FIR Filter Design and Implementation Project Report

Paul Nieves
Prof. Zhang
Advanced VLSI Design
March 14th 2024

Table of Contents

Abstract

This project focuses on the design and implementation of a low-pass FIR filter using both MATLAB and Verilog. The design aims to filter out noise from a sine wave signal while meeting stringent specifications, such as a transition region between 0.2π and 0.23π rad/sample and a stopband attenuation of at least 80 dB. Various architectural approaches are explored, including pipelining and parallel processing, to achieve high performance on FPGA hardware.

1. MATLAB FIR Filter Design and Simulation

1.1 MATLAB Implementation

MATLAB was employed in two primary aspects of this project. First, for FIR filter design, MATLAB's built-in functions (as described in the MathWorks FIR filter-design documentation) were used to construct a 10**2**-tap low-pass FIR filter. The filter was initially designed in its ideal (un-quantized) form to meet the desired specifications. Second, a MATLAB script named sin_gen.m was developed to generate a noisy sine wave. This script creates a clean sine wave at 1 kHz, adds white Gaussian noise based on a specified Signal-to-Noise Ratio (SNR), and then scales the signal to fit within the range of a 16-bit integer. The binary representation of the scaled signal is saved to a file, providing a test input for simulation purposes.

1.2 Simulation Setup

The generated noisy sine wave serves as the test input for the filter model in simulation. In ModelSim (or another FPGA simulation tool), this input is fed to the Verilog filter implementations to verify that the designed FIR filter successfully removes the noise while preserving the desired signal.

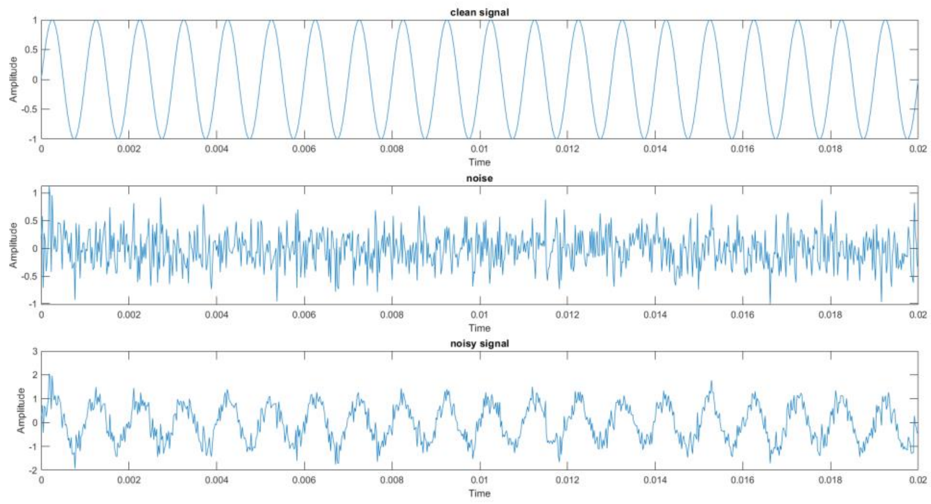


Figure 1: Test Signal Graph MATLAB output

2. Filter Frequency Response and Quantization Effects

2.1 Frequency Response Analysis

The ideal, un-quantized filter exhibits the expected passband along with a steep roll-off in the transition region. After quantizing the filter coefficients, the frequency response was re-evaluated. Although quantization introduces minor deviations from the ideal response, careful scaling and rounding strategies have ensured that the stopband attenuation remains at or above 80 dB.

2.2 Quantization and Overflow Management

The filter coefficients were quantized to a signed 32-bit representation, and analysis showed that this precision is sufficient to closely match the frequency response achieved with floating-point arithmetic. Additionally, techniques such as proper scaling of the input, output, and intermediate data were implemented to prevent arithmetic overflow during the multiply-accumulate (MAC) operations.

