FPGA-Based FIR Filter Design and Simulation

Weiqiang Zhang [1, *] and Zihan Wang [2]

[1] Department of Integrated Circuit, Anhui Polytechnic University, Wuhu, China

[2] Department of Electrical Engineering, Beibu Gulf University, Guangxi, China

* Corresponding Author Email: zwq2656662919@outlook.com

Abstract. This paper proposes an optimized design approach for a Finite Impulse Response (FIR) digital filter, leveraging parallel and pipelined architectures to enhance performance, with simulation results validated in ModelSim. FIR filters are crucial components in digital signal processing (DSP), commonly applied in communication systems, audio processing, and other real-time signal applications. As real-time systems demand high computational speed, the challenge of implementing FIR filters efficiently in hardware, particularly in resource-limited environments, becomes increasingly critical. This study addresses these challenges by incorporating parallel processing to enable simultaneous computation across multiple filter stages, significantly boosting processing speed. Additionally, a pipelined architecture is employed to decompose the filter's operations into multiple stages, reducing latency and further improving throughput. The proposed design is implemented using Verilog Hardware Description Language (HDL), ensuring flexibility and scalability for various hardware platforms. The performance and functionality of the design are thoroughly verified through simulations in ModelSim, demonstrating that the parallel and pipelined FIR filter achieves an optimal trade-off between processing speed and resource consumption. The results show that the design is not only effective for high-frequency applications but also well-suited for embedded and real-time systems, where both speed and resource efficiency are essential. Future work will focus on exploring additional optimizations for filter architectures to further enhance system performance and expand the design's applicability in dynamic and resource-constrained environments.

Keywords: FIR filter, parallel processing, pipelined architecture, digital signal processing, real-time systems, ModelSim simulation, Verilog HDL, hardware optimization, embedded systems, resource utilization.

1. Introduction

Finite Impulse Response (FIR) filters play a vital role in digital signal processing (DSP), particularly in communication systems, audio processing, and image processing, due to their inherent stability and linear phase characteristics [1,2]. As real-time signal processing demands increase, traditional software-based FIR implementations face challenges in meeting performance requirements under constrained hardware conditions [2]. This paper presents an optimised design methodology for FIR filters, leveraging parallel and pipelined architectures to address these limitations effectively [1,3]. The proposed design executes multiple filters taps simultaneously by employing parallel architecture, significantly enhancing processing speed. The pipelined structure divides computations into sequential stages, allowing overlapping operations and improving system throughputxu. Implemented using Verilog HDL, the design was verified through simulation in Model Sim, which demonstrated its robustness and high efficiency in handling real-time signal processing tasks. The results indicate that the proposed design achieves a compelling balance between processing speed, resource utilization, and performance. Compared to traditional FIR designs, this approach reduces latency and computational overhead, enabling efficient operation even in resource-constrained environments. The combination of parallelism and pipelining not only improves filter throughput but also aligns with the demands of modern embedded systems for real-time processing capabilities. This study provides valuable insights into optimizing FIR filters for high-performance applications, highlighting the potential for further advancements in adaptive filtering techniques and hardware-efficient FPGA implementations for next-generation DSP solutions.

2. From Theory to Simulation Waveform Implementation

2.1. Experimental Setup

In this study, the FIR filter is designed using Verilog HDL, and ModelSim is used for simulation verification. The filter design is based on parallel and pipelined architectures, using an 8-tap low-pass filter. Parallel processing allows multiple multiplication and accumulation operations to occur simultaneously, while the pipelined architecture divides the filter operation into several stages, enabling multiple sets of data to be processed during each clock cycle [5,6]. The simulation results indicate how the proposed parallel and pipelined FIR filter processes the input signals in stages. These results align with previous studies demonstrating the efficiency of pipelined architectures in reducing computational delays and improving throughput in FPGA-based systems [4,6].

2.2. Experimental Parameters

1. Filter order: 8-tap, balancing the filtering effect and hardware resource requirements.

2.Filter coefficients: $h[k] = [1, 2, 3, 4, 3, 2, 1, 0]$, designed with symmetric coefficients to ensure linear phase, suitable for smoothing signals. The filter output y[n] is calculated by convolving the input signal with the filter coefficients, as per the following mathematical expression:

$$\boldsymbol{y[n] = \sum_{k=0}^{N-1} h[x] \cdot x[n-k]} \quad (1)$$

3. Input signal: Incremental values (50, 100, ... 400) are used to test the filter's response characteristics.

4. Fixed-point width: Input and coefficients are 16-bit, the output is 32-bit, ensuring precision and preventing overflow.

