

Design of Reconfigurable Linear-Phase FIR Filter Using Region-Aware Hierarchical Arithmetic

Chetti. Venkateswarlu¹, Dr. Tipparti. Anil Kumar²

¹Assistant Professor, Department of Electronics and Communication Engineering, Malla Reddy Engineering College for Women (Autonomous), Hyderabad, Telangana.

²Professor & Principal, Department of Electronics and Communication Engineering, Jyothishmathi Institute of Technology and Science, (Autonomous), Karimnagar, Telangana

¹venkatchetti0@gmail.com, ²tvakumar2000@yahoo.co.in

Abstract

Finite Impulse Response (FIR) filters play a significant role in Digital Signal Processing (DSP) applications, including wireless communication systems, biomedical signal processing, audio and speech processing, image processing, radar systems, and real-time embedded applications. So, the work is to design the Reconfigurable Multi-Window Linear-Phase FIR (RMWLP-FIR) architecture, which incorporates reconfigurable multi-window processing, symmetry pairing, a Region-Aware Hierarchical Multiplier (RAHM), an error-balanced arithmetic structure, and pipelined processing to improve resource utilization, control arithmetic complexity, and reduce power consumption and propagation delay. The RMWLP-FIR achieves the lowest reported resource utilization, requiring 296 LUTs and 153 flip-flops, while recording a setup path delay of 25.450 ns, a hold path delay of 0.358 ns, a calculated combined setup-plus-hold delay of 25.808 ns, and total power consumption of 42.076 mW. Finally, the proposed architectural progression demonstrates the potential of symmetry-based computation, coefficient optimization, configurable window processing, and arithmetic resource sharing to improve the implementation efficiency of linear-phase FIR filters.

Keywords: Digital Signal Processing, FIR filter, Linear-phase filter, Low-power design, Multi-window processing, Pipelined architecture, Reconfigurable architecture, Resource optimization, Symmetry pairing.

1. Introduction

Digital filter-based systems play an essential role in modern electronic and communication technologies by improving signal quality, removing noise, reducing interference, and extracting useful information from input signals. In telecommunications, digital filters are used for channel selection, noise suppression, modulation and demodulation, equalization, and interference reduction. The growing demand for high-performance DSP across communication networks, healthcare equipment, automotive electronics, consumer devices, industrial automation, and aerospace and defense is increasing the importance of digital-filter-based hardware. Figure 1 presents the global integrated circuit (IC) market, rather than the standalone digital filter market. According to the Roots Analysis report, the global IC market was valued at USD 618.6 billion in 2024 and is projected to reach USD 2,169.6 billion by 2035, with a compound annual growth rate (CAGR) of 12.08% from 2024 to 2035. The market covers six major verticals: aerospace and defense, automotive, consumer electronics, healthcare, industrial automation, and telecommunications. This expansion provides a broader market context for digital filter implementations in DSPs, ASICs, and other integrated electronic systems.

The industrial relevance of linear-phase FIR filtering is reflected in the products and DSP platforms developed by semiconductor and communication technology companies. For example, Texas Instruments, Analog Devices, Qualcomm, NXP Semiconductors, Intel, AMD (Xilinx), STMicroelectronics, Renesas Electronics, Broadcom, and MediaTek operate in markets involving digital signal processors, wireless communication, embedded computing, and real-time signal processing. The research can therefore focus on designing a linear-phase FIR architecture in RTL, exploiting coefficient symmetry, and evaluating its implementation on FPGA platforms for communication, audio, biomedical, radar, and embedded DSP applications. The research direction is to develop a hardware architecture that maintains linear-phase characteristics while achieving efficient silicon-area utilization, low power consumption, high operating frequency, and reduced power-delay product.

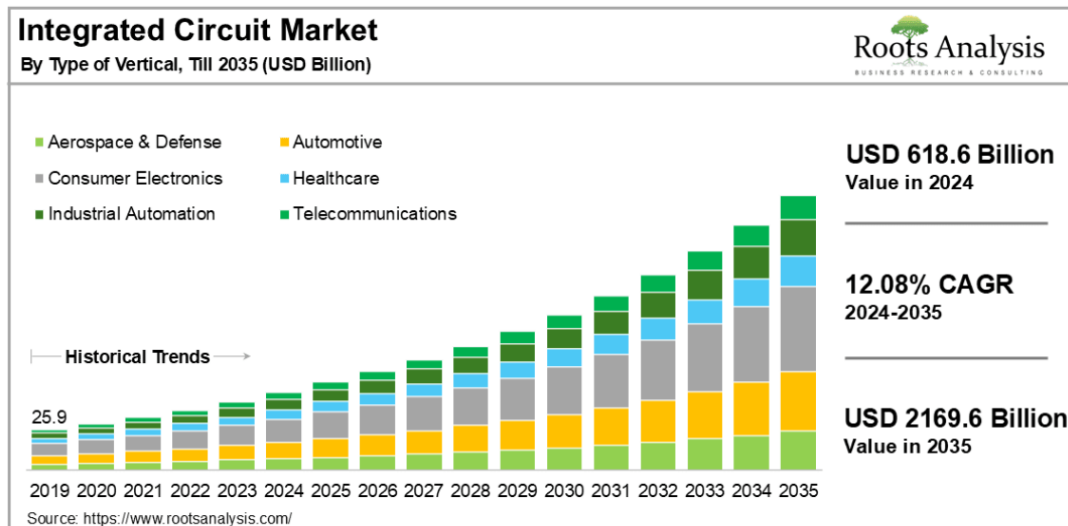


Figure 1. Global Integrated Circuit Market by Industry.

(Source: <https://www.rootsanalysis.com/integrated-circuit-market>)

2. Literature Survey

G. Reddy Hemantha et al. [1] proposed activity-driven FIR filter floor-planning using hybrid PSO-GA, while S. Karunakaran et al. [2] developed COBAD-FIR for power-efficient coefficient optimization. D. Raghunatha Rao et al. [3] investigated distributed-arithmetic-based adaptive FIR filtering, and Venkata Krishna Odugu et al. [4] combined coefficient symmetry, parallel processing, and distributed arithmetic for two-dimensional FIR filter banks. T. Kalavathi Devi et al. [5] explored a FinFET-GDI-based carry-select adder, while Kiran Kumar Bhadavath and Z. Mary Livinsa [6] employed retiming with optimized multipliers and adders. A. V. B. Rao and S. Suman [7] investigated hybrid fault-tolerant FIR architectures, and V. Magesh et al. [8] combined advanced adders with hybrid multipliers. S. Sasikala et al. [9] focused on low-power FIR filtering through hybrid multiplication and operation scheduling. However, these approaches face challenges involving optimization complexity, hardware overhead, timing constraints, and verification.

Bukkapatnam Meghana and B. S. Supreetha [10] investigated modular, reconfigurable higher-order FIR filters, while A. Sriram et al. [11] proposed a self-verifying FIR ASIC using Vedic arithmetic and Han–Carlson accumulation. Keshab K. Parhi [12] presented an iterated hybrid fast parallel FIR architecture, and Pavankumar Ganjimala et al. [13] studied high-throughput, area-efficient functional-link adaptive filters. Rajat Kumar and Aman Kumar [14] combined distributed arithmetic, approximate Karatsuba multiplication, and variable-length carry-select addition. C. Saranya et al. [15] evaluated FPGA-based digital filter architectures for image processing, while V. Anuraj and Dhandapani Vaithiyanathan [16] examined FIR implementations for real-time multimedia applications. A. Abinaya et al. [17] investigated polyphase FIR filtering for next-generation wireless systems. Loganathan Mohana Kannan et al. [18] integrated retimed adaptive LMS filtering with contextual convolutional attention for ECG detection. Despite these developments, challenges remain in balancing reconfigurability, numerical accuracy, throughput, power consumption, and resource utilization.

Pedro T. L. Pereira et al. [19] proposed HDL-DFGen to automate digital filter hardware generation, while Debarshi Datta and Mrinal Kanti Naskar [20] investigated reversible adders and multipliers for FIR implementation. J. Banumathi and G. Karthy [21] combined a Brent–Kung adder with a pipelined Vedic multiplier to improve arithmetic performance. Ziyang Huang et al. [22] incorporated hybrid FIR filtering into an IEEE 802.15.4/4z-compliant ultra-wideband transmitter. Murali Anumothu et al. [23] developed an FSM-enabled reconfigurable debugging approach for area-optimized FPGA-based FIR filters. Thiruvankadam Krishnan [24] investigated adaptive FIR filtering using a 5:2 compressor, while Saksham Goswami et al. [25] explored fast binary counters and sorting techniques for high-performance filtering-related computations. Oksana Moryakova and Håkan Johansson [26] studied closed-form least-squares design for fast-convolution-based variable-bandwidth FIR filters. These approaches demonstrate improvements in automation, arithmetic efficiency, flexibility, and

performance, although additional control logic, routing complexity, and processing latency remain important concerns.

Vinod Kumar and M. A. Majid [27] proposed an energy-efficient FIR architecture using a modified Booth–Wallace multiplier and Kogge–Stone adder for IoT-based wireless sensor networks. K. R. Shobha [28] investigated FPGA-based real-time ECG acquisition and FIR filtering, while Sami Khorbotly and Fayol Ateufack Zeudom [29] examined optimized shift-and-add multipliers for filtering and neural-network applications. S. T. Aarthy and J. L. Mazher Iqbal [30] developed a low-area ECG noise-reduction FIR filter using a Dadda multiplier and parallel-prefix adder.