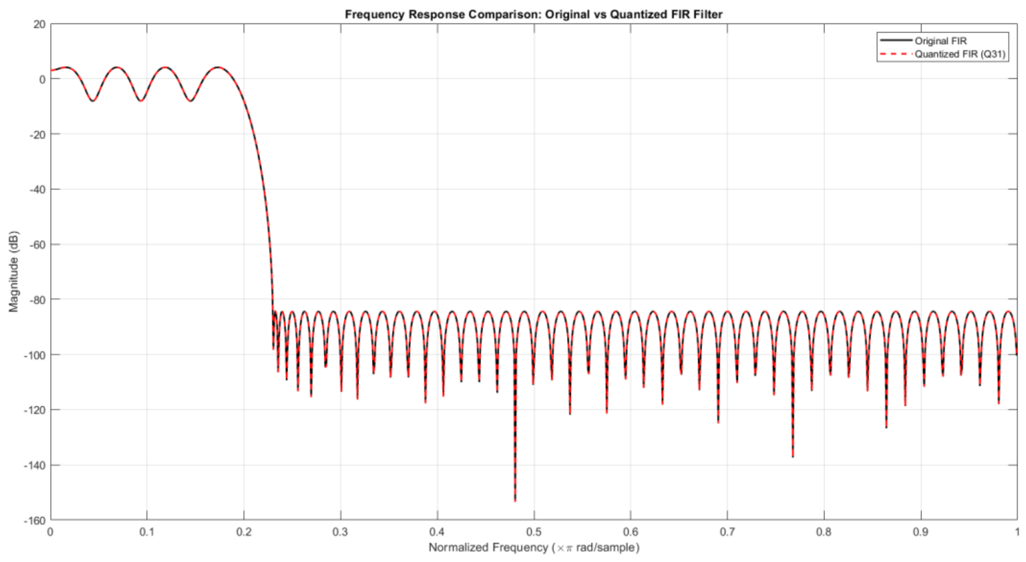


Figure 2.2: Frequency Response Comparison Original vs Quantized FIR Filter

3. FIR Filter Architecture

3.1 Architectural Overview

To meet performance requirements and optimize for FPGA implementation, multiple FIR filter architectures were explored:

- **Directly Implemented (Traditional) Architecture:**

The traditional FIR filter design follows a straightforward Multiply-Accumulate (MAC) approach, where each input sample is delayed, multiplied by a corresponding coefficient, and accumulated sequentially. While simple and easy to implement, this architecture is not optimized for high-speed applications due to its long critical path and lack of pipelining.

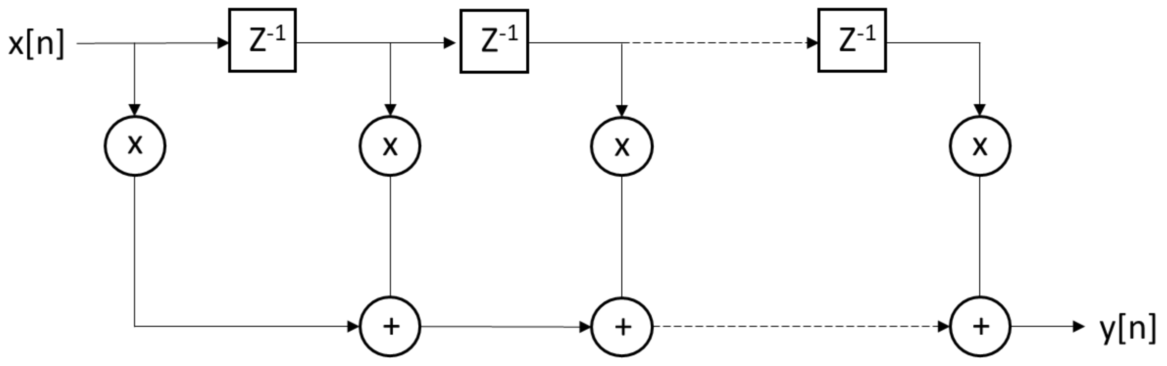


Figure 3: Directly Implemented (Traditional) Architecture

- **Pipelined Architecture:**
  The pipelined design (see module `fir_pipeline`) breaks the MAC operation into stages, each handling a tap. This architecture improves the throughput by allowing operations to be overlapped across several clock cycles.