5, Sampling rate: Indirectly set to 100 MHz, ensuring high-frequency resolution and real-time performance.

6. Pipelined design: multi-stage pipelining enhances computational parallelism, reduces latency, and increases throughput.

2.3. Filter Structure Diagram: The combination of parallel and pipelined design

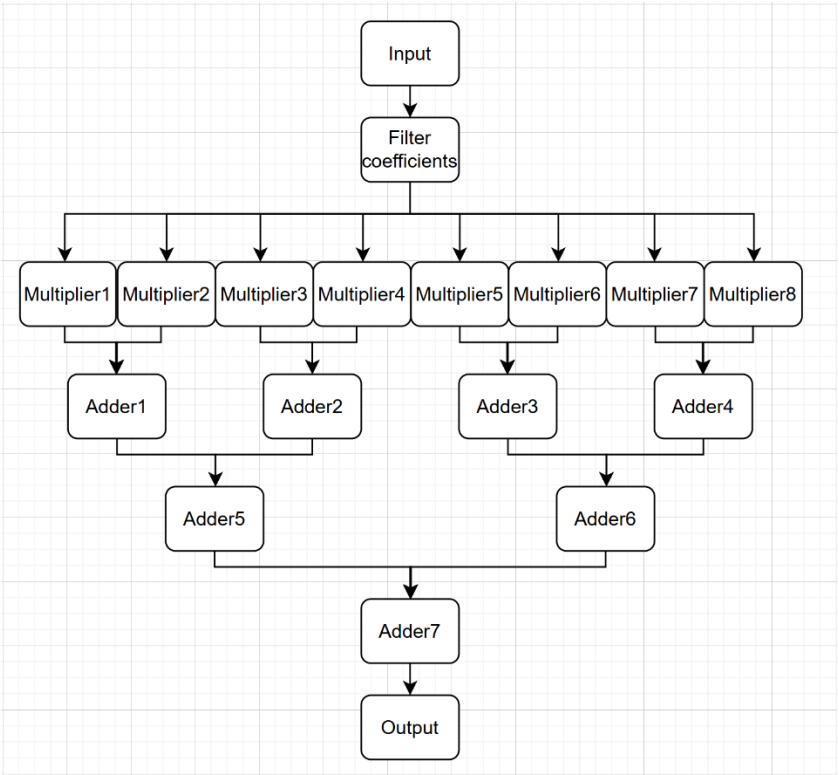

Fig. 1 Diagram of Parallel and Pipelined FIR Filter Architecture

As shown in Fig1, the input signal and filter coefficients are computed through 8 parallel multiplication modules, followed by a 3-stage pipelined accumulator to produce the filter output. This combined parallel and pipelined design significantly improves computation speed and processing efficiency, meeting the needs of high real-time applications.

