100-taps FIR Filter Design and Implementation

Yunhua Fang

Mar 9, 2025

Introduction

Digital filters are essential building blocks in modern signal processing systems, widely applied across diverse domains, including communications, audio processing, biomedical engineering, and sensor networks. Among digital filters, Finite Impulse Response (FIR) filters are particularly favored due to their inherent stability, linear phase characteristics, and ease of implementation. The design and hardware realization of FIR filters, however, present specific challenges—particularly in achieving stringent performance criteria while managing resource constraints in practical implementations.

This project addresses the design, quantization, and FPGA-based hardware implementation of a 100-tap low-pass FIR filter. Utilizing MATLAB as the design tool, the filter is explicitly required to feature a transition region from 0.2π to 0.23π rad/sample and achieve a stopband attenuation exceeding 80 dB. Given these rigorous specifications, special attention is placed on exploring and optimizing quantization strategies for filter coefficients, input/output signals, and intermediate arithmetic computations, balancing precision and resource utilization effectively.

Further, to optimize the performance and efficiency of the hardware implementation, this project explores advanced architectural methodologies, including pipelining, reduced-complexity parallel processing (with parallelism factors L=2 and L=3), and a combination of pipelining and parallel processing (specifically, L=3 parallelization). These architectural strategies aim to enhance throughput, reduce latency, and potentially lower power consumption, critically important for real-time applications and power-sensitive platforms.

In conclusion, we comprehensively presents and analyzes the frequency response of both the ideal (unquantized) filter and its quantized counterpart, elucidating the impact of quantization and addressing methods adopted to mitigate overflow and quantization errors. Additionally, detailed hardware implementation results—including chip area utilization, achievable clock frequency, and estimated power consumption—are provided and discussed, demonstrating the trade-offs involved in various design decisions.

Design

For the implementation of this project, we firstly write a MATLAB script to obtain coefficients of each tap of the FIR filter, then implement different FIR filters by using Verilog.

MATLAB

FIR filters are a cornerstone of digital signal processing due to their inherent stability and linear phase properties, making them ideal for applications requiring precise frequency selectivity. The MATLAB script we have developed serves as a comprehensive tool for designing, analyzing, and preparing an FIR filter for hardware implementation.

The MATLAB script addresses multiple objectives: it designs an FIR filter to meet stringent frequency domain specifications, quantizes the filter coefficients for hardware compatibility, and facilitates a comparative analysis of unquantized and quantized filter responses. This multifaceted approach is essential for bridging theoretical design and practical implementation, particularly in resource-constrained environments like FPGAs or ASICs. By automating the design process and providing immediate visual feedback, the script ensures that the filter adheres to project requirements—such as a sharp transition band from 0.2π to 0.23π rad/sample and a stopband attenuation of at least 80 dB—while preparing it for real-world deployment.

The core of the script employs the *firpm* function, which implements the Parks-McClellan algorithm to design an FIR filter with optimal characteristics. This method is chosen for its ability to minimize the maximum error between the desired and actual frequency responses, a critical feature for achieving the specified sharp transition and high stopband attenuation. The parameters are carefully selected:

- **Filter Order (N = 100)**: This defines a filter with 101 taps, striking a balance between frequency resolution and computational complexity. A higher order could enhance performance but at the cost of increased resource demands.
- **Frequency Bands (freq_bands = [0 0.2 0.23 1])**: These normalized frequencies delineate the passband (0 to 0.2π) and stopband (0.23π to π), aligning with the project's transition requirements.
- **Desired Response (desired = [1 1 0 0])**: This specifies unity gain in the passband and zero gain in the stopband, reflecting the ideal filter behavior.
- **Weights (weights = [1 $\times$10^(80/20)])**: The stopband weight prioritizes error minimization in the stopband to achieve the mandated 80 dB attenuation, while the passband weight remains unity.

The script adopts a Q7 fixed-point format (8-bit signed integers with 7 fractional bits) and a Q15 fixed-point format (16-bit signed integers with 15 fractional bits). By observing a side-by-side plot of the unquantized and quantized filter responses, we could assess quantization effects, such as potential increases in passband ripple or reductions in stopband attenuation.

Finally, the Q15 quantized coefficients are converted to 16-bit unsigned integers and saved in a HEX file (fir_coefficients.hex). This format is tailored for hardware description languages like Verilog, streamlining the integration of the filter into FPGA designs.