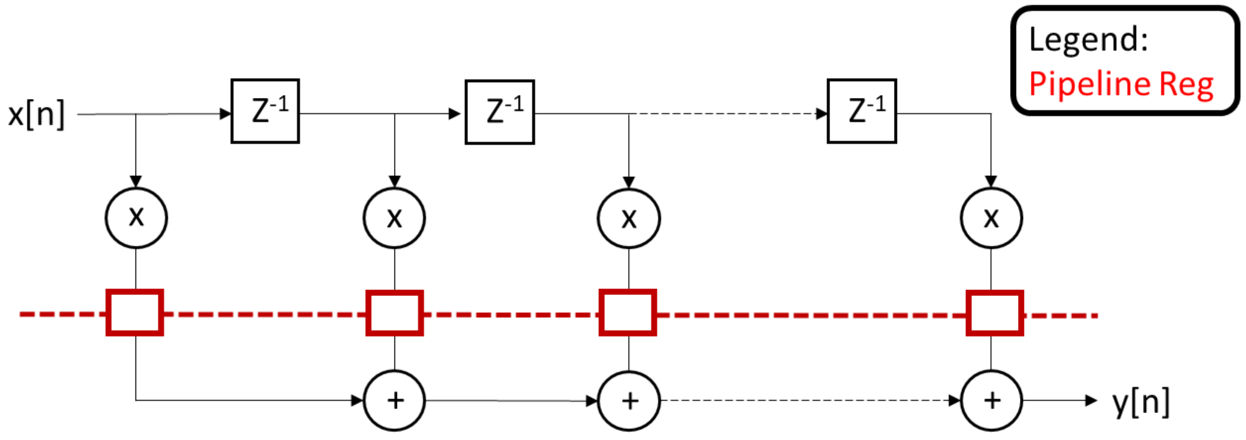


Figure 4: FIR Pipelined

- **Parallel Processing:**
  Two parallel architectures were implemented:
  - **L=2 Parallel Processing:** The filter is partitioned into two sub-filters, denoted as *H*0 and H1 , which process alternating input samples separately (see module *FIR_Filter_L2_Top*). These sub-filters use the traditional FIR filter architecture.

The outputs of $H0$ and $H1$ are then combined to reconstruct the final filtered signal, effectively doubling the throughput.

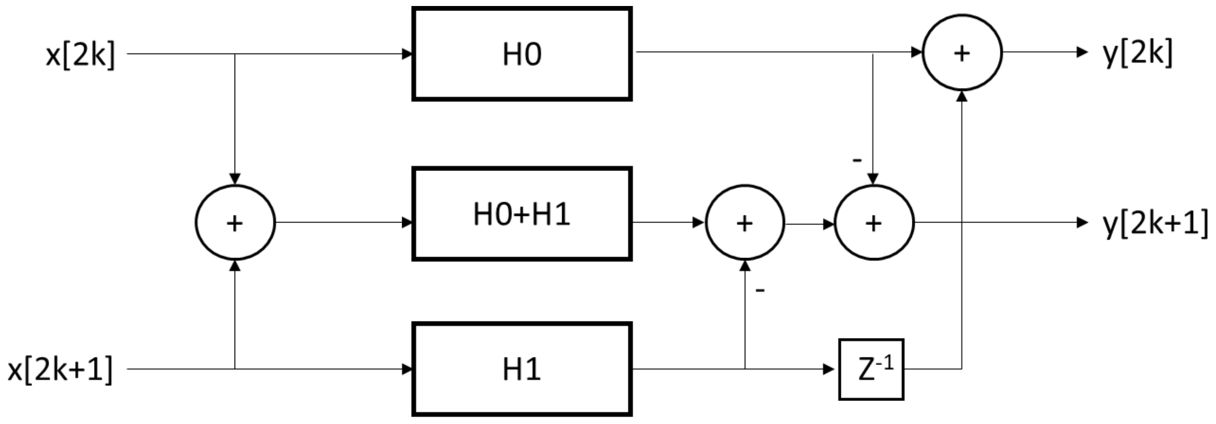


Figure 5: Reduced-complexity 2-parallel FIR filter

- **L=3 Parallel Processing:** Similarly, the filter is divided into three sub-filters, denoted as H0, H1, and H2 (see module FIR_Filter_L3_Top). Each sub-filter processes every third input sample independently using the traditional FIR filter architecture. This approach increases throughput for higher data rates while distributing computational load across the three sub-filters