3. Research Methodology

The RMWLP-FIR architecture as shown in Figure 2 is designed to support configurable digital filtering operations through a unified hardware structure that combines window-based data processing, linear-phase symmetry exploitation, and resource-efficient arithmetic computation. The architecture consists of an Input Interface, Window Buffer Unit, Preprocessing and Data Arrangement Unit, Configurable Arithmetic Processing Unit, Pipeline Registers and Data Alignment Unit, Output Interface, Control Unit, Coefficient/Kernel Memory, and Configuration Registers. The Window Buffer Unit stores and organizes incoming samples, while the preprocessing stage generates the required windows and pairs symmetric samples to reduce redundant computations. The Configurable Arithmetic Processing Unit integrates the RAHM, EBA-SAS, and LCS units to perform configurable arithmetic operations. The Control Unit coordinates filter selection, configuration, address generation, and control signal distribution based on the stored coefficients and operating parameters. Through its reconfigurable multi-window processing capability, the RMWLP-FIR architecture supports different window sizes, filter configurations, and coefficient sets while maintaining the linear-phase characteristics of FIR filtering.

Step 1: Input Interface: The Input Interface receives the discrete-time input signal from an external source. It ensures that incoming samples are transferred to the Window Buffer Unit in the correct sequence. The interface manages the input data format and coordinates sample transfer with the system timing. It establishes the connection between the external signal source and the internal filtering hardware. The received samples are forwarded for buffering and subsequent multi-window processing.

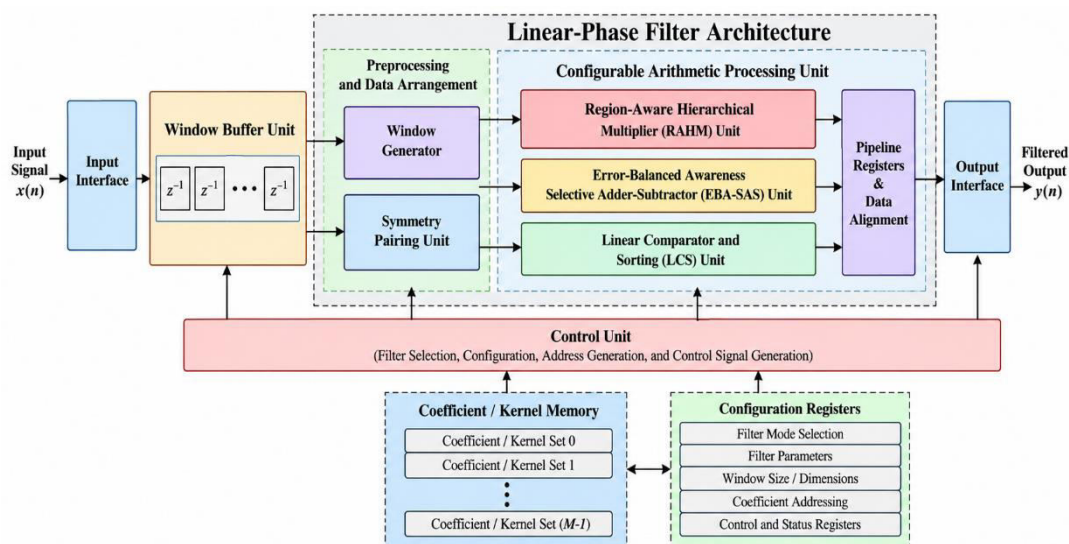


Figure 2. RMWLP-FIR Architecture.

Step 2: Window Buffer Unit: The Window Buffer Unit stores incoming samples and maintains the required history of previously received data. It uses delay elements and buffer registers to retain samples for successive filtering operations. The stored samples are made available to the Window Generator according to the selected window configuration. Reusing buffered samples reduces unnecessary data retrieval and supports continuous signal processing. The buffer operation is coordinated with the Control Unit to maintain correct sample ordering.

Step 3: Window Generator: The Window Generator organizes buffered samples into windows according to the selected window size and filtering configuration. It selects the appropriate samples required for each processing operation. Multiple window configurations can be supported through programmable dimensions and control settings. The generated windows are synchronized with the downstream Symmetry Pairing Unit and arithmetic datapath. This stage ensures that the correct sample groups are available for each filtering operation.

Step 4: Symmetry Pairing Unit: The Symmetry Pairing Unit identifies and pairs samples corresponding to symmetric positions in a linear-phase FIR filter. It exploits coefficient symmetry to reduce the number of independent multiplication operations. The paired samples are arranged according to the selected coefficient set and filtering mode. The unit maintains correct sample-to-coefficient alignment before transferring data to the arithmetic processing stage. For antisymmetric filters, the pairing operation is configured to support the required subtraction-based processing.

Step 5: RAHM Unit: The RAHM Unit performs multiplication between the input samples or symmetry-paired samples and the selected filter coefficients. Its hierarchical structure organizes multiplication into smaller arithmetic operations and combines the resulting partial products. The region-aware design supports the intended allocation of arithmetic resources according to the operand and configuration requirements. The generated multiplication results are transferred to the EBA-SAS Unit for subsequent arithmetic processing. The Control Unit coordinates coefficient selection, data availability, and operation timing.

Step 6: EBA-SAS Unit: The EBA-SAS Unit performs the required addition and subtraction operations on intermediate arithmetic results. It combines partial products and other operands according to the selected filter configuration. Its selective operation supports configurable arithmetic processing within the RMWLP-FIR datapath. The unit maintains the required numerical precision and synchronizes its results with the subsequent processing stage. The resulting data is forwarded to the LCS Unit when required or to the pipeline stage according to the configured datapath.

Step 7: LCS Unit: The LCS Unit performs comparison and sorting operations for processing modes that require ordered data or comparison-based decisions. It evaluates input values according to the selected comparison criteria. When sorting is enabled, it arranges the values in the required order before forwarding the results. Its operation is coordinated with the Control Unit to maintain the correct processing sequence and data alignment. For conventional FIR convolution, this unit should be bypassed unless the selected architecture explicitly requires comparison or sorting.

Step 8: Pipeline Registers and Data Alignment Unit: The Pipeline Registers and Data Alignment Unit temporarily stores intermediate results between successive processing stages. It synchronizes data produced by arithmetic units with different processing latencies. The registers divide the datapath into stages, supporting continuous operation and improved timing performance. Data alignment ensures that related arithmetic results and control signals remain correctly associated. The synchronized results are forwarded to the Output Interface for final signal delivery.

Step 9: Control Unit: The Control Unit manages the overall operation of the RMWLP-FIR architecture. It generates control signals for window generation, symmetry pairing, coefficient selection, arithmetic-unit activation, and output timing. It coordinates data movement between the buffer, preprocessing stage, arithmetic units, and pipeline registers. The unit also manages configuration changes and ensures that processing stages operate in the required sequence. Its centralized control mechanism maintains synchronization throughout the filtering process.

Step 10: Coefficient / Kernel Memory: The memory stores multiple sets of FIR coefficients for different filter configurations. Each set contains the coefficient values required for a particular filtering operation. The Control Unit generates memory addresses according to the selected filter mode and coefficient configuration. The required coefficients are supplied to the arithmetic processing unit in synchronization with the corresponding sample data. This memory organization supports filter reconfiguration without requiring separate coefficient storage for every supported filter mode.

Step 11: Configuration Registers: The Configuration Registers store the parameters required to configure and operate the RMWLP-FIR architecture. These parameters include filter mode selection, filter characteristics,

window dimensions, coefficient addressing, and control and status information. The Control Unit reads these settings to determine the active processing configuration. The stored parameters coordinate the operation of the Window Generator, arithmetic units, and other architectural blocks. This register-based configuration enables the hardware to support different filtering modes and window sizes.

Step 12: Output Interface: The Output Interface receives the processed and synchronized results from the Pipeline Registers and Data Alignment Unit. It transfers the final filtered samples to the external destination in the required data format. The interface coordinates output delivery with the system timing and relevant control signals. It maintains the correct output sequence for continuous signal processing. Finally, it delivers the filtered output signal generated by the configured RMWLP-FIR architecture.

3.1 Region-Aware Hierarchical Multiplier

The RAHM as shown in Figure 3 is designed to reduce unnecessary multiplication activity by dividing the input operands into Low, Medium, and High Regions and checking each region for zero-valued bit patterns. The multiplier activates the required region-level multiplication operations only when their corresponding operand combinations are nonzero. The generated partial products are shifted according to their bit positions and combined hierarchically to obtain the final multiplication result. This region-based structure supports selective partial-product generation and reduces unnecessary switching activity when zero-valued regions occur. The overall hardware benefits depend on the region partitioning, zero-detection logic, operand statistics, and synthesis implementation.

Step 1: Input Operand Reception: The RAHM Unit receives two binary operands: the input sample A from the Symmetry Pairing Unit and the coefficient B from Coefficient / Kernel Memory. Both operands are represented using W -bit binary words. The operands are synchronized before being transferred to the region decomposition stage. Their bit widths determine the size of the Low, Medium, and High Regions.

Step 2: Low, Medium, and High Region Division: Each operand is divided into three regions according to its bit positions. Let the Low Region contain L bits, the Medium Region contain M bits, and the High Region contain H bits, where $W = L + M + H$. The operand A can be expressed as

$$A = A_L + 2^L A_M + 2^{L+M} A_H \quad (1)$$

Similarly, operand B is represented as

$$B = B_L + 2^L B_M + 2^{L+M} B_H \quad (2)$$

Here, A_L, B_L denote the Low Regions, A_M, B_M denote the Medium Regions, and A_H, B_H denote the High Regions.