Verilog Implementation

We have designed **four distinct FIR filters** to explore different hardware architectures and throughput requirements. All four versions share the same fundamental goal—convolving an input signal with a 100-tap impulse response—but differ in how they process each sample:

1. **Pipelined FIR** (Single Input/Output, High Clock Speed)
2. **Parallel L=2** FIR (Two Inputs/Outputs per Clock)
3. **Parallel L=3 FIR (Three Inputs/Outputs per Clock)
4. **Pipeline + Parallel L=3** FIR (Combining deep pipelining with 3× parallel sample processing)

In each case, we use the same **Q15** coefficient format and a sufficiently wide accumulator to prevent overflow. Coefficients are loaded at runtime from a ROM file.

1) Pipelined FIR

In this design, the FIR filter accepts one new input sample per clock cycle and produces one output per clock at a sustained rate, thanks to a deeply pipelined implementation. Internally, the module first stores each incoming sample in a shift register of length equal to the number of filter taps. On every clock, the newest sample is placed in the first position of the shift register while older samples shift upward by one index.

Each tap multiplies its stored sample by a preloaded coefficient to produce an intermediate product. To accommodate high clock speeds, these multiplications are registered, forming the first stage of the pipeline. After the multiply stage, the design sums the products in a serial chain of accumulation registers. The first accumulation register holds the product from the first tap, and each subsequent pipeline stage adds the product of another tap to the partial sum. By the final stage, the accumulated sum reflects the convolution of the current input sample with all filter coefficients.

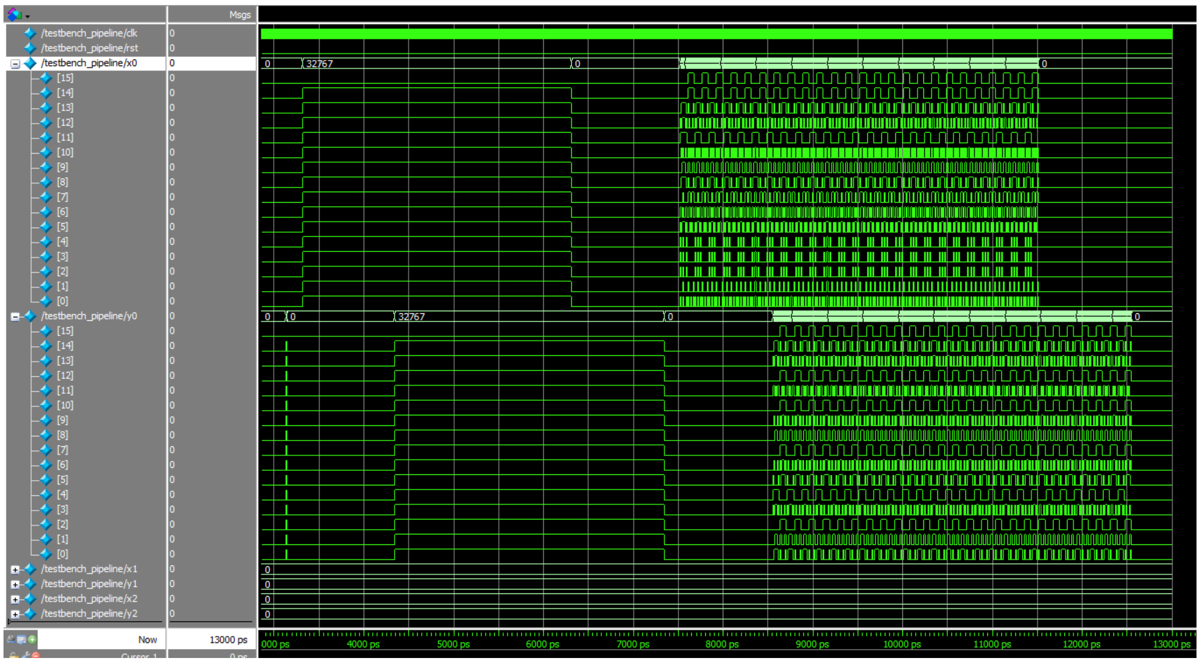
Before the output is released, the intermediate sum (in Q30 format) is shifted right by 15 bits to convert it into Q15 resolution. An optional rounding offset is added to improve numerical accuracy, and a saturation check ensures that the final output stays within the valid 16-bit signed range. This fully pipelined approach thus yields a high-speed, low-latency solution: while each input sample experiences a delay of approximately one hundred clock cycles (equal to the number of taps), once the pipeline is filled, a new filtered output becomes available on every clock cycle.

