FIR Filter Design and Implementation
18 February 2025

Contents

1 Introduction

This report presents the design, implementation, and analysis of several FIR filter architectures as part of a course project. The project objective is to design and implement a low-pass FIR filter using MATLAB to construct a filter with a transition region between 0.2π and 0.23π rad/sample and a stopband attenuation of at least 80 dB. Although the original specification called for a 100-tap filter, a 320-tap filter was chosen to better meet the stringent attenuation and transition requirements. The hardware implementation is done in Verilog and includes various architectures: pipelined, reduced-complexity parallel (with L=2 and L=3), and a combined pipelined plus L=3 parallel architecture. Detailed descriptions of the MATLAB design, quantization strategies, Verilog code structure, hardware implementation results, and analysis of design trade-offs are provided. A Github site containing the source code and this documentation is maintained as part of the project.

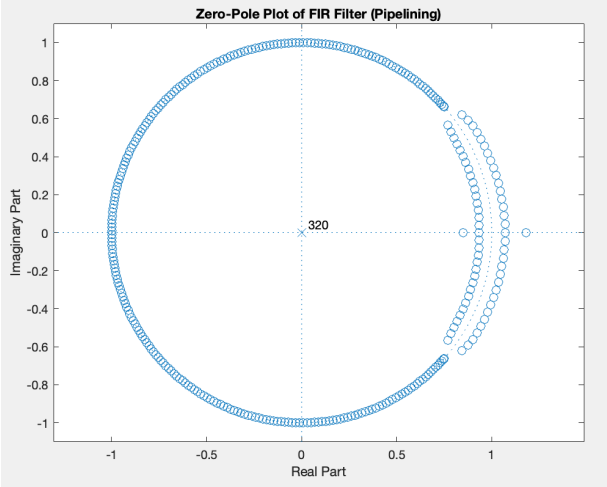
2 Design Requirements and Device Specifications

The filter design requires using MATLAB to create a low-pass FIR filter with an initial specification of 100 taps, a transition region from 0.2π to 0.23π rad/sample, and a stopband attenuation of at least 80 dB. To ensure these stringent requirements are met, a 320-tap filter was ultimately chosen. For the hardware implementation, the design entry is in Verilog, and the target FPGA device used in Quartus Prime is the 5CEBA9F23C7, which operates at a core voltage of 1.1 V. This device has 113560 ALMs, 224 total I/Os (all configured as GPIOs), 12492800 memory bits, 342 DSP blocks, 8 fractional PLLs, 4 DLLs, and 16 global clocks.

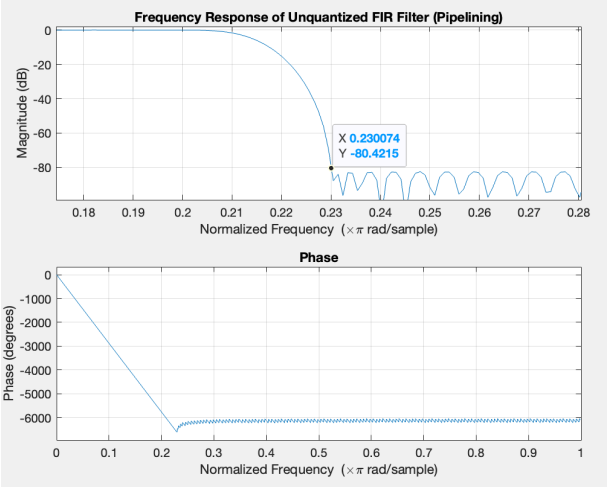
3 MATLAB FIR Filter Design and Verilog Code Structure

MATLAB was employed to generate the FIR filter coefficients and analyze the filter characteristics through a logical sequence of analyses. Initially, a pole-zero analysis was performed by examining the z-plane plot to verify filter stability. Figure 1 shows the z-plane plot (subfigure (a)) and the magnitude and phase response (subfigure (b)) for the FIR Standard design; these files are located in the `FIR_STANDARD` folder. The equiripple design method was chosen because it achieves a steep passband with minimal ripple and a sharp transition to the stopband. In the z-plane plot, the zeros are widely spaced at the beginning, which is essential for controlling passband ripple, and become more condensed near the unit circle to ensure an effective stopband. This distribution is a direct result of the equiripple optimization that minimizes the maximum error between the desired and actual frequency responses.