Step 3: Region-Wise Zero Detection: The zero-detection circuit checks whether each region contains only zeros. A region containing zero contributes no value when multiplied by any other region. The detection outputs are used to control the generation of the corresponding partial products. For region A_r , the zero flag is

$$Z_{A_r} = \begin{cases} 1, & A_r = 0 \\ 0, & A_r \neq 0 \end{cases} \quad (3)$$

Similarly, for region B_s ,

$$Z_{B_s} = \begin{cases} 1, & B_s = 0 \\ 0, & B_s \neq 0 \end{cases} \quad (4)$$

where $r, s \in \{L, M, H\}$. The partial product for a region pair can be enabled using

$$E_{rs} = \neg(Z_{A_r} \vee Z_{B_s}) \quad (5)$$

Therefore, the region-pair multiplication is performed only when both participating regions are nonzero.

Step 4: Low-Region Multiplication: The Low Region multiplication unit processes the least significant portions of the operands. Its output contributes to the lower-order portion of the final product. If either Low Region is zero, the corresponding multiplication contribution is suppressed. Otherwise, the unit generates the Low Region product.

$$P_{LL} = A_L B_L \quad (6)$$

The zero-aware form is

$$P_{LL} = \begin{cases} A_L B_L, & E_{LL} = 1 \\ 0, & E_{LL} = 0 \end{cases} \quad (7)$$

This operation avoids generating unnecessary multiplication activity for a zero-valued Low Region pair.

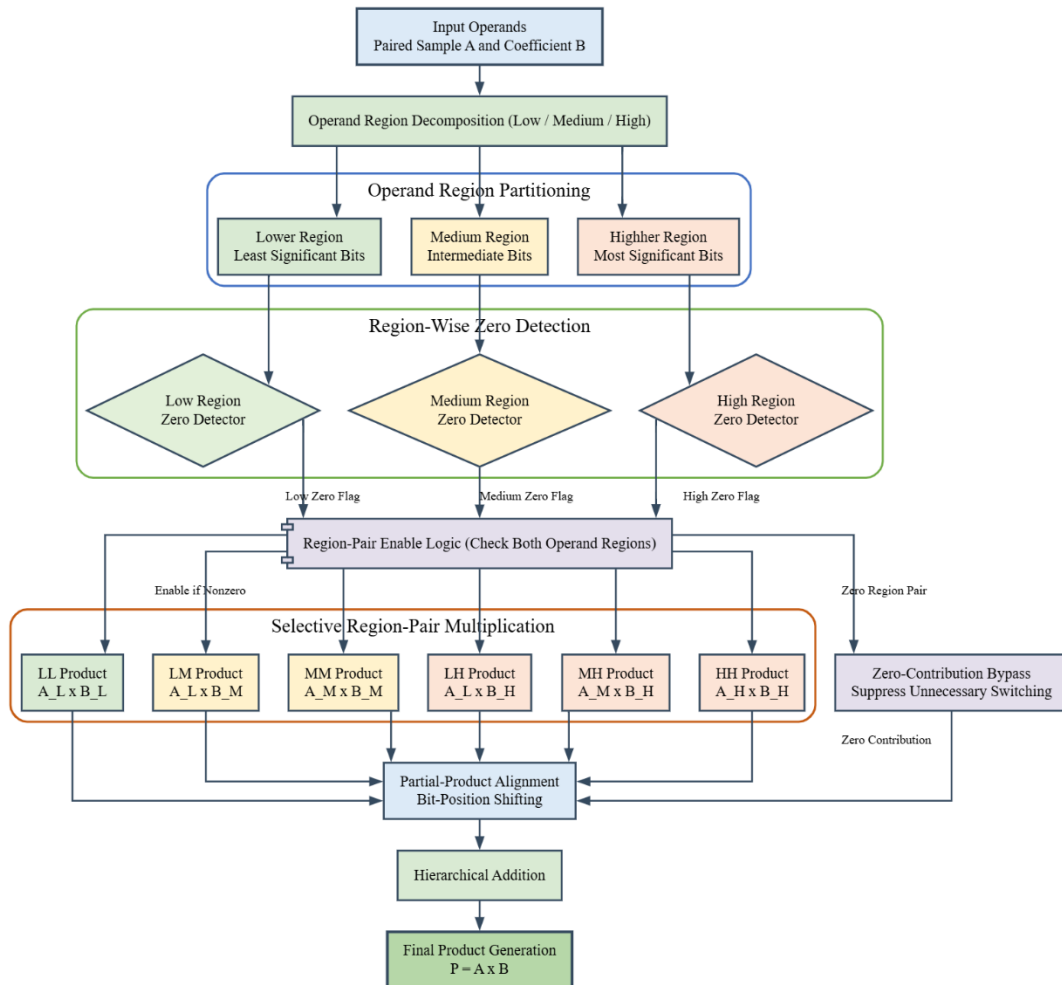


Figure 3. RAHM Flowchart.

Step 5: Medium-Region Multiplication: The Medium Region multiplication unit processes the intermediate operand portions. It generates the partial products involving the Medium Regions of the two operands. Each multiplication is controlled by the zero-detection results for the corresponding region pair.

$$P_{MM} = A_M B_M \quad (8)$$

For cross-region multiplication, the unit also generates terms such as

$$P_{LM} = A_L B_M, \quad P_{ML} = A_M B_L \quad (9)$$

These terms are required to reconstruct the complete product and must be retained whenever both participating regions are nonzero.

Step 6: High-Region Multiplication: The High Region multiplication unit processes the most significant portions of the operands. It generates high-order and cross-region partial products according to the selected region combinations. When a participating region is zero, its corresponding multiplication term is bypassed. Nonzero terms are retained and shifted according to their original bit positions.

$$P_{HH} = A_H B_H \quad (10)$$

Other necessary cross-region terms include

$$P_{LH} = A_L B_H, P_{HL} = A_H B_L, P_{MH} = A_M B_H, P_{HM} = A_H B_M \quad (11)$$

All nine region-pair products must be considered for a general three-region unsigned multiplication, although zero detection can eliminate terms whose operand regions are zero.

Step 7: Region-Based Partial-Product Generation: After the region-level multiplications, the generated partial products are organized according to their bit significance. Each term is associated with the appropriate shift amount, determined by the positions of the corresponding operand regions. The complete multiplication can be expanded as

$$P = A_L B_L + 2^L (A_L B_M + A_M B_L) + 2^{2L} A_M B_M + 2^{L+M} (A_L B_H + A_H B_L) + 2^{2L+M} (A_M B_H + A_H B_M) + 2^{2(L+M)} A_H B_H \quad (12)$$

This expression considers that the Low and Medium Regions each contain L bits, and the High Region contains H bits. Each region-level contribution is positioned correctly before combination.

Step 8: Hierarchical Partial-Product Combination: The aligned partial products are combined using hierarchical addition stages. The first stages combine terms with similar bit positions, while subsequent stages combine the intermediate sums to generate the complete product. Zero-valued terms can be omitted from the active arithmetic path, provided that the remaining terms retain their correct shift positions. Carry propagation must be handled across the addition stages to preserve arithmetic accuracy. The result is

$$P = \sum_{r \in \{L, M, H\}} \sum_{s \in \{L, M, H\}} 2^{q_r + q_s} A_r B_s \quad (13)$$

$$q_L = 0, \quad q_M = L, \quad q_H = L + M \quad (14)$$

This formulation includes all nine region-pair contributions.

Step 9: Final Product Generation and Synchronization: The final product is produced after the required partial products have been aligned and combined. The output logic preserves the product's required bit width and ensures that the result corresponds to the correct input sample and coefficient. A valid signal indicates when the multiplication result is ready for the next processing stage. The product is then transferred to the EBA-SAS Unit.

$$P(n) = A(n) \times B(n) \quad (15)$$

where $P(n)$ represents the multiplication result associated with the current processing operation.

3.2 EBA-SAS Unit

The EBA-SAS Unit as shown in Figure 4 performs configurable addition and subtraction operations on the partial products generated by the RAHM. It selects the appropriate arithmetic operation according to the control signals and the active filtering configuration. The unit combines multiplication results while maintaining the required numerical precision, carry propagation, and data alignment. Its error-aware mechanism can be designed to balance arithmetic accuracy and hardware complexity through selective exact or approximate arithmetic, provided that the corresponding error-control logic is implemented. The resulting arithmetic values are synchronized and forwarded to the subsequent processing stage of the RMWLP-FIR architecture.

Step 1: Input Operand Reception: The EBA-SAS Unit receives the multiplication results generated by the RAHM Unit. These operands represent partial products associated with the input samples and selected filter coefficients. The unit validates the availability and alignment of the operands before performing arithmetic operations. The operands are represented as binary values of the required bit width. Let the two input operands be A and B . The arithmetic output is represented as

$$Y = f(A, B, S) \quad (16)$$

where S is the operation-selection signal.

Step 2: Arithmetic Operation Selection: The operation-selection logic determines whether the operands require addition or subtraction. The Control Unit generates the selection signal according to the active arithmetic mode. For addition, the operands are combined directly; for subtraction, the second operand is subtracted from the first.