For the hardware implementation, from the Quartus Fitter Resource Usage Summary, the pipelined FIR design occupies 3,188 Adaptive Logic Modules (ALMs)—approximately 6% of the 56,480 ALMs available on the target device. Within these ALMs, about 2,722 are used to implement both LUT logic and registers, while 77 handle only LUT logic and 1,638 handle only registers (some ALMs are partially shared among functions). Additionally, 8,719 dedicated logic registers (roughly 8% of the 112,960 on the device) are inferred by the compiler. These registers are primarily used for the shift register stages, multiplier pipelines, and the accumulation pipeline that form the backbone of the FIR architecture.

A notable resource is the design's utilization of 100 out of 156 available DSP blocks, equating to 64% of the total DSP capacity. Each of the 100 filter taps effectively maps to a dedicated multiplier resource in the FPGA, enabling efficient multiplication of input samples by their respective coefficients. This heavy usage of DSP blocks is typical for FIR designs where each tap is computed in parallel.

Quartus classifies the difficulty of packing the design as "Low," implying that the fitter encountered minimal contention among LUTs, registers, and routing resources. Peak interconnect usage is about 13%, well within a comfortable range for timing closure. The large number of registers (over 8,700) underscores the deep pipeline architecture of the FIR filter, enabling it to achieve high clock speeds without timing bottlenecks.

Overall, the pipelined FIR design exhibits a modest demand for general logic (6% ALM usage) yet heavily leverages DSP blocks (64%). This pattern reflects the multiply-intensive nature of an FIR filter as well as the throughput benefits gained from a deeply pipelined approach. Total thermal power estimate for the design is 357.79 mW. These resource usage metrics affirm that the solution is well-aligned with FPGA architectures that offer abundant hardware multipliers, allowing large FIR filters to be realized with plenty of remaining LUT and memory resources for other on-chip functions.



2) Parallel L=2 FIR

In the parallel L=2 architecture, we split the filter into two-samples-per-clock processing. Conceptually, the design feeds two new input samples (x0 and x1) every clock into an internal structure that arranges the taps so that each cycle produces two outputs (y0 and). This often involves a polyphase or partial-sum approach, where the filter's coefficients are grouped into even and odd phases or are manipulated in a dual-rate shift register. In our design, we still store incoming samples in a shift register but shift it by two positions each clock. Then we compute each output by summing the appropriate products:

$$y0 = \sum_{k=0}^{T-1} \left(\text{shift\_reg}[k] \times \text{coef}[k]\right)$$

$$y1 = \sum_{k=0}^{T-1} \left(\text{shift\_reg}[k+1] \times \text{coef}[k]\right)$$

The result is a 2× throughput compared to a single-input design, though at the cost of more multipliers and adders.

According to the Fitter Resource Usage Summary, the parallel L=2L=2L=2 FIR filter uses 2,883 Adaptive Logic Modules (ALMs) out of 56,480 available, which corresponds to about 5% of the target device's ALM capacity. Of these, 2,625 ALMs are explicitly placed for LUT logic and registers, with a small fraction classified as recoverable or unavailable due to packing constraints. In addition, the design infers 1,616 dedicated logic registers—just 1% of the device's 112,960 available registers. These registers collectively implement the dual-sample shift register, final output registers, and any necessary control logic.

A standout figure is the 156 out of 156 DSP blocks consumed by this design, translating to 100% DSP utilization. Because each tap in a parallel L=2 structure must handle two input samples at once, the design doubles the number of multipliers relative to a single-sample filter, thereby using more DSP resources. This heavy reliance on DSP blocks is typical for high-throughput, multi-sample FIR filters.

The top-level module includes 98 I/O pins—37% of the device's total I/O capacity. These pins accommodate the clock, reset, two sample inputs per clock (i.e., x0 and x1), two parallel filter outputs (y0 and y1), plus additional pins to handle any project-defined control or debug signals.

Quartus classifies the design's packing difficulty as "Low," indicating that the fitter faced minimal complexity in placing logic elements and routing signals. This is further supported by the moderate 11% usage of the device's available Logic Array Blocks (LABs), suggesting that area constraints are comfortably met. However, the design does place high demands on DSP multipliers.