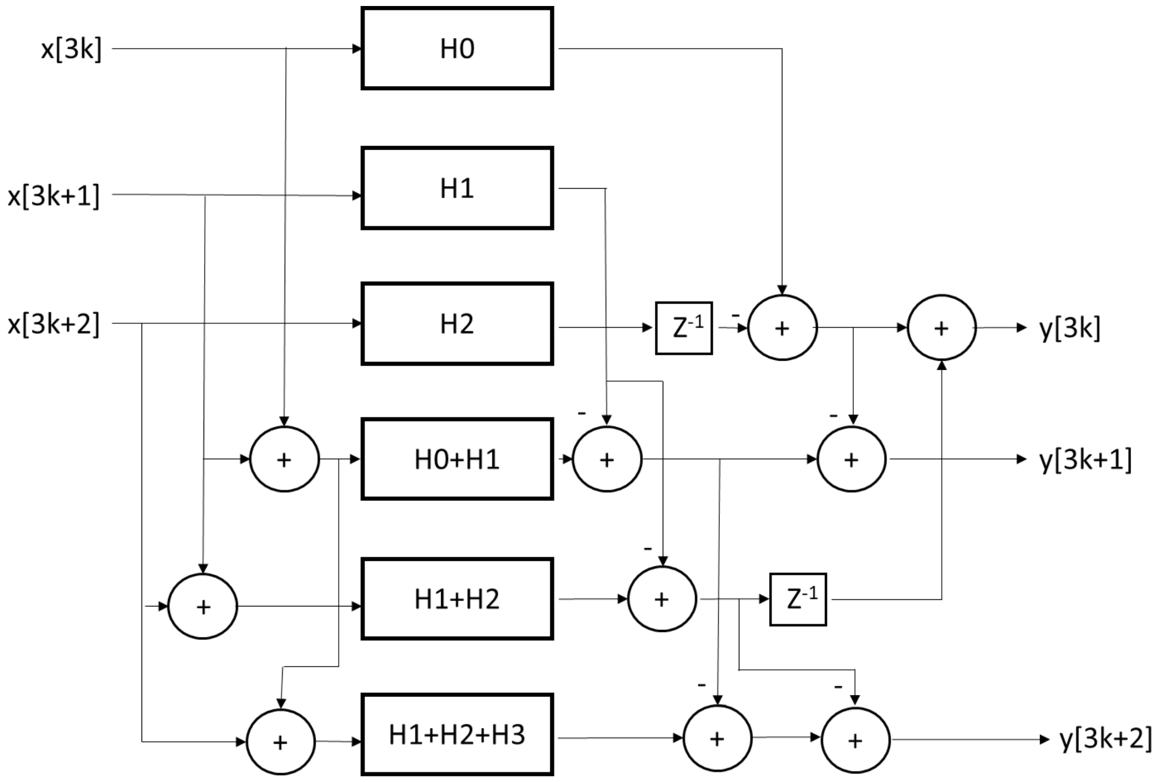


Figure 6: Reduced-complexity 3-parallel FIR filter

- **Combined Pipelining and Parallel Processing:**
  The *FIR_Filter_L3_Pipelined_Top* module integrates both pipelining and L=3 parallel processing for even higher performance. Unlike the standalone L=3 architecture, the sub-filters in this implementation use a pipelined version of the traditional FIR filter, reducing critical path delays and allowing for higher clock frequencies.

3.2 Verilog Code Structure

The Verilog code is organized into modular components:

- **Delay Line Implementation:** Used in every module to store input samples.
- **Coefficient Loading:** Filter coefficients are read from external files (e.g., `"lpFilterTapsBinarySigned.txt"`) during initialization.
- **MAC Operation:** Each tap multiplies the corresponding delay sample with its coefficient and accumulates the result, with intermediate pipeline registers used to support high clock frequencies.

