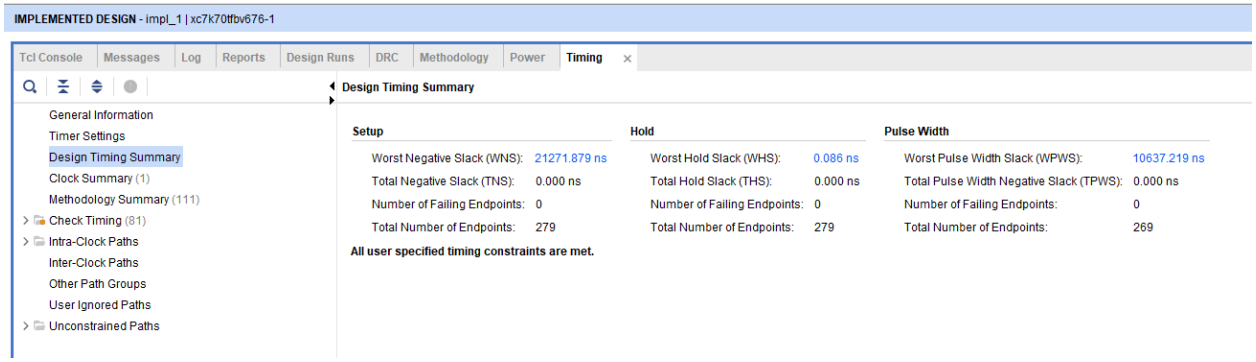

Figure 16: Pipelining Timing

ii. Parallelization (N=2) Hardware Implementation Results

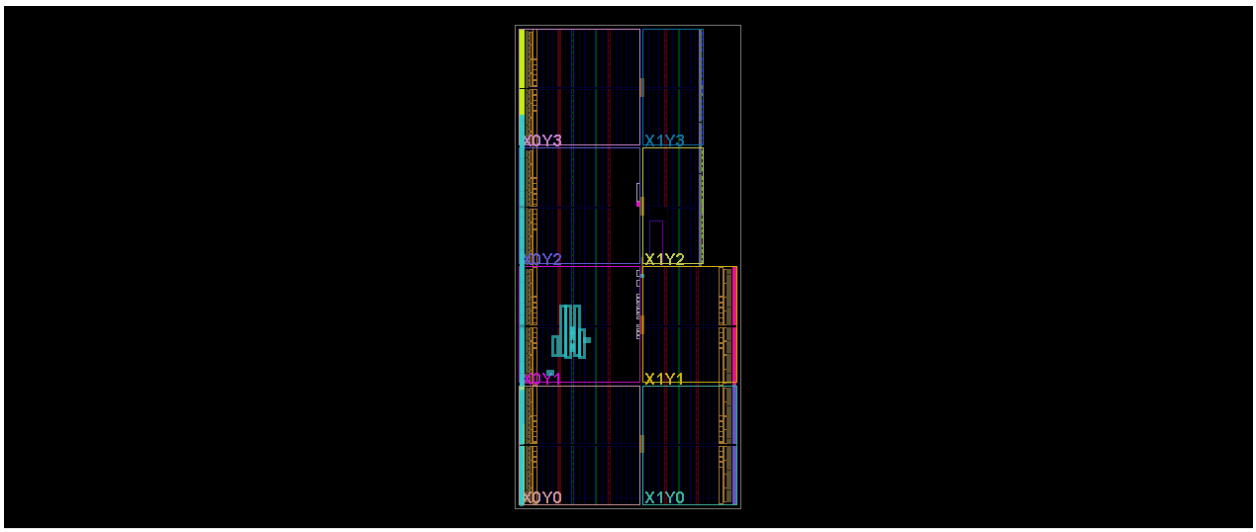

Figure 17: Parallelization (N=2) Device Layout

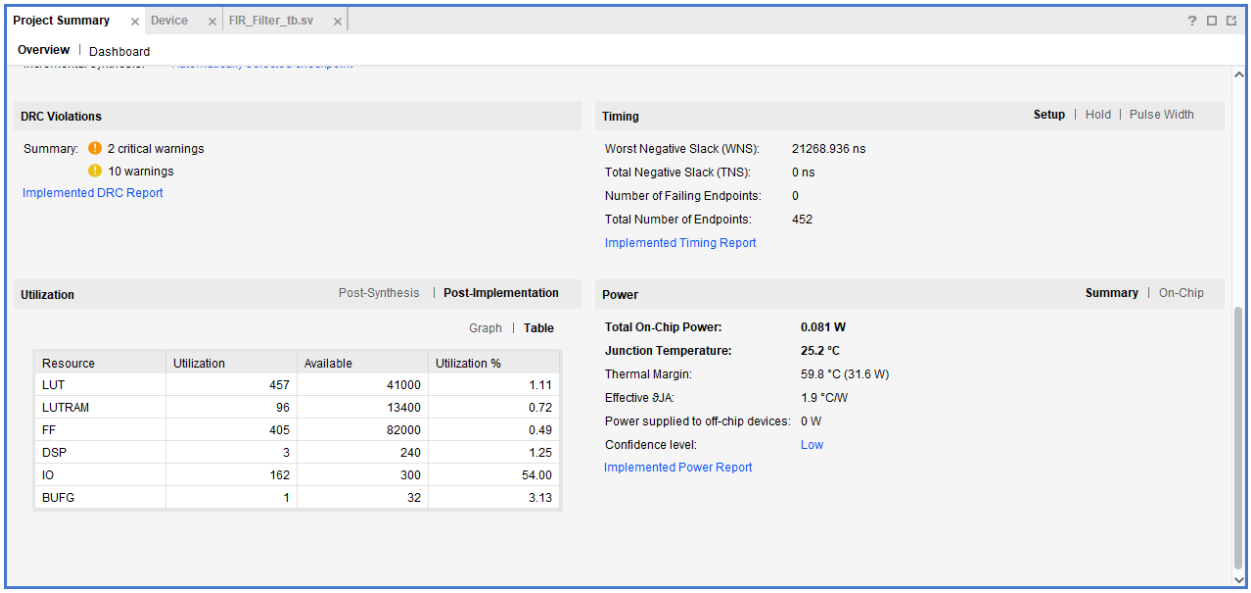

Figure 18: Parallelization (N=2) Cell Count & Summary

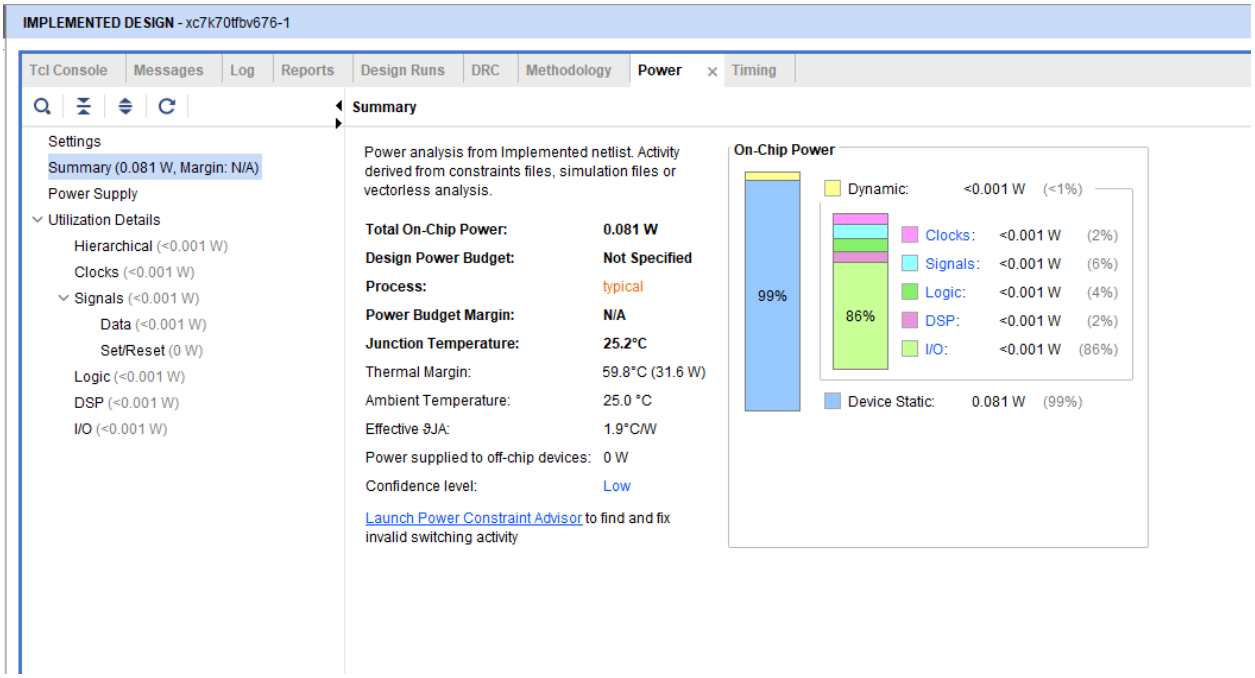

Figure 19: Parallelization (N=2) Power