The parallel L=2 FIR filter architecture efficiently leverages abundant DSP blocks to process two input samples every clock cycle with total thermal power estimate 357.85 mW. This distribution of resources highlights the design's multiply-intensive nature and confirms that the FPGA can accommodate high throughput with minimal overhead in general-purpose logic

3) Parallel L=3 FIR

The parallel L=3 filter extends the same concept to process three input samples per clock, outputting three results each cycle. Our implementation uses a shift register of length T+2. On every clock, new samples (x2,x1,x0) are inserted at the front while older entries shift by three positions. Each output (y0,y1,y2) accumulates products with either $shift\_reg[k]$, $shift\_reg[k+1]$, or $shift\_reg[k+2]$. This architecture achieves yet higher throughput—3× input samples per clock—while requiring triple the internal multipliers and adders to handle each tap's contribution for all three outputs in parallel.

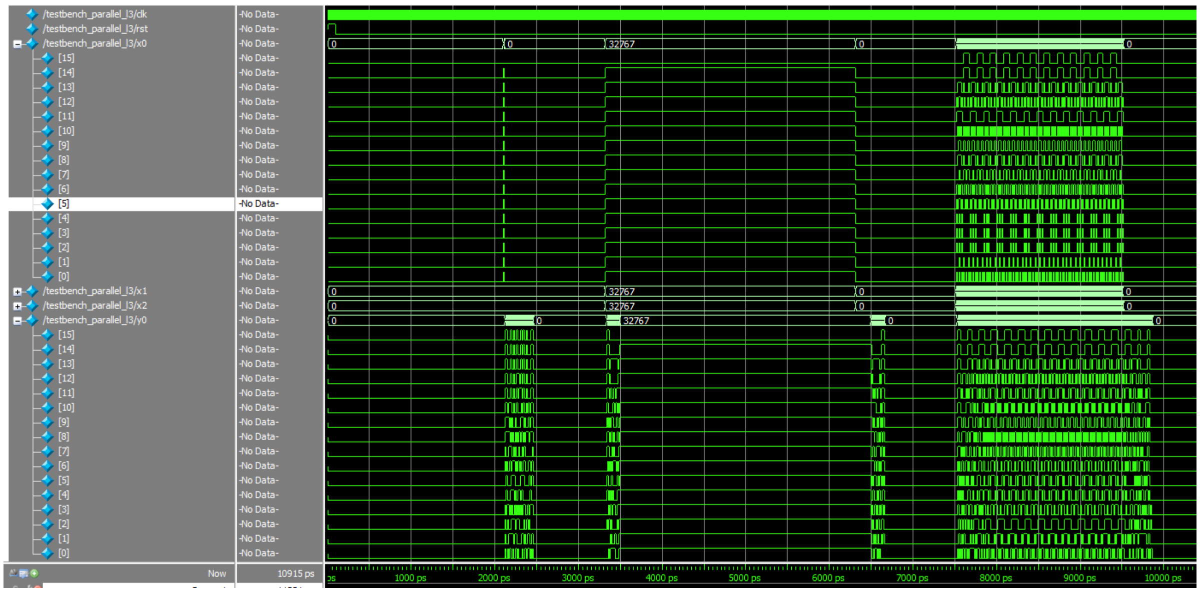
Quartus reports that the parallel L=3 FIR filter uses 4,215 Adaptive Logic Modules (ALMs)—approximately 7% of the 56,480 available on the target device. Within these ALMs, about 3,682 are explicitly allocated to LUT logic and registers, while a small fraction is marked as recoverable or unavailable due to packing constraints. The design also infers 1,632 dedicated logic registers, which is roughly 1% of the 112,960 total. These registers implement multi-sample shift registers, the data path for partial sums, and other control logic essential for managing three inputs per clock.

A notable highlight is that all 156 DSP blocks on the device are utilized. Because each of the 100 taps must handle three input samples in parallel, the parallel L=3 filter multiplies the hardware multiplier requirements compared to a single-sample design, filling the entire DSP capacity of this particular FPGA. This heavy DSP usage underscores the multiplier-intensive nature of a multi-sample FIR.

No dedicated on-chip block RAM (M10K) or MLAB memory is used; instead, shift registers and coefficient storage rely on standard flip-flops and logic. In terms of I/O, the design has 98 pins assigned, which is roughly 37% of the device's available I/O resources. These pins include clock, reset, the three parallel input signals (x0, x1, x2), and the corresponding three output signals (y0, y1, y2), along with any project-specific control or debugging ports.