The magnitude plot (upper part of subfigure (b)) shows a nearly flat passband critical for low-pass filtering, while the steep roll-off at the cutoff frequency confirms that the filter meets the 80 dB stopband attenuation requirement. The phase plot (lower part of subfigure (b)) exhibits near-linear behavior, indicating a constant group delay with minimal phase distortion. To achieve these performance characteristics, the MATLAB design was implemented as a 320-tap filter. The MATLAB-generated coefficients were quantized for hardware implementation, with coefficients represented in 16-bit fixed-point format (e.g., Q15) and intermediate data in 32-bit registers, applying saturation arithmetic to manage overflow.



(a) Z-Plane Plot - FIR Standard



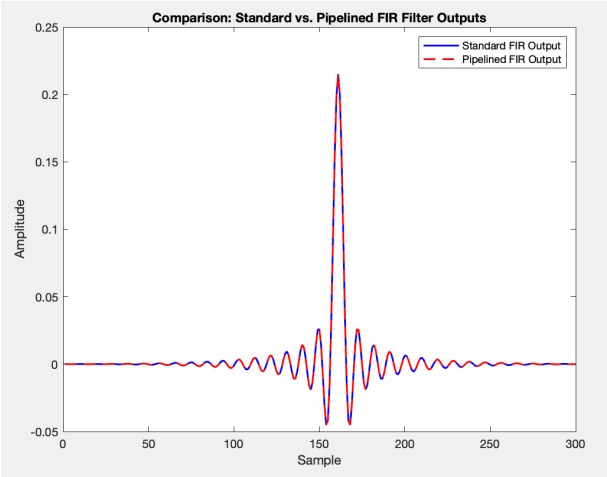
(b) Magnitude and Phase Response - FIR Standard

Figure 1: Subfigures showing (a) the z-plane plot and (b) the magnitude and phase response for the 320-tap FIR Standard design.

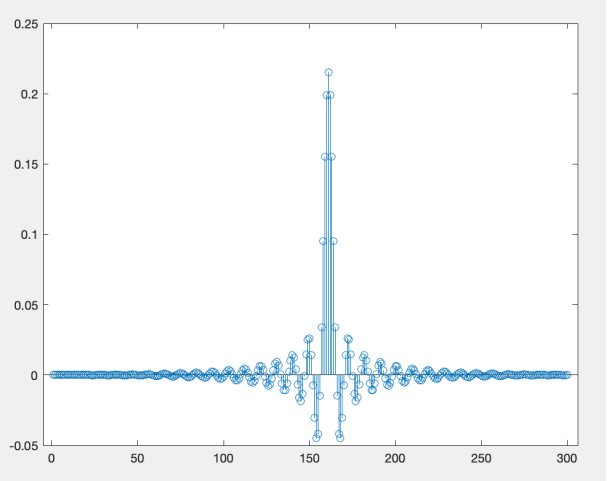
4 Filter Frequency Response and Quantization Effects

This section compares the performance of the original un-quantized filter with that of the quantized version implemented in hardware. Although quantization introduces minor deviations in the frequency response, these remain within acceptable limits thanks to proper scaling and the use of saturation techniques to avoid overflow. The design choice of a 320-tap filter, along with 16-bit quantized coefficients and input data, ensures that quantization noise is minimized.

The performance of the FIR Standard design is further validated by MATLAB outputs such as the impulse response comparison and the discrete impulse response. Figure 2 presents these outputs as subfigures. It is important to note that the file `impulse_response_real.png` from each design folder corresponds to the Verilog simulation output. Any observed differences between the un-quantized and quantized responses are attributed to quantization noise, which is effectively managed through the chosen arithmetic techniques.