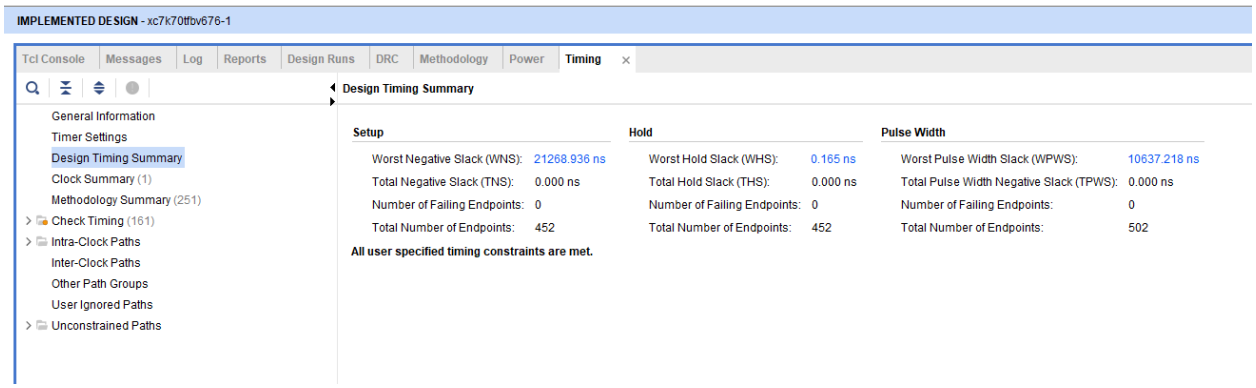

Figure 20: Parallelization (N=2) Power

iii. Parallelization (N=3) Hardware Implementation Results

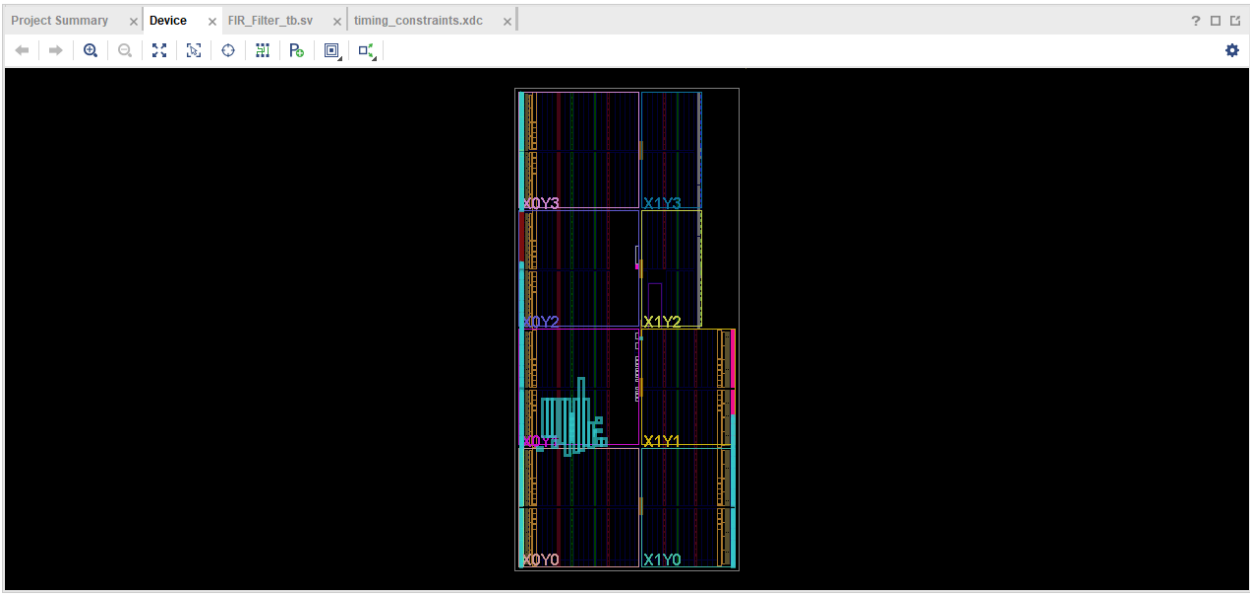

Figure 21: Parallelization (N=3) Device Layout

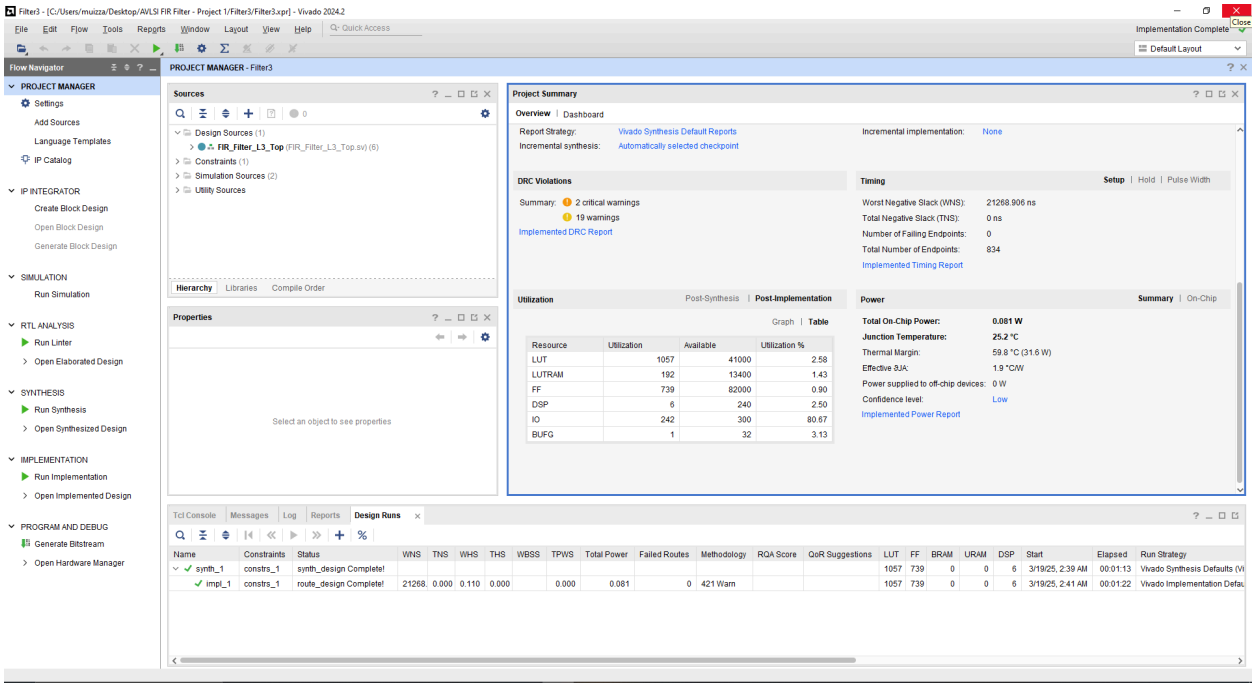

Figure 22: Parallelization (N=3) Cell Count & Summary

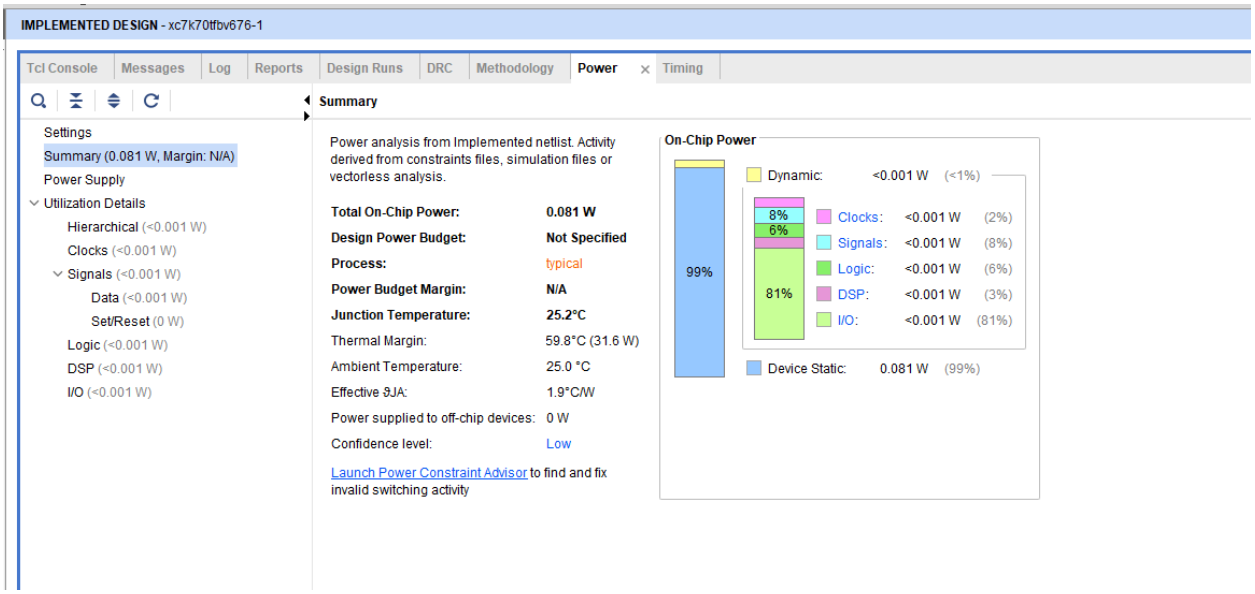