2.4. Verilog Implementation of Parallel-Pipelined 8-Tap FIR Filter

The following is the relevant Verilog code (Fig2):

```
module fir_filter_detailed_pipeline_tb;
    reg clk;
    reg rst;
    reg signed [15:0] x_in;
    wire signed [31:0] y_out;
    wire signed [31:0] pipeline_out_1_0;
    wire signed [31:0] pipeline_out_1_1;
    wire signed [31:0] pipeline_out_1_2;
    wire signed [31:0] pipeline_out_1_3;
    wire signed [31:0] pipeline_out_2_0;
    wire signed [31:0] pipeline_out_2_1;
    wire signed [31:0] pipeline_out_3;

    // Instantiate the FIR filter
    fir_filter_detailed_pipeline uut (
        .clk(clk),
        .rst(rst),
        .x_in(x_in),
        .y_out(y_out),
        .pipeline_out_1_0(pipeline_out_1_0),
        .pipeline_out_1_1(pipeline_out_1_1),
        .pipeline_out_1_2(pipeline_out_1_2),
        .pipeline_out_1_3(pipeline_out_1_3),
        .pipeline_out_2_0(pipeline_out_2_0),
        .pipeline_out_2_1(pipeline_out_2_1),
        .pipeline_out_3(pipeline_out_3)
    );

    // Clock generation
    always #5 clk = ~clk;

    initial begin
        // Initialize signals
        clk = 0;
        rst = 1;
        x_in = 16'd0;

        // Reset the filter
        #10 rst = 0;

        // Apply input samples
        #10 x_in = 16'd50;
        #10 x_in = 16'd100;
        #10 x_in = 16'd150;
        #10 x_in = 16'd200;
        #10 x_in = 16'd250;
        #10 x_in = 16'd300;
        #10 x_in = 16'd350;
        #10 x_in = 16'd400;

        // Wait and finish
        #100;
        $stop;
    end
endmodule
```

```
module fir_filter_detailed_pipeline (
    input wire clk,
    input wire rst,
    input wire signed [15:0] x_in,
    output reg signed [31:0] y_out,
    output reg signed [31:0] pipeline_out_1_0,
    output reg signed [31:0] pipeline_out_1_1,
    output reg signed [31:0] pipeline_out_1_2,
    output reg signed [31:0] pipeline_out_1_3,
    output reg signed [31:0] pipeline_out_2_0,
    output reg signed [31:0] pipeline_out_2_1,
    output reg signed [31:0] pipeline_out_3
);

    // Coefficients for 8-tap FIR filter
    reg signed [15:0] h [0:7];

    // Input sample register array
    reg signed [15:0] x [0:7];
    reg signed [31:0] partial_sum [0:7];
    reg signed [31:0] stage_sum [0:7];

    // Pipeline registers for detailed output tracking
    reg signed [31:0] pipeline_reg_1 [0:3];
    reg signed [31:0] pipeline_reg_2 [0:1];
    reg signed [31:0] pipeline_reg_3;

    integer i;

    initial begin
        // Initialize filter coefficients (example values)
        h[0]  = 16'd1;  h[1]  = 16'd2;  h[2]  = 16'd3;  h[3]  = 16'd4;
        h[4]  = 16'd3;  h[5]  = 16'd2;  h[6]  = 16'd1;  h[7]  = 16'd0;
    end

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            // Reset all states
            for (i = 0; i < 8; i = i + 1) begin
                x[i] <= 16'd0;
                partial_sum[i] <= 32'd0;
                stage_sum[i] <= 32'd0;
            end
            for (i = 0; i < 4; i = i + 1) begin
                pipeline_reg_1[i] <= 32'd0;
            end
            for (i = 0; i < 2; i = i + 1) begin
                pipeline_reg_2[i] <= 32'd0;
            end
            pipeline_reg_3 <= 32'd0;
            pipeline_out_1_0 <= 32'd0;
            pipeline_out_1_1 <= 32'd0;
            pipeline_out_1_2 <= 32'd0;
            pipeline_out_1_3 <= 32'd0;
            pipeline_out_2_0 <= 32'd0;
            pipeline_out_2_1 <= 32'd0;
            pipeline_out_3 <= 32'd0;
            y_out <= 32'd0;
```

```
        end else begin
            // Shift the input samples
            for (i = 7; i > 0; i = i - 1) begin
                x[i] <= x[i-1];
            end
            x[0] <= x_in;

            // Compute partial products
            for (i = 0; i < 8; i = i + 1) begin
                partial_sum[i] <= x[i] * h[i];
            end

            // Pipeline stage for summation
            pipeline_reg_1[0] <= partial_sum[0] + partial_sum[1];
            pipeline_reg_1[1] <= partial_sum[2] + partial_sum[3];
            pipeline_reg_1[2] <= partial_sum[4] + partial_sum[5];
            pipeline_reg_1[3] <= partial_sum[6] + partial_sum[7];

            pipeline_out_1_0 <= pipeline_reg_1[0];
            pipeline_out_1_1 <= pipeline_reg_1[1];
            pipeline_out_1_2 <= pipeline_reg_1[2];
            pipeline_out_1_3 <= pipeline_reg_1[3];

            pipeline_reg_2[0] <= pipeline_reg_1[0] + pipeline_reg_1[1];
            pipeline_reg_2[1] <= pipeline_reg_1[2] + pipeline_reg_1[3];

            pipeline_out_2_0 <= pipeline_reg_2[0];
            pipeline_out_2_1 <= pipeline_reg_2[1];

            pipeline_reg_3 <= pipeline_reg_2[0] + pipeline_reg_2[1];
            pipeline_out_3 <= pipeline_reg_3;

            y_out <= pipeline_reg_3;
        end
    end

endmodule
```