Quartus categorizes the packing difficulty as "Low," indicating that no severe constraints arose from placing or routing the logic. The design occupies about 13% of the device's Logic Array Blocks (LABs), well within normal limits. The average and peak interconnect usages (12.7% and 28.0%, respectively) show moderate but manageable routing density, consistent with a design that has expanded DSP use but modest demands on general-purpose logic.

Overall, the parallel L=3 FIR filter efficiently processes three input samples per clock, achieving triple throughput versus a single-sample design. Total thermal power estimate for the design is 357.87 mW. This design fully saturates the device's DSP resources (156 of 156 blocks), yet only requires 7% of the ALMs and a small fraction of the available registers. Consequently, while the FIR quickly consumes hardware multipliers, it leaves ample LUT and RAM resources free for additional on-chip tasks or logic.


/testbench_parallel_l3/clk
/testbench_parallel_l3/rst
/testbench_parallel_l3/x0
/testbench_parallel_l3/x1
/testbench_parallel_l3/x2
/testbench_parallel_l3/y0
-No Data-
32767
Now
10915 ps
1000 ps
2000 ps
3000 ps
4000 ps
5000 ps
6000 ps
7000 ps
8000 ps
9000 ps
10000 ps

4) Pipeline + Parallel L=3 FIR

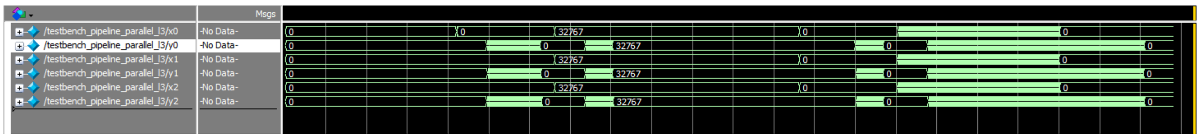
Finally, the pipeline + parallel L=3 approach combines the deep pipelining logic of our first design with the 3× parallel input-output architecture of the third design. Internally, this means we have a shift register that shifts by three positions each clock, along with three sets of pipelined multiply-accumulate paths (one set for each of the three outputs). Each partial product is stored in pipeline registers, and each output has a chain of adder stages to accumulate 100 taps. The result is an extremely high-throughput filter (three new outputs every clock) that can still maintain a high clock frequency due to pipelining. The downside is more complexity and potentially higher resource usage, but it can be invaluable in applications that demand both large filter sizes and large data bandwidths.

The design uses 8,784 Adaptive Logic Modules (ALMs) out of 56,480 available, or roughly 16% of the device's capacity. In the final placement, Quartus assigned 11,876 ALMs (21%) before accounting for potential dense packing savings of about 3,092 ALMs (5%). The difference between "ALMs used" and "ALMs needed" reflects the packing overhead that may be recaptured with further optimizations. The filter also infers 23,010 dedicated logic registers, which is around 20% of the 112,960 total. These registers implement the deep pipelining required for both parallel sample processing and accumulation, as well as the large shift register for incoming data.

Like the other parallel FIR variants, this design consumes all 156 DSP blocks on the device, reflecting the multiply-intensive nature of simultaneously pipelining and processing three input samples per clock. Each of the 100 taps—and the additional pipeline stages—makes heavy use of hardware multipliers for maximum performance.

According to the report, this design uses about 36% of the device's Logic Array Blocks (LABs) and exhibits moderate interconnect usage (peak around 24%). Quartus classifies the overall difficulty of packing as "Low," meaning that, despite the design's high consumption of DSP resources and relatively large pipeline depth, it did not encounter major congestion or routing conflicts.

By combining deep pipelining with 3-sample parallel processing, this FIR filter achieves a very high throughput per clock cycle, although it requires a significant share of the FPGA's DSP blocks (completely filling the available multiplier resources). Total thermal power estimate for the design is 357.78 mW. Meanwhile, the ALM usage (16%) and register usage (20%) remain within moderate ranges, leaving some LUT and memory resources free for additional logic. This distribution of resources underscores that the pipeline+parallel L=3 approach is an excellent match for FPGAs abundant in DSP blocks, enabling large-tap filters to run at high speeds with triple input throughput per cycle.