The selected operation is implemented using the corresponding arithmetic path or a shared configurable datapath. The operation is expressed as

$$Y = \begin{cases} A + B, & S = 0 \\ A - B, & S = 1 \end{cases} \quad (17)$$

where $S = 0$ selects addition and $S = 1$ selects subtraction.

Step 3: Selective Addition Operation: When addition is selected, the EBA-SAS Unit combines the input operands using its adder circuitry. The addition logic processes the corresponding bits and propagates carry information between bit positions. For multioperand arithmetic, the partial products are combined through successive addition stages or an appropriate adder structure. The unit preserves the required numerical precision and generates the sum for the subsequent processing stage. The addition operation is

$$Y_{\text{add}} = A + B \quad (18)$$

For individual binary bit positions, the sum and carry are

$$s_i = a_i \oplus b_i \oplus c_i \quad (19)$$

$$c_{i+1} = a_i b_i + c_i(a_i \oplus b_i) \quad (20)$$

where a_i and b_i are operand bits, c_i is the incoming carry, s_i is the sum bit, and c_{i+1} is the outgoing carry.

Step 4: Selective Subtraction Operation: When subtraction is selected, the unit calculates the difference between the two input operands. Subtraction can be implemented using a dedicated subtractor or an adder configured with two's-complement arithmetic. In the two's-complement approach, the second operand is inverted and a carry-in of one is applied. The circuit handles borrow or carry propagation according to the selected implementation. The resulting difference is forwarded for subsequent processing. The subtraction operation is

$$Y_{\text{sub}} = A - B \quad (21)$$

Using two's-complement arithmetic,

$$A - B = A + \overline{B} + 1 \quad (22)$$

where $\overline{B}$ represents the bitwise complement of B .

Step 5: Error-Aware Arithmetic Processing: The error-aware mechanism manages the trade-off between numerical accuracy and hardware complexity. If the architecture includes exact and approximate arithmetic paths, the selection logic determines which path is activated according to the specified accuracy requirements. An exact mode preserves the required arithmetic result, while an approximate mode introduces a controlled deviation from the exact result. The approximation policy must define the acceptable error limits and the conditions under which each path is selected. For an exact result Y_{exact} and an approximate result Y_{approx} , the absolute error is

$$E_{\text{abs}} = |Y_{\text{exact}} - Y_{\text{approx}}| \quad (23)$$

The relative error, when $Y_{\text{exact}} \neq 0$, is

$$E_{\text{rel}} = \frac{|Y_{\text{exact}} - Y_{\text{approx}}|}{|Y_{\text{exact}}|} \quad (24)$$

These measures can be used to evaluate the arithmetic accuracy of the implemented EBA-SAS design.

Step 6: Result Alignment and Width Management: After arithmetic processing, the EBA-SAS Unit aligns the result with the corresponding sample and filtering operation. Sign extension is applied when required for signed operands, and the output width is selected to accommodate the expected numerical range. Carry information and overflow conditions are handled according to the arithmetic representation. The resulting value is synchronized with the pipeline control signals before being transferred to the next stage. For two unsigned W -bit operands, a full-precision addition result can require $W + 1$ bits:

$$Y_{\text{add}} \in [0, 2^{W+1} - 2] \quad (25)$$

For signed arithmetic, the required width depends on the operand ranges and the number of accumulated terms.

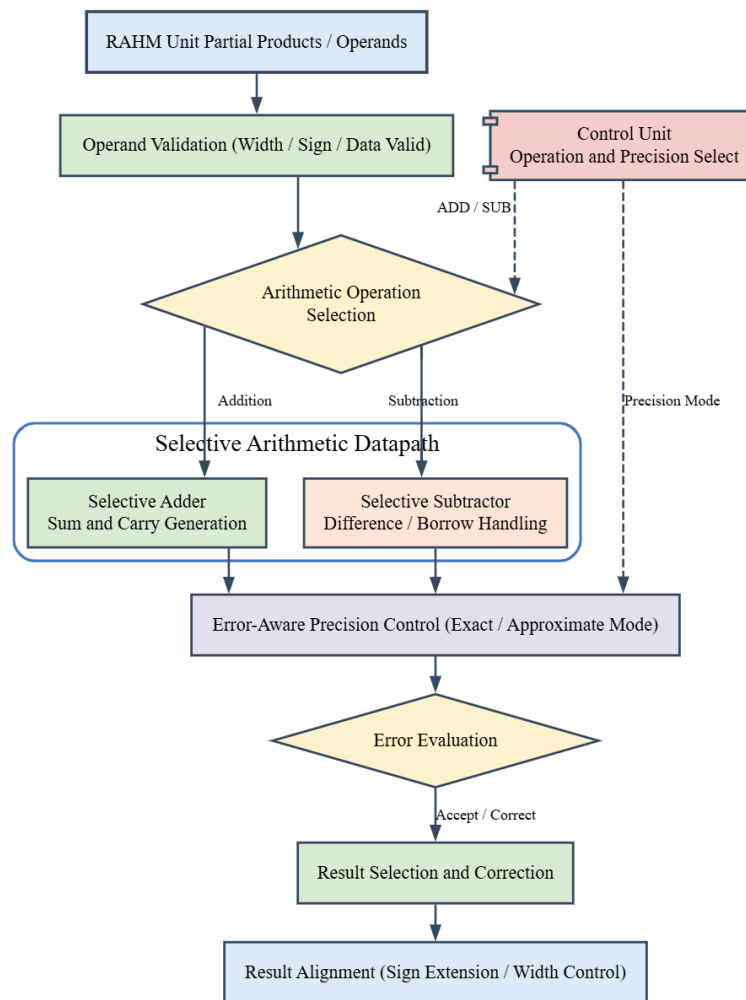


Figure 4. EBA-SAS Unit Flowchart.

Step 7: Pipeline Synchronization: The computed arithmetic result is registered and synchronized with the subsequent processing stages. Pipeline registers maintain the association between the output value, the corresponding sample window, and the active filter configuration. The Control Unit coordinates valid, enable, and timing signals to prevent incorrect data alignment. When a downstream stage is not ready, the implementation must manage the transfer according to its handshake or pipeline-control protocol. The synchronized result is then forwarded to the next configured processing stage. A registered arithmetic output can be represented as

$$Y_{\text{reg}}(n+1) = Y(n) \quad (26)$$

when the register is enabled and the result is valid.

Step 8: Output Transfer: The EBA-SAS Unit transfers the computed arithmetic result to the next stage of the RMWLP-FIR architecture. If comparison or sorting is required by the selected processing configuration, the result is forwarded to the Linear Comparator and Sorting (LCS) Unit. Otherwise, the LCS stage is bypassed, and the result proceeds directly to the Pipeline Registers and Data Alignment Unit or the next required accumulation stage. The output-valid signal identifies the availability of the processed result. This completes the selective addition or subtraction operation.

4. Results and Discussion

This section gives the detailed analysis of simulation and synthesis results, which are implemented using Xilinx-Vivado Software. Further, the results are compared with the state of art methodologies. In addition, MATLAB response analysis results also presented and compared with existing methods.

4.1 Synthesis Results

Figure 4 illustrates the complete RTL schematic of the proposed RMWLP-FIR architecture, integrating the window buffer, symmetry pairing unit, coefficient memory, RAHM, EBAS, LCS limiter, and pipeline output register. The coefficient memory provides programmable coefficients, while the window buffer stores input samples and the symmetry pairing unit generates intermediate sample combinations. The RAHM and EBAS modules perform arithmetic processing, and the LCS limiter constrains the output according to the specified minimum and maximum limits. The pipeline register synchronizes the output data and validity signal. This modular organization supports reconfigurable filtering while reducing redundant arithmetic operations through structured sample pairing and resource organization. Figure 5 presents the RTL schematic of the window buffer, which contains a series of asynchronous-reset registers for storing the input-sample history. The registers are interconnected to shift incoming samples through the buffer, producing the window data required by the symmetry pairing unit. The diagram shows multiple history registers, including history_reg[0] through history_reg[6], together with the corresponding data and control connections.