Figure 23: Parallelization (N=3) Power

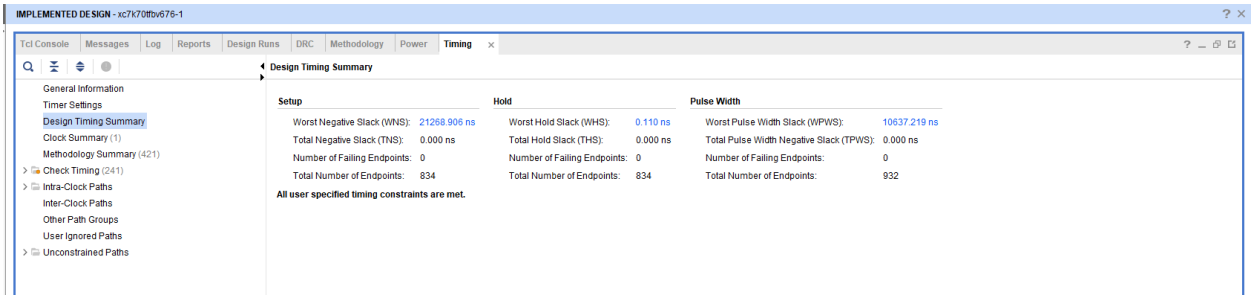

Figure 24: Parallelization (N=3) Timing

iv. Pipelining & Parallelization (N=3) Hardware Implementation Results

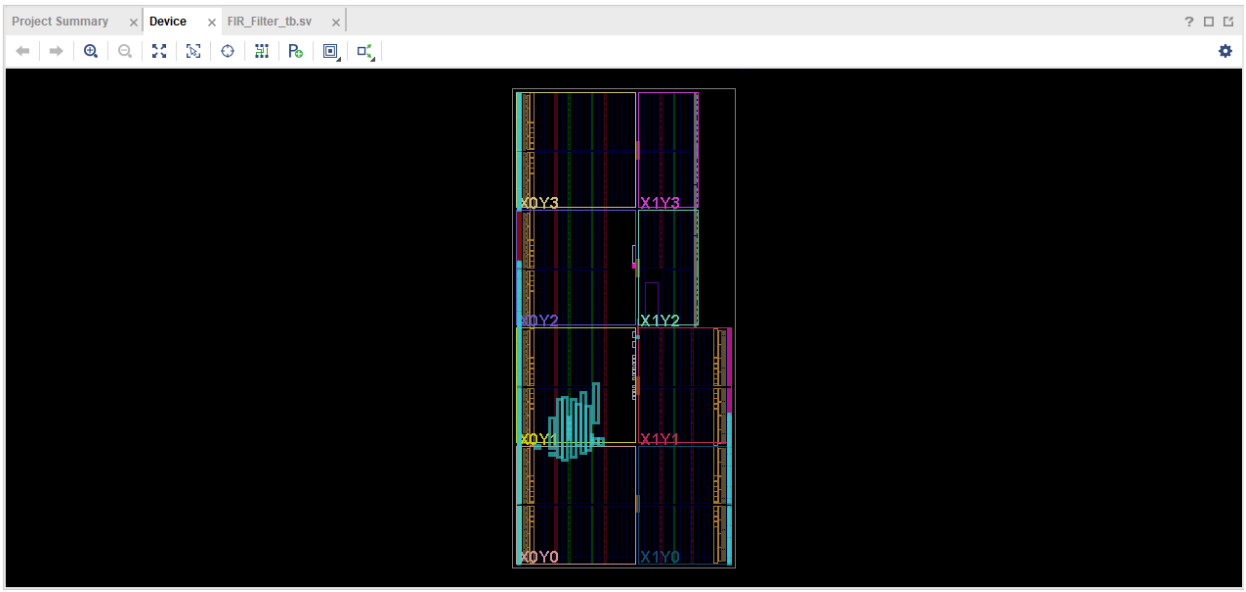

Figure 25: Pipelining & Parallelization (N=3) Device Layout

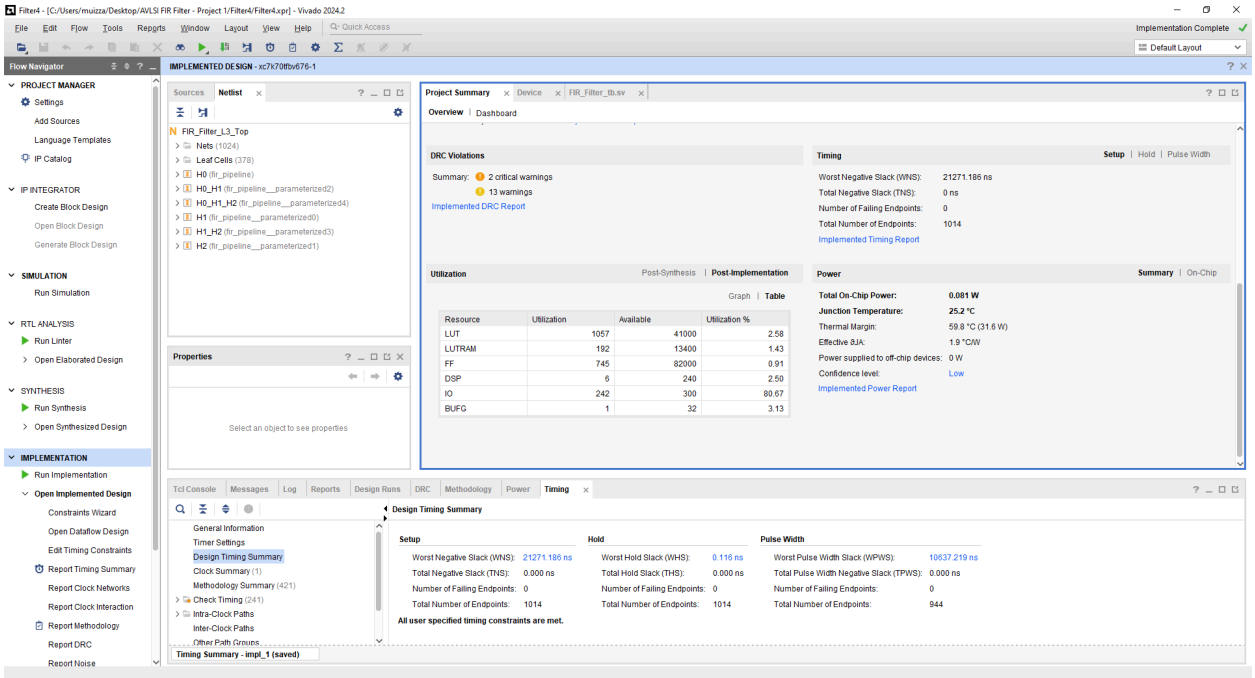

Figure 25: Pipelining & Parallelization (N=3 Cell Count & Summary