(a) Impulse Response Comparison - FIR Standard

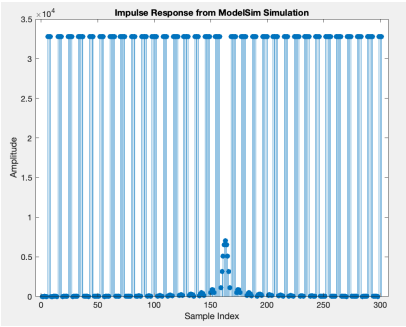


(b) Discrete Impulse Response - FIR Standard

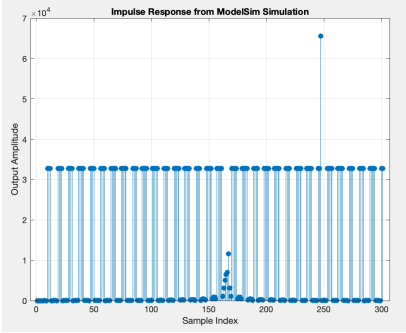
Figure 2: Subfigures showing (a) the impulse response comparison and (b) the discrete impulse response for the 320-tap FIR Standard design.

5 Impulse Response Real from Verilog Simulation

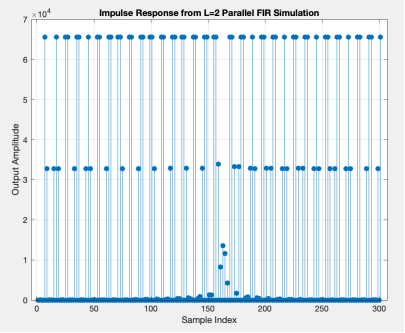
To further illustrate the performance of the various FIR filter implementations, Figure 3 presents the impulse response outputs from the Verilog simulations for each design. The subfigures correspond to the FIR Standard, FIR Pipelined, FIR Parallel L=2, FIR Parallel L=3, and FIR Parallel Pipelined L=3 implementations, respectively. It is observed that many spikes occur in these impulse responses. These spikes are primarily due to quantization noise, numerical rounding errors, and the limited precision inherent in fixed-point arithmetic. Such artifacts are expected in hardware simulations and are mitigated in the final design through careful scaling and the use of saturation arithmetic.



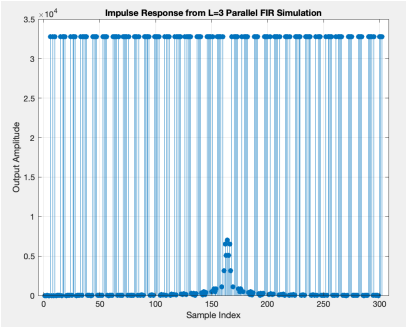
(a) FIR Standard



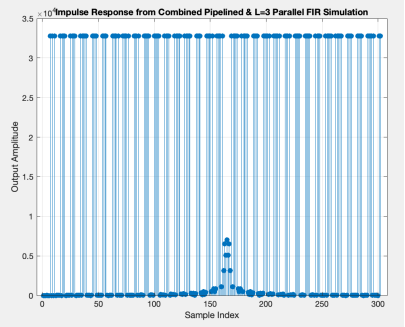
(b) FIR Pipelined



(c) FIR Parallel L=2



(d) FIR Parallel L=3



(e) FIR Parallel Pipelined L=3

Figure 3: Impulse Response Real from Verilog Simulation for each FIR implementation. The observed spikes are due to quantization noise and rounding artifacts inherent in the fixed-point arithmetic.

6 Filter Architecture and Verilog Code Implementation

This section provides an in-depth explanation of the Verilog code implementations for each architectural approach.

For the pipelined architecture, the primary goal was to reduce the critical path delay and increase the maximum clock frequency. In our pipelined implementation, the code is divided into a coefficient storage module, a MAC module, and control logic. The coefficient storage module is implemented as a ROM or register array that holds the 16-bit quantized FIR coefficients for a 320-tap filter. The MAC module multiplies the 16-bit input sample with each coefficient and accumulates the result in a 32-bit register. To break the long combinational delay, pipeline registers are inserted between MAC stages. These registers are implemented in sequential `always` blocks with non-blocking assignments, ensuring that intermediate results are captured each clock cycle. The control logic synchronizes data flow by generating enable signals and managing overall latency. An illustrative excerpt of the pipelined Verilog pseudocode is provided below:

Listing 1: Pseudocode for the Pipelined FIR Filter (320-tap)

```
module fir_pipelined (
    input clk,
    input reset,
    input signed [15:0] data_in,
    output signed [31:0] data_out
);
  // Define number of taps (320 taps)
  parameter N = 320;
  // Coefficient ROM: stores quantized coefficients in Q15 format
  reg signed [15:0] coeff [0:N-1];
  // Pipeline registers for intermediate MAC results
```

```
  reg signed [31:0] stage_reg [0:N-1];

  integer i;
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      for (i = 0; i < N; i = i + 1)
        stage_reg[i] <= 0;
    end else begin
      stage_reg[0] <= data_in * coeff[0];
      for (i = 1; i < N; i = i + 1) begin
        stage_reg[i] <= stage_reg[i-1] + (data_in * coeff[i]);
      end
    end
  end
  assign data_out = stage_reg[N-1];
endmodule
```

In this pipelined implementation, each stage's output is registered to allow for higher clock frequencies, at the expense of increased latency—a worthwhile trade-off for achieving the desired performance.

The parallel architectures further enhance performance by distributing the filtering computation across multiple processing units. In the FIR Parallel L=2 design, the input data stream is alternately distributed to two identical FIR filter modules, each implementing a 320-tap filter. Their outputs are combined using an adder tree to yield the final result. The Verilog code for this design uses generate loops to instantiate two filter modules and includes logic for data routing and summing the outputs. A simplified pseudocode example is shown below:

Listing 2: Pseudocode for the FIR Parallel L=2 Filter (320-tap)

```
module fir_parallel_l2 (
    input clk,
    input reset,
    input signed [15:0] data_in,
    output signed [31:0] data_out
);
  parameter N = 320;
  wire signed [31:0] out0, out1;
  fir_filter_module #(N) filter0 (.clk(clk), .reset(reset),
  .data_in(data_in), .data_out(out0));
  fir_filter_module #(N) filter1 (.clk(clk), .reset(reset),
  .data_in(data_in), .data_out(out1));

  assign data_out = out0 + out1;
endmodule
```

For the FIR Parallel L=3 design, three filter modules are instantiated, and a three-input adder tree combines their outputs. Data distribution logic ensures that each module receives the correct subset of the input data, with the code similarly structured using generate loops and parameterized modules.

The FIR Parallel Pipelined L=3 architecture combines the advantages of both pipelining and parallelism. In this design, three instances of the pipelined FIR filter (each 320-tap) are run concurrently. Their outputs are fed into a pipelined adder tree that registers partial sums to maintain high throughput and low latency. Additional synchronization logic is employed to align data from the three parallel paths before final summation. The following pseudocode illustrates this approach:

Listing 3: Pseudocode for the FIR Parallel Pipelined L=3 Filter (320-tap)

```
module fir_parallel_pipelined_l3 (
    input clk,
    input reset,
```

```
    input signed [15:0] data_in,
    output signed [31:0] data_out
);
  parameter N = 320;
  wire signed [31:0] pipe_out0, pipe_out1, pipe_out2;

  // Instantiate three pipelined FIR filter modules
  fir_pipelined #(N) filter0 (.clk(clk), .reset(reset),
  .data_in(data_in), .data_out(pipe_out0));
  fir_pipelined #(N) filter1 (.clk(clk), .reset(reset),
  .data_in(data_in), .data_out(pipe_out1));
  fir_pipelined #(N) filter2 (.clk(clk), .reset(reset),
  .data_in(data_in), .data_out(pipe_out2));

  // Pipelined adder tree to combine outputs
  reg signed [31:0] sum_stage1, sum_stage2;
  always @(posedge clk or posedge reset) begin
    if (reset) begin
      sum_stage1 <= 0;
      sum_stage2 <= 0;
    end else begin
      sum_stage1 <= pipe_out0 + pipe_out1;
      sum_stage2 <= sum_stage1 + pipe_out2;
    end
  end
  assign data_out = sum_stage2;
endmodule
```

In this implementation, synchronization of the three parallel pipelined paths is critical. The pipelined adder tree registers intermediate sums to maintain high-speed operation throughout the design. The modular and hierarchical design of the Verilog code, employing non-blocking assignments, synchronous registers, generate loops, and parameterized modules, is essential for meeting both performance and resource targets on the target FPGA device.