Fig. 2 Verilog code for the parallel and pipelined FIR filter design

3. Simulation Verification and Analysis

3.1. Waveform and Interpretation of FIR Filter Module Output Signals

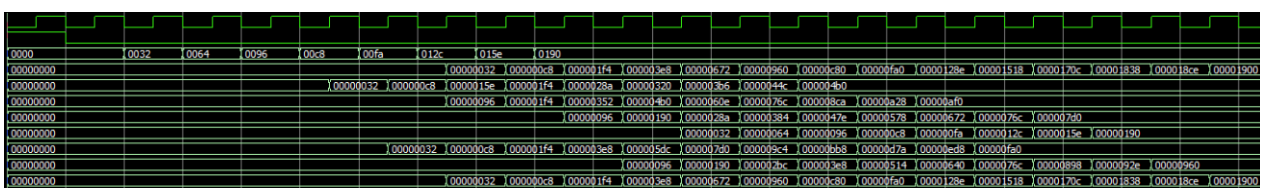

Fig. 3 Waveform

Waveform and Interpretation of FIR Filter Module Output Signals as shown in figure3.

3.1.1. y_out (final filter output):

y_out is the final output signal of the FIR filter, obtained as the accumulated result of the input samples filtered by the filter coefficients. In the waveform, you will see the value of y_out change over time, indicating that the filter is processing the input samples and outputting the corresponding filtered results.

3.1.2. pipeline_out_1_0-pipeline_out_1_3(first stage pipeline outputs):

These signals represent the output of the first stage pipeline of the filter, i.e., partial multiplication and accumulation results. pipeline_out_1_0 is the accumulation result of the 1st and 2nd products. pipeline_out_1_1 is the accumulation result of the 3rd and 4th products. pipeline_out_1_2 is the accumulation result of the 5th and 6th products. pipeline_out_1_3 is the accumulation result of the 7th and 8th products. In the waveform, you will see these signals change gradually with each clock cycle, indicating intermediate results of multiplication and accumulation operations.

3.1.3. pipeline_out_2_0 and pipeline_out_2_1 (second stage pipeline outputs):

These signals are further accumulation results from the partial sums of the first stage. pipeline_out_2_0 is the accumulation of pipeline_out_1_0 and pipeline_out_1_1. pipeline_out_2_1 is the accumulation of pipeline_out_1_2 and pipeline_out_1_3. In the waveform, you will see these signals accumulate step by step within each clock cycle, representing further computation steps.

3.1.4. pipeline_out_3 (third stage pipeline output):

pipeline_out_3 is the accumulation result of the two partial sums from the second stage (pipeline_out_2_0 and pipeline_out_2_1). In the waveform, its value represents the final result of all multiplications and accumulations, but without the delayed processing of the filter.

3.2. Testbench Module fir_filter_detailed_pipeline_tb Simulation Output Meaning

clk (clock signal): The clock signal is clk, which appears as a square wave in the waveform, indicating rising and falling edges. The clock signal controls the computation pace of the filter, with all operations triggered on the rising edge.

rst (reset signal): The reset signal rst is used to initialize the internal state of the filter. In the simulation waveform, the reset signal starts high and then drops low. When rst is high, all registers of the filter are cleared; when rst goes low, the filter starts operating normally.

x_in (input signal): x_in is the signal input to the filter. In the waveform, you can see the changes in the input signal x_in, such as gradual changes from 50 to 400. Each time the signal changes, the filter processes the input.

y_out and pipeline_out_* signals: All these output signals can be seen changing with the input signal in the waveform. Each pipeline_out_* signal shows intermediate computation results at different stages of the filter, reflecting the detailed steps of parallel and pipelined operations. y_out is the overall output of the filter, accumulating and outputting filtered values as the input signal is processed.

3.3. Key Features in the Waveform

Interaction between Clock and Reset: At the beginning of the simulation, the first signal is high, initializing all filter registers to 0. After the rest goes low, the filter starts responding to the input signal x_in.

Changes in Input and Intermediate Results: As the input signal x_in gradually increases, all pipeline_out_* signals and the final output y_out also change accordingly. These gradual changes indicate how the filter processes the signal in stages.

Pipelining Characteristics: It can be observed that, as the clock cycles progress, there are clock cycle differences between the various pipeline_out_* signals, demonstrating the stepwise processing characteristic of pipelining. For example, pipeline_out_1_* represents the earliest multiplication and accumulation parts, while pipeline_out_2_* and pipeline_out_3 indicate the progressive advancement of the accumulation process.