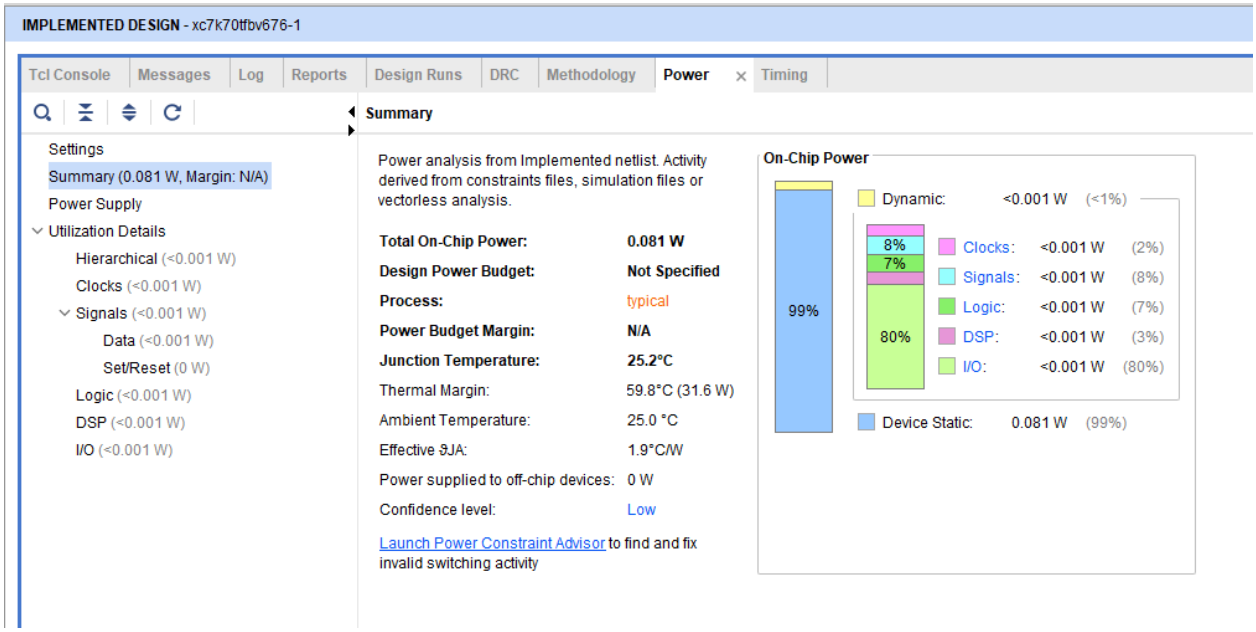

Figure 27: Pipelining & Parallelization (N=3 Power

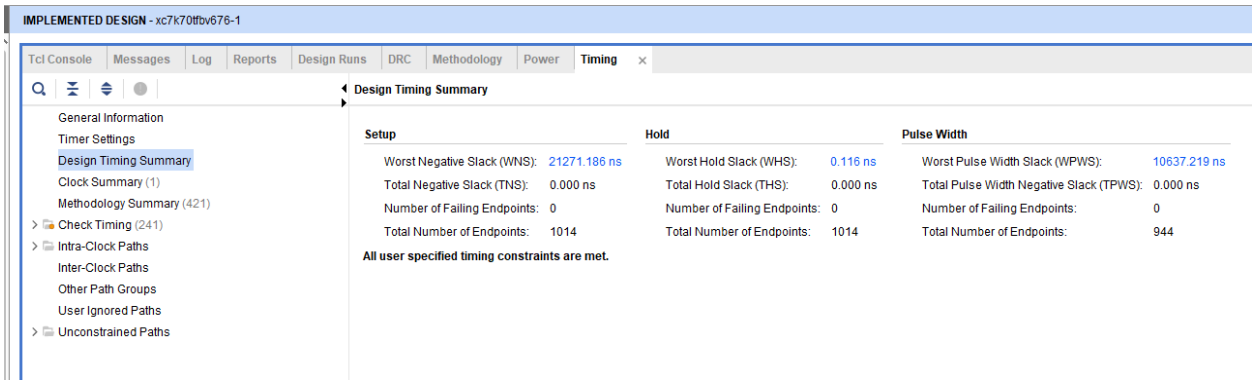

Figure 28: Pipelining & Parallelization (N=3 Timing

Table 1: Hardware Implementation Summary

	Area (Total Cell Count)	On-Chip Power (Watt/area)	Worst Hold Slack (nsec)
Pipelined	488	0.081	0.086 ns
Parallel (N=2)	1,124	0.081	0.165 ns
Parallel (N=3)	2,237	0.081	0.110
Pipelined and Parallel (N=3)	2,243	0.081	0.116

6. Analysis & Conclusion

This project highlights the tradeoff between performance and area. While you can achieve increased performance with processes like L=3 parallel architecture, it comes with a significant cost in total cell count, meaning a larger area is used on-chip. We also see an increase in overall power consumption, as the consistent power density is now calculated over a larger area. Timing can increase with pipelining and parallelization, but once again comes with the drawback of area, power, and complexity. This simplified pipelining does not provide the expected benefit that full pipelining should. The critical path of this pipeline provides some level of improvement, but it is still too much when compared to a fully efficient pipeline.

Quantization of filter coefficients provided immense value by decreasing the total number of bits per tap. Decreasing to 24-bit allowed us to reduce bitcount, while still retaining precision that matched with the double-floating values. Proper scaling and overflow management with delays ensured that MAC operations occurred successfully, without limiting filter operation.

This project illustrates the process and development of a low-pass FIR filter through MATLAB and HDL. This filter can be improved in areas such as timing with processes such as quantization, pipelining, and parallelization, but at the cost of increased power draw and cell area. The correct design choice will depend on user requirements, use case, and operational environment.