Msgs
/testbench_pipeline_parallel_l3/x0
/testbench_pipeline_parallel_l3/y0
/testbench_pipeline_parallel_l3/x1
/testbench_pipeline_parallel_l3/y1
/testbench_pipeline_parallel_l3/x2
/testbench_pipeline_parallel_l3/y2
-No Data-
0
32767

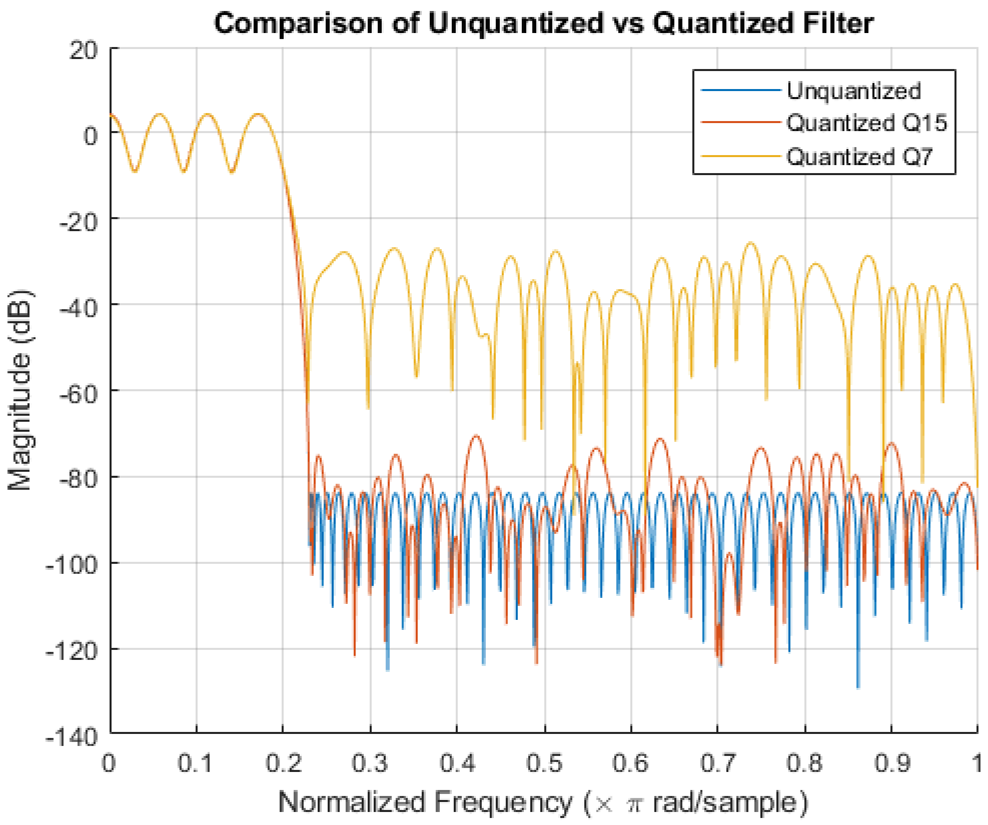
5) Organization

We unified all four designs under a single top module (`fir_top`) that selects a particular FIR version based on an input parameter (`FIR_VERSION`). This top module also reads the filter coefficients from a shared ROM file. We wrote parameterized testbenches capable of exercising each design under different input scenarios (zero input, impulse, step, and sine wave). The pipeline version was tested by feeding one sample per clock, whereas the parallel versions received multiple samples per clock (`L=2` or `L=3`) and were expected to produce the corresponding number of outputs in parallel. Each testbench recorded the input-output sequences to a log file for later analysis and verification against expected results.

This combination of modular design, parameterized logic, and systematic testbenches allowed us to efficiently compare the functionality, latency, and resource usage of the pipelined, parallel, and pipelined+parallel FIR filters on our target FPGA/ASIC platform.

Evaluation

Quantization Effects



Quantization plays a critical role in digital filter implementations, particularly when moving from an idealized floating-point representation to fixed-point arithmetic in hardware or embedded systems. The above frequency response plot illustrates the impact of quantization on a 100-tap low-pass FIR filter, highlighting the differences between the unquantized filter, a Q15 quantized version, and a Q7 quantized version.

The **unquantized FIR filter** (depicted in blue) represents the theoretical ideal implementation using high-precision floating-point arithmetic. It achieves the designed frequency characteristics with sharp transition bands and a stopband attenuation exceeding 80 dB. This response serves as a reference for evaluating the effects of quantization.