Filter Delay: There is a delay between the input x_in and the output y_out. This is because the filter requires several clock cycles to complete all stages of accumulation before producing an output. Therefore, there is a time lag between input and output, which is typical of FIR filter delay characteristics.

4. Experimental Results and Discussion

The ModelSim simulation demonstrates the real-time response of the parallel and pipelined FIR filter to different input signals. Compared with traditional serial designs, the parallel design allows each filter tap to be computed simultaneously, significantly improving processing speed. In the implementation of a 16-tap filter, the serial design requires 16 clock cycles to complete all operations, whereas the parallel design requires only 1 clock cycle to complete the same computation, resulting in a 16-fold increase in processing speed and smoother output signal changes. The pipelined structure further improves system throughput, reducing overall processing delay and enabling the filter to efficiently handle high-frequency signal inputs in a shorter time. The simulation results indicate that the combination of parallel and pipelined structures performs excellently in terms of efficiency and real-time performance, making it suitable for real-time signal processing applications.

5. Conclusion

This paper presents a novel approach to the design of Finite Impulse Response (FIR) filters, leveraging the combined use of parallel and pipelined architectures to optimize performance for real-time signal processing applications. The FIR filter, a key component in digital signal processing (DSP), is widely used in fields such as communications, audio processing, and image processing, due to its inherent stability and linear phase characteristics. However, real-time processing of FIR filters, especially in resource-constrained embedded systems, often faces challenges in maintaining high processing speed and efficient resource utilization. To address these issues, the study proposes an optimized FIR filter design that integrates both parallel and pipelined structures to significantly improve computational efficiency and throughput.

Parallel architecture enables multiple filters taps to be processed simultaneously, allowing for the concurrent execution of various multiplication and accumulation operations. This parallel processing reduces the overall computation time, thereby enhancing the speed of the FIR filter. On the other hand, the pipelined structure divides the filter operation into multiple stages, with each stage handling part of the computation in a synchronized manner across clock cycles. This pipelining approach further improves the system's throughput and reduces latency, making the filter well-suited for applications where real-time processing is critical.

The proposed filter design was implemented using Verilog Hardware Description Language (HDL), and its functionality and performance were validated through simulation using the ModelSim tool. The simulation results demonstrate that the FIR filter, when optimized with parallel and pipelined structures, delivers excellent performance in terms of both speed and resource efficiency. The filter exhibits stable behaviour under high-frequency signals, making it ideal for real-time signal processing applications that require rapid and accurate data processing. Compared to traditional serial FIR filter designs, the parallel-pipelined filter design significantly reduces processing time and enhances throughput while maintaining a moderate level of resource consumption.

This study provides valuable insights into optimizing FIR filters for embedded systems, where constraints on processing speed and hardware resources are common. The combination of parallel and pipelined architecture offers a robust solution to improve the performance of digital filters in such environments. Moreover, the results highlight the potential for further optimization in future research, particularly in exploring adaptive filtering techniques and resource-efficient FPGA implementations. The proposed design sets a new direction for FIR filter applications in embedded systems, contributing to more efficient and high-performance real-time signal processing solutions.

Acknowledgements

All the authors contributed equally, and their names were listed in alphabetical order.

References

[1] Liu, P., & Xu, J. Design and Implementation of FIR Digital Filter Based on FPGA (Master's thesis, Northwestern Polytechnical University), 2006.

[2] Rabaey, J. M., & Pedram, M. A survey of FPGA-based digital filter implementations. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 1995, 3(3), 263-272.

[3] Nelson, T. L., & Duda, P. M. Efficient FPGA design of digital filters using pipelining and parallelism. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 1998, 45(6), 747-754.

[4] Lee, L. P. L. L., Chang, C. H., & Lee, K. J. Design of a high-speed FIR filter using parallel processing and pipelining on FPGA. IEEE Transactions on Circuits and Systems I: Regular Papers, 2003, 50(4), 520-528.

[5] Rajan, M. P. K., & Vijayakumar, S. R. Design and implementation of parallel FIR filter using FPGA. International Journal of Engineering & Technology, 2014, 5(3), 315-320.

[6] Srinivasan, M. R. N. S., & Kumar, B. P. S. R. An efficient implementation of FIR filters on FPGA with pipelined architecture. Journal of Electrical Engineering & Technology, 2011, 6(4), 507-513.