3.3 Filter Coefficients and Polyphase Decomposition

In the parallel processing architectures, the FIR filter coefficients are split into sub-filters H0, H1 and H2 (for L=3) or H0 and H1 (for L=2). These sub-filters each use the traditional FIR filter architecture and are obtained through polyphase decomposition.

Filter Length and Sub-Filter Coefficients

- The original FIR filter has N coefficients.
- In **L=2 parallel processing**, the coefficients are split into two sub-filters:
  - H0 contains the even-indexed coefficients: h[0], h[2], h[4],…
  - H1 contains the odd-indexed coefficients: h[1], h[3], h[5],…
  - The effective filter length of each sub-filter is **N/2**.
- In **L=3 parallel processing**, the coefficients are split into three sub-filters:
  - H0 contains coefficients at indices 0, 3, 6,…
  - H1 contains coefficients at indices 1, 4, 7,…
  - H2 contains coefficients at indices 2, 5, 8,…
  - The effective filter length of each sub-filter is **N/3**.

Coefficient Generation and Binary Representation

The coefficients are generated using a Python script (floatmakercoeficent.py), which:

1. Reads the full set of filter coefficients.
2. Applies polyphase decomposition based on the parallelization factor (L=2 or L=3).
3. Computes additional coefficient sums for optimized hardware implementation:
   - H0+H1
   - H1+H2 (for L=3)
   - H0+H1+H2 (for L=3)

4. Converts the coefficients into a 32-bit binary representation and stores them in text files for FPGA implementation.

By reducing the length of each sub-filter, parallel processing decreases the latency per filter operation and increases throughput. However, it requires additional summation logic to combine the sub-filter outputs correctly.

4. Hardware Implementation Results

4.1 Implementation Details

The hardware implementation of the FIR filter design was carried out using Synopsys Design Compiler targeting the gscl45nm technology. The designs were synthesized for various architectures, including a simple pipelined design and more advanced parallelized versions. These reports provide important metrics that illustrate the trade-offs between area, speed, and power consumption.

4.2 Area Utilization

The synthesis reports provided key insights into the area utilization of each design. For the pipelined design (module *fir_pipeline*), the report shows a modest total cell area of approximately 811 units, with 276 ports, 387 nets, and a balanced distribution of 64 combinational and 64 sequential cells. In contrast, the *FIR_Filter_L3_Pipelined_Top* design exhibits a much larger cell area of about 13,831 units, reflecting the increased complexity from extensive parallel processing with 3,600 ports and 6,856 nets. The intermediate designs, *FIR_Filter_L2_Top* and *FIR_Filter_L3_Top*, show total cell areas of approximately 1,763 and 5,624 units, respectively, with corresponding increases in port and net counts.

4.3 Clock Frequency

Explicit clock frequency values were not provided in the synthesis reports due to warnings about undefined clocks. However, the architectural improvements in both pipelined and parallelized designs are intended to achieve higher operating frequencies that meet the real-time filtering requirements.

4.4 Power Estimation

Power estimation was evaluated under typical operating conditions at a 1.1 V supply. The *fir_pipeline* design demonstrated negligible dynamic power (approximately 0 µW) with a leakage power of around 8.22 µW. In the more advanced architectures, the *FIR_Filter_L3_Pipelined_Top* design reported a total dynamic power of approximately 246.13 µW and leakage power of 105.16 µW, resulting in a total power consumption of about 351.29 µW. Similarly, the *FIR_Filter_L2_Top* design consumed roughly 12.43 µW of dynamic power and 17.08 µW of leakage power, totaling around 29.51 µW, while the *FIR_Filter_L3_Top* design exhibited 83.81 µW dynamic power and 42.62 µW leakage power, amounting to a total power of approximately 126.43 µW.

4.5 Summary Table

The following table summarizes the key synthesis metrics, including area, power (dynamic, leakage, and total), and other pertinent data across the various FIR filter architectures.