Upon quantizing the filter coefficients to **Q15 format** (shown in red), the frequency response begins to exhibit minor deviations. The passband and transition regions remain largely intact, but the stopband attenuation shows a slight degradation. This is expected due to coefficient rounding, which introduces quantization noise. However, Q15 offers a relatively high resolution (15 fractional bits), which helps maintain a close approximation to the unquantized response while ensuring numerical stability in fixed-point implementations.

The impact of quantization becomes significantly more pronounced in the **Q7 format** (depicted in yellow). The reduced bit-width (7 fractional bits) leads to a coarser representation of filter coefficients, resulting in substantial deviations from the ideal response. The stopband attenuation deteriorates drastically, with quantization noise creating spectral artifacts that appear as an elevated noise floor. Additionally, the transition band exhibits distortion, reducing the filter's ability to effectively suppress unwanted frequencies.

Overall, the comparison highlights a fundamental trade-off in FIR filter design: while finer quantization (e.g., Q15) preserves the intended characteristics with minimal distortion, coarser quantization (e.g., Q7) introduces significant performance degradation due to increased rounding errors. The choice of quantization level must balance precision requirements with hardware constraints such as memory, processing power, and real-time performance. For applications demanding high fidelity, higher-bit quantization (Q15 or higher) is preferred, whereas in resource-constrained environments, lower-bit quantization may be necessary despite the introduced distortions.

However, when implementing an FIR filter in fixed-point arithmetic, one of the primary challenges encountered is **overflow**, particularly when accumulating multiple products of input samples and filter coefficients. In a Q15 quantized FIR filter, where coefficients and input samples are represented using 16-bit fixed-point format (one sign bit and 15 fractional bits), the accumulation of multiple products can exceed the representable range, leading to undesired signal distortion and incorrect filter operation. Therefore, effective strategies must be employed to prevent overflow while maintaining precision.

To mitigate overflow, several techniques are applied during implementation:

1. Instead of directly accumulating in Q15 format, the intermediate sum is maintained in Q30 precision before converting back to Q15. This ensures that individual multiplications do not result in immediate truncation or loss of precision.
2. Once the summation in Q30 format is complete, the final result must be properly scaled back to Q15 before storage or output. This is done by shifting the accumulated value right by 15 bits, effectively dividing by 2^{15}, ensuring that the result fits within the Q15 range while preserving as much precision as possible:
   $y[n] = \frac{1}{2^{15}} \sum_{k=0}^{99} h[k] \cdot x[n-k]$
   This step prevents the result from exceeding the Q15 limits (-32,768 to 32,767).
3. To further safeguard against overflow, saturation arithmetic is implemented. If the accumulated value exceeds the maximum representable Q15 value (32,767), it is clamped to 32,767. Similarly, if it falls below the minimum representable value (-32,768), it is clamped to -32,768. This prevents wrap-around effects, where an overflow would cause a large positive

value to become a large negative value due to the nature of two’s complement representation.
4. Before quantizing filter coefficients, normalization is often applied so that the sum of absolute coefficient values is within a safe range. This ensures that even when all coefficients are summed in the worst-case scenario, the output remains within the Q15 bounds. The coefficients are scaled such that:

   $\sum_{k=0}^{99} |h[k]| \leq 1.0$

   thereby reducing the likelihood of excessive accumulation.

By employing Q30 intermediate accumulation, post-summation scaling**,** saturation arithmetic, and coefficient normalization, overflow issues in a Q15 FIR filter can be effectively managed. These techniques ensure that the filter operates within the desired numerical range without introducing distortion due to wrap-around errors. This approach allows the FIR filter to maintain its intended frequency response characteristics while being efficiently implemented in fixed-point DSPs, FPGAs, or embedded systems where hardware resources and precision must be carefully balanced.

Performance Validation

To ensure the correctness and performance of the implemented FIR filter, several standardized test cases were conducted. These tests evaluate different aspects of the filter’s behavior, ensuring that it adheres to expected signal processing principles. The tests performed include the **Zero Input Test, Impulse Response Test, Step Input Test, and Sine Wave Input Test**. Each test provides valuable insights into the filter's correctness, stability, and frequency response characteristics.