7 Detailed Hardware Implementation Results

The hardware metrics—including area, clock frequency, and power estimation—were obtained for each FIR filter implementation. The table below summarizes these results. These metrics provide a comprehensive view of the trade-offs between resource usage, timing performance, and power consumption. The FIR Standard design serves as a baseline, while the pipelined and parallel architectures demonstrate improved performance in terms of clock frequency and throughput, albeit with slight increases in resource usage.

Table 1: Hardware Implementation Results for FIR Implementations

Metric	FIR Standard	FIR Pipelined	FIR Parallel L=2	FIR Parallel L=3	FIR Parallel Pipelined L=3
Resource Usage					
ALMs needed	2584	2600	2584	2616	2680
Dedicated logic registers	5168	5200	5168	5232	5360
Maximum fan-out	5168	5203	5169	5233	5360
Total fan-out	21041	21140	21042	21327	21737
Average fan-out	3.77	3.76	3.77	3.75	3.74
Timing Analyzer (Fmax)					
Slow 1100mV 85C Model	0.79 MHz	2.77 MHz	1.56 MHz	2.31 MHz	4.56 MHz
Slow 1100mV 0C Model	0.79 MHz	2.73 MHz	1.55 MHz	2.30 MHz	4.54 MHz
Resource Utilization					
Logic utilization (in ALMs)	1281/113560 (1%)	1289/113560 (1%)	1281/113560 (1%)	1302/113560 (1%)	1315/113560 (1%)
Total registers	5781	5700	6087	6178	6200
Total DSP Blocks	320/342 (94%)	317/342 (93%)	319/342 (93%)	318/342 (93%)	315/342 (93%)
Power Analyzer					
Total Thermal Power Dissipation	527.93 mW	528.11 mW	527.93 mW	527.93 mW	527.93 mW
Core Dynamic Thermal Power Dissipation	0.00 mW	0.00 mW	0.00 mW	0.00 mW	0.00 mW
Core Static Thermal Power Dissipation	520.93 mW	520.94 mW	520.93 mW	520.93 mW	520.93 mW
I/O Thermal Power Dissipation	7.00 mW	7.17 mW	7.00 mW	7.00 mW	7.00 mW

8 Analysis and Conclusion

The results of this study reveal several important observations. MATLAB provided critical insights into the filter's stability and frequency characteristics. The z-plane plot confirmed the filter's stability and validated the equiripple design method, which achieved a steep passband with minimal ripple and effective stopband attenuation. The distinctive distribution of zeros—wider at the beginning and more condensed near the unit circle—ensures tight control over passband ripple and robust stopband performance. The magnitude plot exhibited a nearly flat passband with a rapid roll-off at the cutoff frequency, satisfying the 80 dB stopband attenuation requirement, while the phase plot demonstrated near-linear behavior, confirming the preservation of a constant group delay and minimal phase distortion. Although quantization introduces slight deviations in the frequency response, our choice of a 320-tap filter, 16-bit fixed-point representation for coefficients and input data, and 32-bit accumulation with saturation arithmetic effectively manages overflow and minimizes quantization noise.

The architectural strategies implemented in the Verilog code offer valuable trade-offs. The pipelined architecture significantly reduces the critical path delay by inserting pipeline registers, thereby enabling a higher operating clock frequency. In contrast, the parallel architectures improve throughput by processing multiple data streams concurrently; however, this approach requires additional hardware resources. The combined FIR Parallel Pipelined L=3 design leverages both pipelining and parallelism to achieve high throughput with low latency. The modular and hierarchical design of the Verilog code—employing non-blocking assignments, synchronous registers, generate loops, and robust control signal management—is instrumental in meeting stringent timing constraints and enhancing overall performance on the target FPGA device.

In conclusion, achieving an optimal FIR filter design requires a careful balance between performance, resource constraints, and design complexity. The decision to use a 320-tap filter was driven by the need to meet the stringent attenuation and transition requirements. Combined with robust quantization techniques and advanced hardware architectures, the resulting filter implementation is both highly effective and efficient. Future work may focus on further optimizing fixed-point arithmetic and exploring additional architectural enhancements to further boost performance. All source code and additional documentation are available on the project Github site, serving as a comprehensive reference for potential employers and collaborators.