Design	Ports	Nets	Total Cells	Comb. Cells	Seq. Cells	Total Cell Area	Dynamic Power	Leakage Power	Total Power
fir_pipeline	276	387	258	64	64	810.95	~0 µW	8.22 µW	8.22 µW
FIR_Filter_L2_Top	652	820	601	145	128	1762.69	12.43 µW	17.08 µW	29.51 µW
FIR_Filter_L3_Top	1660	2596	1611	692	256	5624.09	83.81 µW	42.62 µW	126.43 µW
FIR_Filter_L3_Pipelined_Top	3600	6856	3447	1743	640	13831.21	246.13 µW	105.16 µW	351.29 µW

Table 1: Summary of Synthesis Metrics for FIR Filter Architectures

4.6 Filter Simulation Results

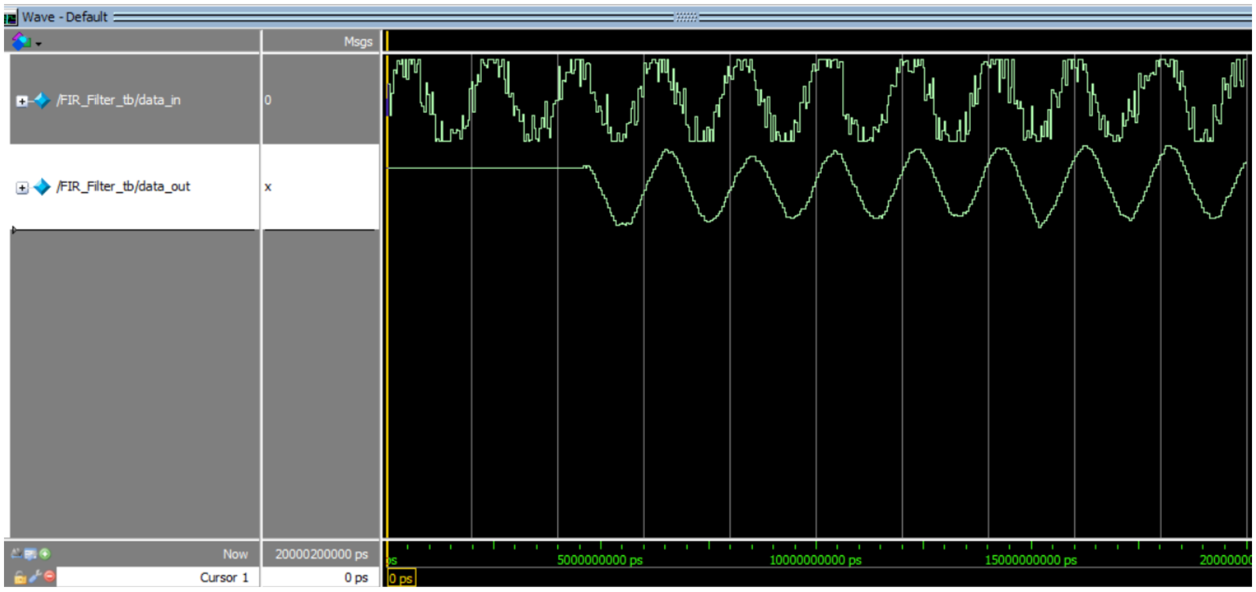


Figure 6.1: FIR Pipelined Testbench Results

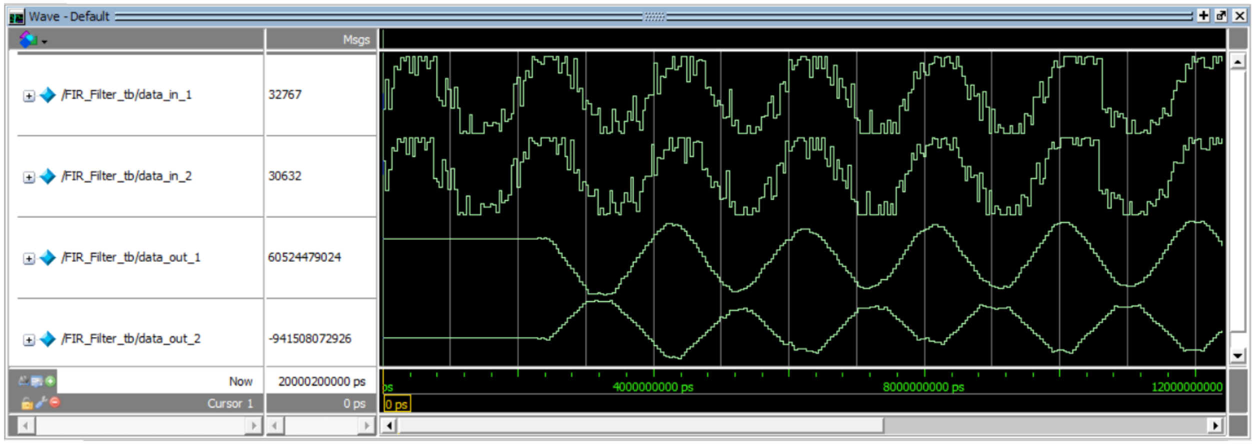
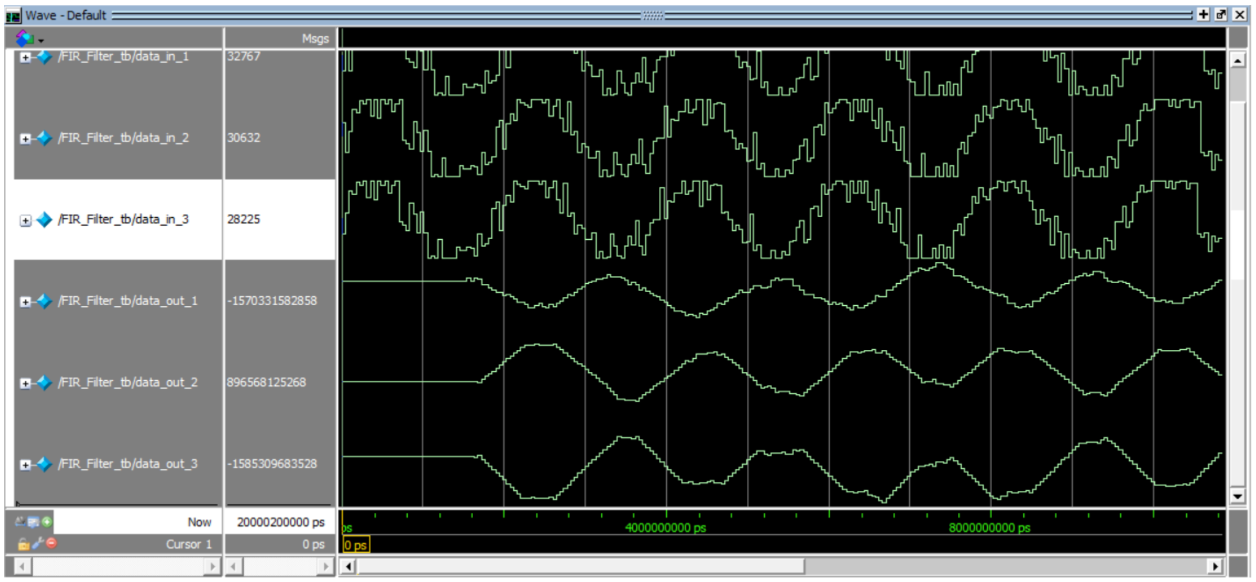
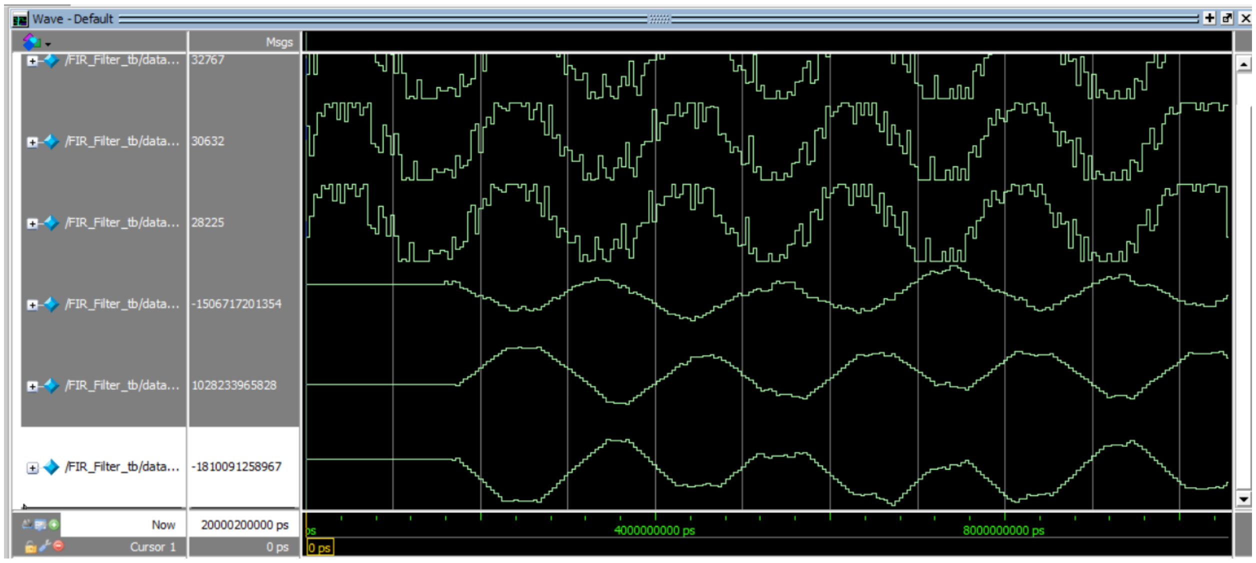