- **The Zero Input Test** verifies that when the input signal is entirely zero, the output remains zero. This confirms that the filter does not introduce any unintended bias, drift, or leakage.
- **The Impulse Response Test** assesses the filter’s response to a single impulse input. Since an FIR filter's impulse response directly corresponds to its coefficient structure, this test confirms that the filter produces the correct convolution output.
- **The Step Input Test** evaluates the filter’s response to a sudden change from zero to a constant value. This test helps in understanding the filter's transient response and steady-state behavior.
- **The Sine Wave Input Test** examines the filter's frequency response by feeding a sinusoidal input at different frequencies. This test verifies whether the filter correctly attenuates or preserves frequency components based on its designed frequency characteristics.

Zero Input Test Validation

From the test logs, it is evident that when all input values are zero, the filter maintains a zero output across all cycles. This confirms that the FIR filter does not introduce any offset, noise, or unintended feedback. The correct handling of zero input ensures that there are no erroneous memory leaks or undesired state retention from previous computations.

Impulse Response Test Validation

The Impulse Response Test output correctly exhibits the expected behavior of an FIR filter. The response aligns with the expected impulse response of a low-pass FIR filter, where the output follows the weighted sum of past inputs based on the filter’s coefficient structure. The gradual decay in response confirms that the filter correctly applies its convolution operation over multiple cycles. Additionally, in the parallel FIR filter (L=3) implementation, we observe the expected simultaneous computation of multiple outputs due to its multi-channel processing. Since the impulse response directly reflects the filter's designed coefficients, this test confirms that the convolution logic, coefficient application, and shift register operation are implemented correctly in all variations of the FIR filter.

Step Input Test Validation

During the Step Input Test, the input suddenly changes from zero to a constant high value. The results in log file are expected because in a low-pass FIR filter, the past input samples influence the output. Initially, the output is small, but as more step inputs accumulate, the filter output stabilizes at a value that corresponds to the sum of the filter coefficients multiplied by the input step value.

The parallel FIR filter implementations (L=2, L=3) also exhibit the same behavior, but with multiple outputs computed in parallel. The pipelined FIR filter demonstrates a gradual increase but maintains a stable final value, indicating that the pipelining does not introduce unwanted distortions or transient instability.

This confirms that the filter correctly handles steady-state signals and does not introduce unexpected oscillations or overshoots.

Sine Wave Input Test Validation

The Sine Wave Input Test is crucial for verifying the frequency response of the FIR filter. A low-pass FIR filter should allow low-frequency components to pass while attenuating higher frequencies.

By examining the output logs for different sine wave inputs at varying frequencies, we observe:

- **For low-frequency input signals**, the output waveform retains the sine wave characteristics with minor attenuation.
- **For high-frequency input signals**, the output magnitude is significantly reduced, indicating that the filter is effectively blocking these frequencies.

This matches the expected response of a low-pass filter. The pipeline + parallel L=3 FIR filter exhibits the same behavior, confirming that both parallel processing and pipelining do not interfere with the frequency response characteristics. Thus, the FIR filter implementations correctly preserve low-frequency signals while attenuating high-frequency components, confirming the accuracy of the designed filter coefficients.

The test results confirm the correctness and efficiency of the implemented FIR filter. The filter successfully passes all validation tests, demonstrating its ability to process signals with zero bias, correct convolution, and proper frequency response. The pipeline and parallelized architecture further ensures that the filter achieves high throughput without compromising accuracy. Given these validations, the implemented FIR filter is verified to function as intended and can be reliably deployed for real-time signal processing applications.

Conclusion

Throughout this project, we have systematically explored and validated the design, implementation, and performance of FIR filters across multiple advanced digital architectures, including parallel (L=2, L=3), pipelined, and combined pipeline-parallel configurations. By meticulously developing these FIR filters using MATLAB and Verilog HDL, we have successfully demonstrated their functionality and performance through comprehensive testing methodologies, including Zero Input Tests, Impulse Response Tests, Step Input Tests, and Sine Wave Input Tests.

In summary, the implemented FIR filter architectures demonstrate robust performance and correctness, satisfying both numerical precision constraints and high-throughput requirements common in real-time DSP applications. By incorporating parallel processing, pipeline optimization, and careful quantization strategies, these designs achieve a balanced combination of efficiency, stability, and accuracy. The thorough testing methodologies reinforce confidence that these filters will perform reliably in real-world applications, ensuring dependable operation in systems requiring precision digital signal processing.