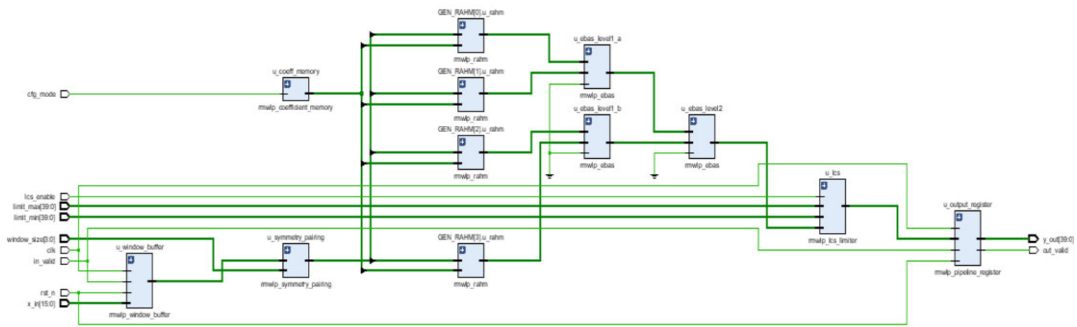


Figure 4. RTL Schematic of the RMWLP-FIR Architecture

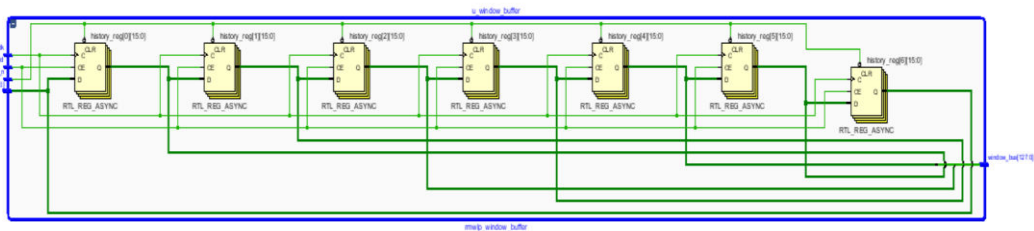


Figure 5. RTL Schematic of the Window Buffer

Figure 6 illustrates the RTL implementation of the symmetry pairing unit, which contains subtractors, multipliers, barrel-shift operations, adders, and multiplexers for generating configurable sample combinations. The module accepts the window-buffer data and uses the selected window size and configuration signals to control the pairing operations. Intermediate results are processed through the arithmetic stages and selected by multiplexers to produce the required pair outputs. Figure 7 presents the simulation waveform of the window-buffer module over approximately 0–110 ns. The displayed signals include clk, rst_n, in_valid, x_in[7:0], window_bus[39:0], and the internal index signal i[3:0]. At the selected cursor of 40 ns, the clock is 0, reset is deasserted (rst_n = 1), in_valid = 1, and the input sample is 6. The displayed window bus value is 774, while the index value is 2. The input sequence includes samples 3, 6, 9, 12, 15, and 18, showing how incoming data progresses through the window buffer. Figure 8 shows the simulation waveform of the symmetry pairing unit, displaying window_bus[63:0], window_size[3:0], pair_bus[35:0], and i[31:0] over approximately 0–4 ns. The displayed window bus is 0807060504030201, representing packed sample data, while the window-size signal changes from 8 to 4. The pair-bus value changes from 048241209 to 000000a05, and the index signal displays 8. These changes demonstrate the selection and generation of pair data according to the configured window size. Figure 9 presents the simulation results of the arithmetic subtractor over approximately 0–5 ns. The input signals a[15:0] and b[15:0], the subtract control, and the 17-bit result[16:0] output is displayed. Initially, the inputs are 100 and 23, producing a result of 123 when subtraction is disabled and addition is selected. When subtraction is enabled, the result

becomes 77, corresponding to 100 minus 23. The waveform also shows the input combination 65496 and 70, producing 30 under the displayed arithmetic operation.

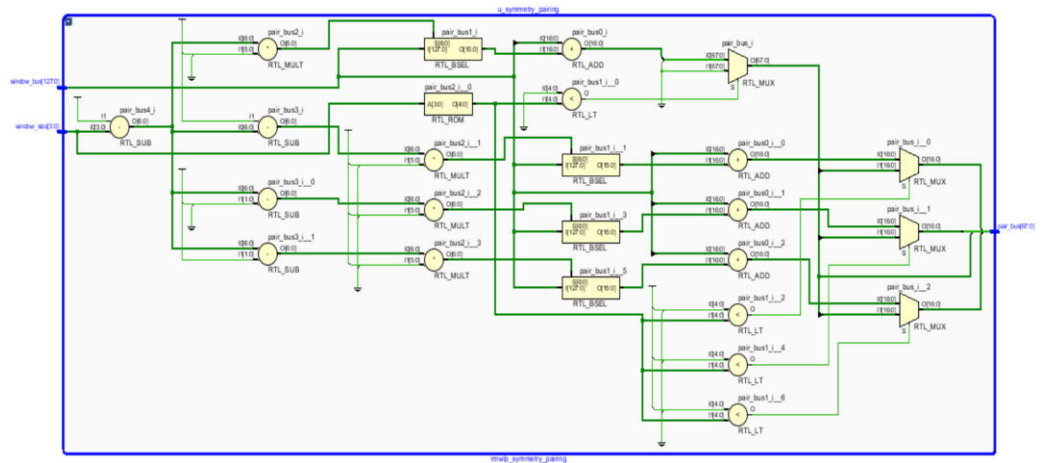


Figure 6. RTL Schematic of the Symmetry Pairing Unit.

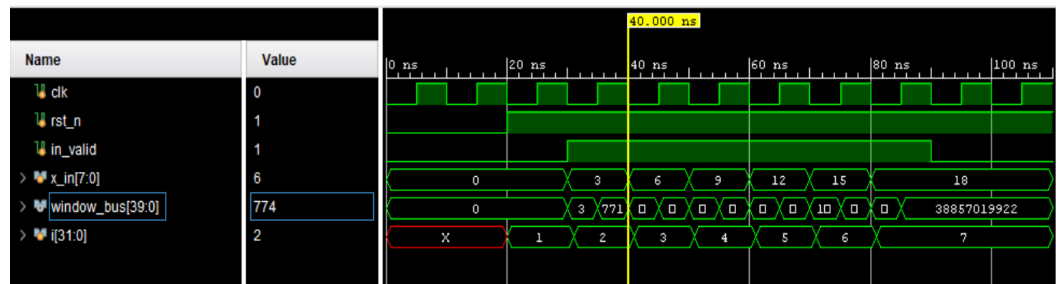


Figure 7. Simulation Waveform of the Window Buffer

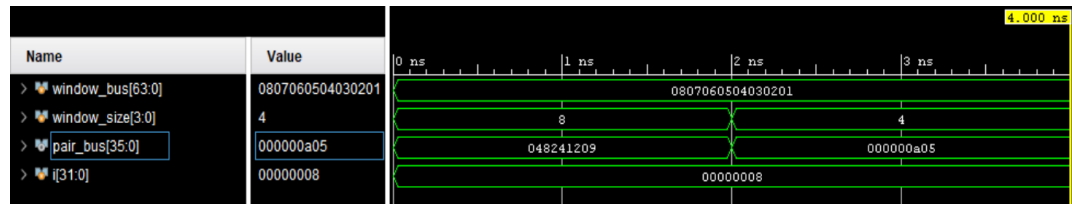


Figure 8. Simulation Waveform of the Symmetry Pairing Unit.

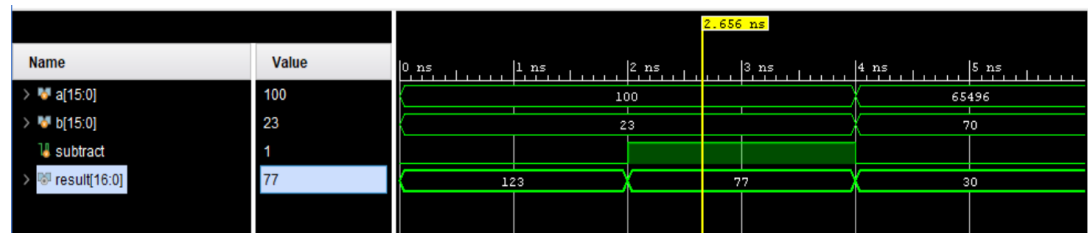


Figure 9. Simulation Waveform of the Arithmetic Unit.

Figure 10 illustrates the operation of the LCS limiter over approximately 0–7 ns. The displayed signals are value_in[15:0], limit_min[15:0], limit_max[15:0], enable, and value_out[15:0]. The minimum limit is 65436, the maximum limit is 100, and the enable signal is initially asserted. For an input value of 45, the output remains 45; for an input value of 150, the output is limited to 100. The waveform also displays an input value of 65406, which is passed as shown, followed by another input value of 150 producing an output of 150 when the enable signal is inactive. Figure 11 shows the output-register simulation waveform over approximately 0–60 ns. The signals include clk, rst_n, in_valid, data_in[15:0], out_valid, and data_out[15:0]. The input data sequence includes hexadecimal values 0000, 006f, ffe7, and 03e7, corresponding to 0, 111, –25 in signed 16-bit representation, and 999, respectively. The output register transfers the input data to the output according to the clock and control

conditions, with out_valid indicating the validity of the output. At the selected cursor of 56 ns, the displayed clock is 1, reset is deasserted, in_valid is 0, and data_out is ffe7. The waveform demonstrates sequential data transfer and output-valid control.