Figure 6.2: Reduced-complexity parallel processing L=2 Testbench Results



```
Wave - Default
                                  Msgs
/FIR_Filter_tb/data_in_1          32767
/FIR_Filter_tb/data_in_2          30632
/FIR_Filter_tb/data_in_3          28225
/FIR_Filter_tb/data_out_1         -1570331582858
/FIR_Filter_tb/data_out_2         896568125268
/FIR_Filter_tb/data_out_3         -1585309683528
Now                               20000200000 ps
Cursor 1                          0 ps
```

Figure 6.3: Reduced-complexity parallel processing L=3 Testbench Results



```
Wave - Default
                          Msgs
/FIR_Filter_tb/data...    32767
/FIR_Filter_tb/data...    30632
/FIR_Filter_tb/data...    28225
/FIR_Filter_tb/data...    -1506717201354
/FIR_Filter_tb/data...    1028233965828
/FIR_Filter_tb/data...    -1810091258967
Now                       20000200000 ps
Cursor 1                  0 ps
```

Figure 6.4: Reduced-complexity parallel processing L=3 and Pipelined Testbench Results

5. Further Analysis and Conclusion

5.1 Performance vs. Complexity

The project illustrates a careful balance between performance and complexity achieved through both pipelining and parallel processing. The L=3 parallel architecture provided significant improvements in throughput and processing speed, as seen in the increased cell area (13,831.21 units) and higher power consumption (351.29 µW total power) in the more advanced designs. However, these benefits came at the cost of increased design complexity and additional logic required to manage data dependencies. On the other hand, the simpler pipelined design (*fir_pipeline*) maintained a minimal power footprint (~8.22 µW leakage) while offering satisfactory performance. This highlights the trade-offs between improved speed and resource utilization.

5.2 Quantization Impact

The quantization strategy employed in this design, using a signed 32-bit representation for the filter coefficients, was highly effective. Analysis confirmed that this level of precision closely matched the floating-point frequency response, ensuring that quantization errors remained minimal. Furthermore, the implemented techniques for scaling and overflow management successfully prevented arithmetic overflow during multiply-accumulate (MAC) operations, preserving the overall filter performance.

5.3 Scalability

The modular approach used in the Verilog implementation demonstrated excellent scalability. This structure enables easy integration of additional filter taps and further parallelization with minimal modifications. The ability to scale the design efficiently ensures adaptability to a variety of applications, making it a practical and reusable solution.

5.4 Conclusion

This project successfully demonstrates the design and implementation of a low-pass FIR filter that meets architecture requirements. The integration of MATLAB for initial filter design and noise simulation, along with a robust and scalable Verilog implementation optimized through pipelining and parallel processing, provides valuable insights into practical design trade-offs. The detailed synthesis metrics, including area utilization and power consumption, highlight the effectiveness of the chosen design strategies.