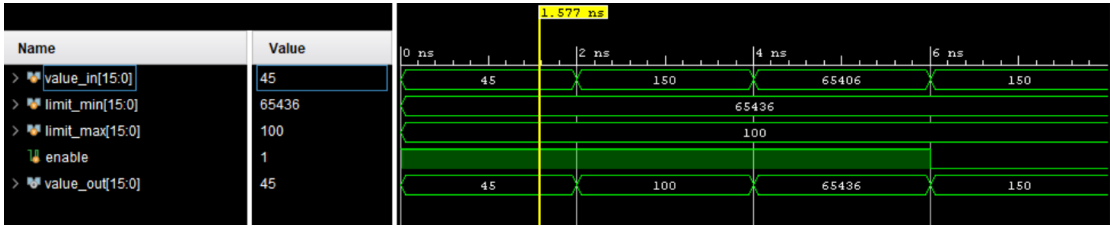


Figure 10. Simulation Waveform of the LCS Limiter

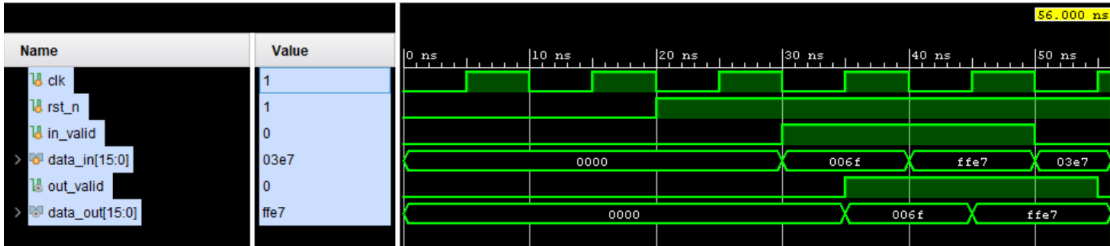


Figure 11. Simulation Waveform of the Output Register

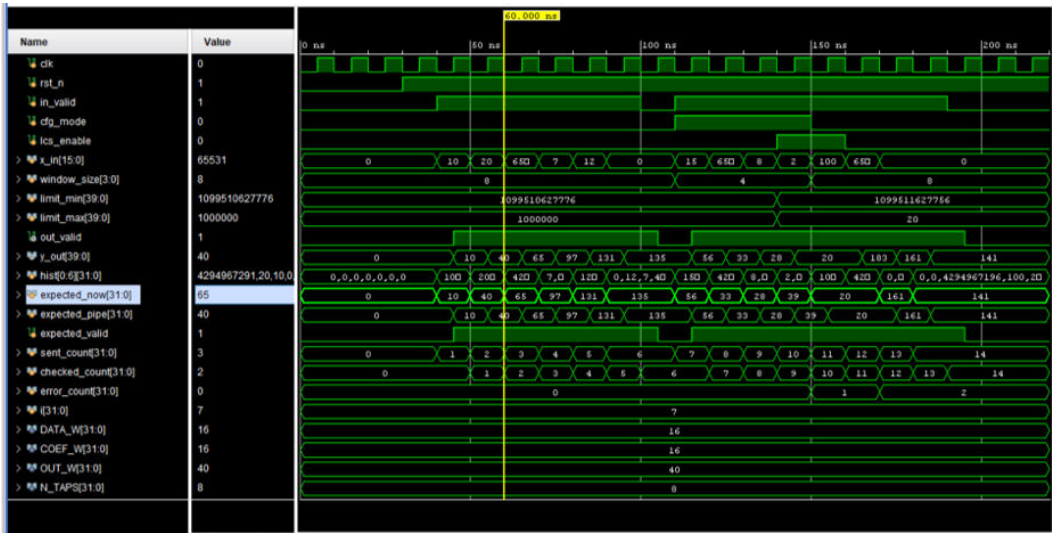


Figure 12. Overall Simulation Waveform of RMWLP-FIR Filter.

Figure 12 presents the integrated simulation waveform of the RMWLP-FIR filter over approximately 0–240 ns. The displayed signals include clk, rst_n, in_valid, cfg_mode, lcs_enable, x_in[15:0], window_size[3:0], limit_min[39:0], limit_max[39:0], out_valid, y_out[39:0], the history bus, expected output, expected-valid signal, and the sent, checked, and error counters. The configuration switches between window sizes 8 and 4, demonstrating the reconfigurable operation of the filter. The reported parameters include DATA_W = 16, COEF_W = 16, OUT_W = 40, and N_TAPS = 8. At the selected cursor of 60 ns, x_in is 65531, the window size is 8, and y_out is 40. Figure 13 presents the FPGA resource utilization of the RMWLP-FIR architecture. The design uses 296 LUTs out of 133,800 available LUTs, corresponding to approximately 0.22% utilization. It requires 153 FFs out of 267,600 available FFs, corresponding to 0.06% utilization, and 3 DSP blocks out of 740 available DSPs, corresponding to approximately 0.41% utilization. The design also uses 146 I/O ports out of 500 available ports, giving 29.20% utilization, and 1 BUFG out of 32 available BUFGs, corresponding to 3.13% utilization. The low LUT, FF, and DSP utilization indicates that the implementation occupies a small portion of the available FPGA logic and arithmetic resources.

Resource	Utilization	Available	Utilization...
LUT	296	133800	0.22
FF	153	267600	0.06
DSP	3	740	0.41
IO	146	500	29.20
BUFG	1	32	3.13

Figure 13. Resource Utilization of the RMWLP-FIR Architecture.

Figure 14 illustrates the estimated power consumption of the RMWLP-FIR architecture. The reported dynamic power is 41.669 mW, comprising 3.565 mW of signal power, 2.064 mW of logic power, 1.357 mW of DSP power, and 34.684 mW of I/O power. The dynamic components sum to 41.670 mW, with the 0.001 mW difference from the reported 41.669 mW attributable to rounding. Static power is 0.407 mW, giving a reported total power of approximately 42.076 mW. I/O power accounts for approximately 83% of the displayed dynamic-power breakdown, making it the dominant component. Figure 15 presents the setup timing report for the RMWLP-FIR architecture, showing ten reported paths. The total path delays range from 24.974 ns to 25.450 ns, the logic delays range from 12.450 ns to 12.722 ns, and the net delays range from 12.252 ns to 13.000 ns. The maximum displayed total delay is 25.450 ns, consisting of 12.450 ns of logic delay and 13.000 ns of net delay. Figure 5.22 shows the hold timing report for the RMWLP-FIR architecture, with ten displayed path delays ranging from 0.324 ns to 0.358 ns. The logic delays range from 0.193 ns to 0.223 ns, while the net delays range from 0.124 ns to 0.165 ns. The maximum displayed total path delay is 0.358 ns, comprising 0.193 ns of logic delay and 0.165 ns of net delay. The listed paths primarily connect registers within the window buffer and one logic level and one route, with reported high-fanout values of 2 or 3. These short paths are relevant to hold analysis because data can arrive too quickly to satisfy a register's minimum-delay requirement.

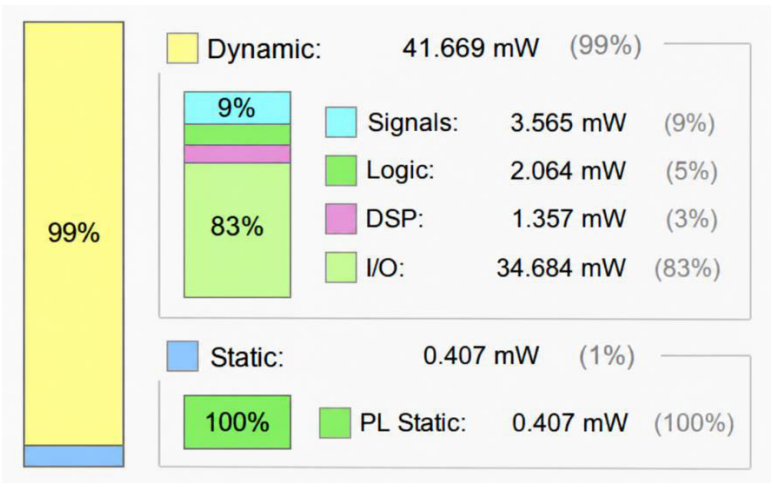


Figure 14. Power Consumption of the RMWLP-FIR Architecture

General Information	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Timer Settings	∞	13	10	150	window_size[0]	u_output_regist..a_out_reg[4]D	25.450	12.450	13.000
Design Timing Summary	∞	13	10	150	window_size[0]	u_output_regist..a_out_reg[6]D	25.445	12.450	12.995
Check Timing (756)	∞	13	10	150	window_size[0]	u_output_regist..a_out_reg[2]D	25.262	12.450	12.812
Intra-Clock Paths	∞	15	12	150	window_size[0]	u_output_regist..a_out_reg[13]D	25.143	12.722	12.421
Inter-Clock Paths	∞	15	12	150	window_size[0]	u_output_regist..a_out_reg[7]D	25.136	12.722	12.414
Other Path Groups	∞	15	12	150	window_size[0]	u_output_regist..a_out_reg[5]D	25.133	12.722	12.411
User Ignored Paths	∞	15	12	150	window_size[0]	u_output_regist..a_out_reg[11]D	25.068	12.722	12.345
Unconstrained Paths	∞	13	10	150	window_size[0]	u_output_regist..a_out_reg[0]D	25.000	12.450	12.550
NONE to NONE	∞	13	10	150	window_size[0]	u_output_regist..a_out_reg[3]D	24.996	12.450	12.546
Setup (10)	∞	15	12	150	window_size[0]	u_output_regist..a_out_reg[14]D	24.974	12.722	12.252

Figure 15. Setup Timing Analysis of RMWLP-FIR Architecture

General Information	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Timer Settings	1	1	2	u_window_buffer/d2_reg[23]C	u_window_buffer/d3_reg[23]D	0.324	0.193	0.131
Design Timing Summary	1	1	2	u_window_buffer/d2_reg[1]C	u_window_buffer/d3_reg[1]D	0.327	0.193	0.134
> Check Timing (1442)	1	1	2	u_window_buffer/d3_reg[48]C	u_window_buffer/d4_reg[48]D	0.328	0.193	0.135
Intra-Clock Paths	1	1	2	u_window_buffer/d2_reg[6]C	u_window_buffer/d3_reg[6]D	0.334	0.193	0.141
Inter-Clock Paths	1	1	2	u_window_buffer/d2_reg[57]C	u_window_buffer/d3_reg[57]D	0.347	0.223	0.124
Other Path Groups	1	1	2	u_window_buffer/d2_reg[12]C	u_window_buffer/d3_reg[12]D	0.347	0.193	0.154
User Ignored Paths	1	1	3	u_window_buffer/d1_reg[14]C	u_window_buffer/d2_reg[14]D	0.353	0.193	0.160
Unconstrained Paths	1	1	2	u_window_buffer/d3_reg[47]C	u_window_buffer/d4_reg[47]D	0.353	0.193	0.160
NONE to NONE	1	1	2	u_window_buffer/d3_reg[13]C	u_window_buffer/d4_reg[13]D	0.356	0.193	0.163
Setup (10)	1	1	2	u_window_buffer/d2_reg[59]C	u_window_buffer/d3_reg[59]D	0.358	0.193	0.165
Hold (10)								

Figure 16. Hold Timing Analysis of RMWLP-FIR Architecture

4.2 Comparative Analysis

Table 1 compares the hardware resource utilization of the proposed RMWLP-FIR architecture with COBAD-FIR, RHO-FIR, BKAVM-FIR, and FSM-FIR. The COBAD-FIR architecture requires 1,050 LUTs, 850 FFs, 8 DSPs, and 4 BUFGs, whereas RHO-FIR requires 820 LUTs, 650 FFs, 6 DSPs, and 3 BUFGs. BKAVM-FIR utilizes 650 LUTs, 480 FFs, 5 DSPs, and 2 BUFGs, while FSM-FIR requires 480 LUTs and 320 FFs. The proposed RMWLP-FIR achieves the lowest resource utilization, requiring only 296 LUTs, 153 FFs, 3 DSPs, and 1 BUFG. These results demonstrate that the proposed architecture improves hardware resource efficiency through symmetry pairing, reconfigurable multi-window processing, and arithmetic resource optimization.

Table 2 presents the timing performance comparison of the proposed RMWLP-FIR architecture with COBAD-FIR, RHO-FIR, and BKAVM-FIR. The setup path delays of COBAD-FIR, RHO-FIR, and BKAVM-FIR are 34.000 ns, 29.000 ns, and 27.000 ns, respectively, whereas the proposed RMWLP-FIR achieves the lowest setup path delay of 25.450 ns. Its setup logic delay and setup net delay are 12.450 ns and 13.000 ns, respectively. The hold path delays are 0.650 ns, 0.500 ns, 0.420 ns, and 0.358 ns for COBAD-FIR, RHO-FIR, BKAVM-FIR, and the proposed RMWLP-FIR, respectively. The proposed architecture also achieves the lowest hold logic delay of 0.193 ns and hold net delay of 0.165 ns. Consequently, the combined setup-plus-hold delay decreases from 34.650 ns for COBAD-FIR, 29.500 ns for RHO-FIR, and 27.420 ns for BKAVM-FIR to 25.808 ns for the proposed RMWLP-FIR, demonstrating improved timing performance.

Table 1. Area Comparison of RMWLP-FIR with Existing Methods

FIR Architecture	LUTs	FFs	DSPs	BUFG
COBAD-FIR	1,050	850	8	4
RHO-FIR	820	650	6	3
BKAVM-FIR	650	480	5	2
FSM-FIR	480	320	4	2
Proposed RMWLP-FIR	296	153	3	1

Table 3 compares the power consumption of the proposed RMWLP-FIR architecture with COBAD-FIR, RHO-FIR, and BKAVM-FIR. COBAD-FIR consumes 6.500 mW of signal power, 4.500 mW of logic power, 2.500 mW of DSP power, and 41.000 mW of I/O power, resulting in a dynamic power consumption of 54.500 mW, static power of 0.600 mW, and total power of 55.100 mW. RHO-FIR reduces the total power consumption to 49.050 mW, with dynamic and static power consumptions of 48.500 mW and 0.550 mW, respectively. BKAVM-FIR further reduces the total power to 45.200 mW, with dynamic power of 44.700 mW and static power of 0.500 mW. In comparison, the proposed RMWLP-FIR achieves the lowest total power consumption of 42.076 mW, comprising 3.565 mW of signal power, 2.064 mW of logic power, 1.357 mW of DSP power, and 34.684 mW of I/O power. Its dynamic and static power consumptions are 41.669 mW and 0.407 mW, respectively. These results demonstrate that the proposed architecture reduces power consumption through optimized arithmetic operations and efficient hardware resource utilization.

Table 2. Delay Comparison of RMWLP-FIR with Existing Methods

Timing Parameter	COBAD-FIR	RHO-FIR	BKAVM-FIR	Proposed RMWLP-FIR
Setup Path Delay (ns)	34.000	29.000	27.000	25.450
Setup Logic Delay (ns)	17.000	14.500	13.500	12.450
Setup Net Delay (ns)	17.000	14.500	13.500	13.000
Hold Path Delay (ns)	0.650	0.500	0.420	0.358
Hold Logic Delay (ns)	0.350	0.270	0.230	0.193
Hold Net Delay (ns)	0.300	0.230	0.190	0.165
Total Delay (ns)	34.650	29.500	27.420	25.808

Table .3. Power Consumption Comparison of RMWLP-FIR with Existing Methods

Power Parameter	COBAD-FIR	RHO-FIR	BKAVM-FIR	Proposed RMWLP-FIR
Signal Power (mW)	6.500	5.000	4.200	3.565
Logic Power (mW)	4.500	3.500	2.800	2.064
DSP Power (mW)	2.500	2.000	1.700	1.357
I/O Power (mW)	41.000	38.000	36.000	34.684
Dynamic Power (mW)	54.500	48.500	44.700	41.669
Static Power (mW)	0.600	0.550	0.500	0.407
Total Power (mW)	55.100	49.050	45.200	42.076

4.3 MATLAB Results

The magnitude response of RMWLP-FIR presented in Figure 17 compares the Least-Squares, Gaussian, Hamming, Hann, and Blackman filter designs using quantized coefficients. The results illustrate differences in cutoff behavior, transition regions, and stopband attenuation. The Gaussian response exhibits comparatively high stopband sidelobes, whereas the Hann and Blackman responses show stronger attenuation in several frequency regions. These differences demonstrate the influence of coefficient design and window selection on the frequency-domain characteristics of the RMWLP-FIR filtering process. The phase response of RMWLP-FIR presented in Figure 18 illustrates the phase characteristics of the five FIR filter designs after coefficient quantization. The approximately linear phase trends are consistent with symmetric FIR coefficients, while the discontinuities and gaps at higher frequencies arise from phase wrapping and low-magnitude regions near spectral nulls. The phase response should be assessed primarily in the passband, where the filter preserves the relative phase relationships of signal components. For a 64th-order symmetric FIR filter, the theoretical group delay is 32 samples. The input-referenced FFT comparison presented in Figure 19 evaluates the original input spectrum against the filtered spectra obtained using the Least-Squares, Gaussian, Hamming, Hann, and Blackman designs. The input signal contains frequency components that are passed or attenuated according to each filter's frequency response. The output spectra demonstrate attenuation of unwanted frequency components and illustrate the effects of filter design on spectral suppression. Using a common input-referenced scale enables a more meaningful comparison of output amplitudes than independently normalizing each filtered spectrum.

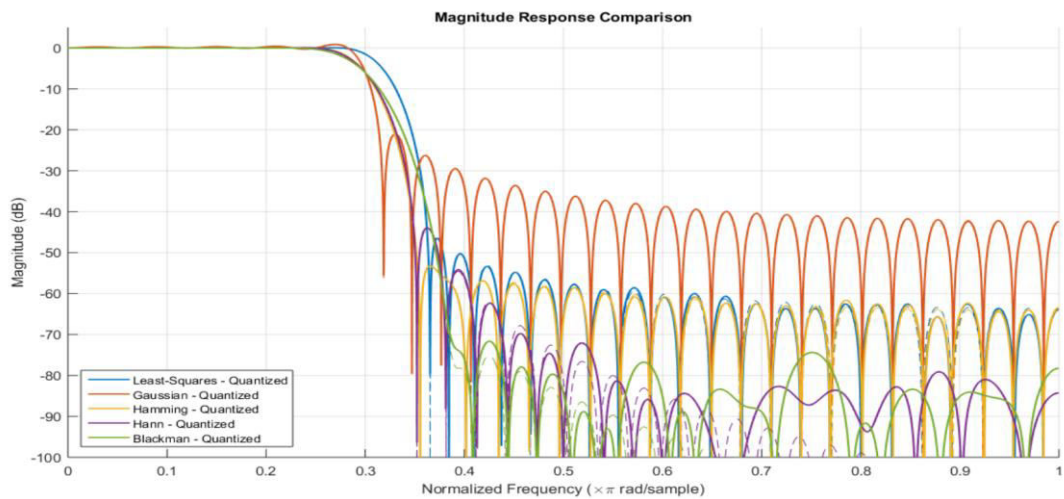


Figure 17. Magnitude Response Comparison of RMWLP-FIR with Different Window Functions

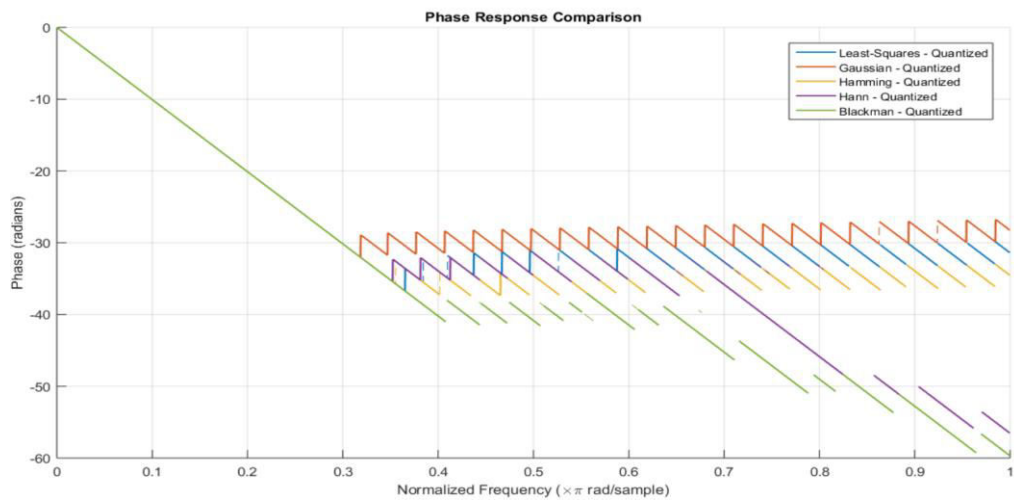


Figure 18. Phase Response Comparison of RMWLP-FIR with Different Window Functions

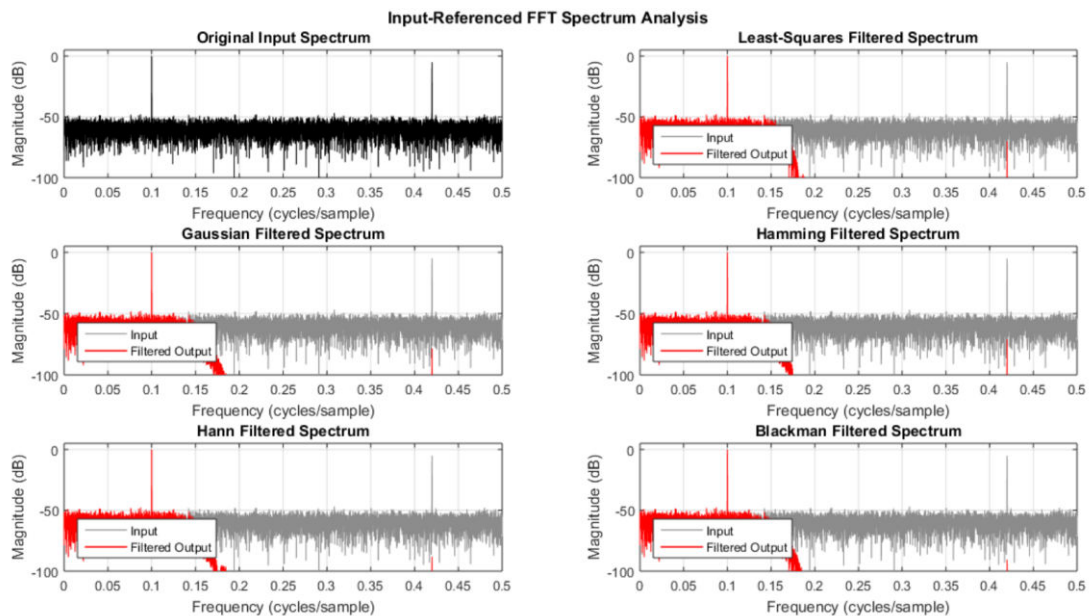


Figure 19. Input-Referenced FFT Spectrum Analysis of RMWLP-FIR

5. Conclusion

The proposed RMWLP-FIR filter demonstrates efficient hardware utilization, reduced timing delays, and low power consumption for FPGA-based DSP. The implementation requires 296 LUTs, 153 FFs, 3 DSP blocks, and 1 BUFG, indicating a compact hardware architecture. The reported maximum setup path delay is 25.450 ns, comprising 12.450 ns of logic delay and 13.000 ns of net delay, while the maximum displayed hold path delay is 0.358 ns. The estimated dynamic power consumption is 41.669 mW, consisting of 3.565 mW of signal power, 2.064 mW of logic power, 1.357 mW of DSP power, and 34.684 mW of I/O power; the static power is 0.407 mW, resulting in a total power consumption of approximately 42.076 mW. These results highlight the potential of configurable window processing, symmetry-based arithmetic, and resource sharing to reduce hardware complexity.

References

- [1]. Hemantha, G. Reddy, M. Tejeswar Reddy, B. Vamsi, and P. Venkata Siva Reddy. "Activity-Driven FIR Filter Floor-Planning Using Hybrid Accelerated PSO-GA Algorithm for Low-Power VLSI Applications." In 2026 Contemporary Computing Innovations Conference (CCIC), pp. 1-6. IEEE, 2026.
- [2]. Karunakaran, S., Praveen Kumar Balachandran, Vinoop Kanna, and Subhav Singh. "COBAD-FIR: a hybrid coati-badger optimization framework for power-efficient FIR coefficient enhancement in VLSI systems." *Engineering Research Express* 8, no. 10 (2026): 105216.
- [3]. Rao, D. Raghunatha, Sudharani Paradesi, and K. Sri Pavani. "VLSI Architecture of Adaptive FIR Filter Using Distributed Arithmetic for Low-Power DSP Applications." In 2026 Fifth International Conference on Power, Control and Computing Technologies (ICPC2T), pp. 985-990. IEEE, 2026.
- [4]. Odugu, Venkata Krishna, P. Ramakrishna, T. Vasudeva Reddy, and G. Harish Babu. "Optimized 2-D FIR filter bank architecture using various symmetries with parallel processing and DA." *Computers and Electrical Engineering* 131 (2026): 110941.
- [5]. Kalavathi Devi, T., S. Umadevi, M. Madhan Mohan, K. N. Baluprithviraj, A. Stephen Sagayaraj, and P. Sakthivel. "FPGA Implementation and Performance Enhancement of FIR Filter with Hybrid FinFET-GDI logic based Carry Select Adder." *Analog Integrated Circuits and Signal Processing* 128, no. 3 (2026): 23.
- [6]. Bhadavath, Kiran Kumar, and Z. Mary Livinsa. "An Optimized Retiming-based FIR Filter Architecture using Efficient Multipliers and Adders." *Recent Advances in Electrical & Electronic Engineering* 19, no. 1 (2026): E23520965352314.
- [7]. Rao, A.V.B. and Suman, S., 2026, July. Hybrid fault tolerance and protection for digital FIR filter implementations. In *AIP Conference Proceedings* (Vol. 3418, No. 1, p. 070044). AIP Publishing LLC.
- [8]. Magesh, V., Mithesh, T.S., Ponmaniselvan, S. and Kumar, P.P., 2026. Simulation and Implementation of Advanced Adder and Hybrid Multiplier for FIR Filter. *Industrial Internet of Things and Advanced Techniques for Sensor Data Aggregation and Fusion*, pp.47-59.
- [9]. Sasikala, S., S. N. Ijean, K. Kayalvizhi, K. Anusree, and S. Kamalakannan. "Design and FPGA Development of a Hybrid Multiplier Architecture with Standard Operation Scheduling for a Low-Power FIR Filter." In 2026 Fourth International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), pp. 1-6. IEEE, 2026.
- [10]. Meghana, Bukkapatnam, and B. S. Supreetha. "Modular Design and Implementation of a Reconfigurable Higher Order FIR Filter." In *Technology for a Resilient Future*, pp. 167-170. CRC Press, 2027.
- [11]. Sriram, A., Balachandran Harini, N. Rajeswari, D. Deepthi, and T. N. Prabakar. "Self-Verified FIR Filter ASIC with Vedic Arithmetic and Han-Carlson Accumulation at 90-nm." In 2026 7th International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 94-99. IEEE, 2026.

- [12]. Parhi, Keshab K. "An Iterated Hybrid Fast Parallel FIR Filter." In 2026 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1719-1723. IEEE, 2026.
- [13]. Ganjimala, Pavankumar, Vinay Chakravarthi Gogineni, and Subrahmanyam Mula. "High-throughput and area-efficient VLSI architectures for functional link adaptive filters." *Circuits, Systems, and Signal Processing* 45, no. 6 (2026): 5045-5067.
- [14]. Kumar, Rajat, and Aman Kumar. "FPGA Based Implementation Of Distributed Arithmetic-FIR Filter Design Using Approximate Karatsuba Multiplier And VLCSA." *Int. J. Sci. R. Tech* 3, no. 5 (2026).
- [15]. Saranya, C., Kumar Ponnusamy, R. Ramya, R. Jagadeesan, and D. Poornakumar. "Performance Analysis of FPGA-Based Digital Filter Architectures for Image Processing Applications." *Circuits, Systems, and Signal Processing* (2026): 1-24.
- [16]. Anuraj, V., and Dhandapani Vaithiyanathan. "Study of FIR Filter Hardware Architecture for Real-Time Multimedia Applications." *Machine Learning in Nanoelectronics: Devices, Circuits and Systems* (2026): 343-362.
- [17]. Abinaya, A., N. Radha, and V. Vinodhini. "Optimized Polyphase FIR Filtering: A Pragmatic Approach for Next-Generation Wireless Systems." In 2026 International Conference on Next Generation Electronics (NELEX), pp. 1-5. IEEE, 2026.
- [18]. Kannan, Loganathan Mohana, Krishna Prakash Arunachalam, Rudramani Bhutia, and Elangovan Muniyandy. "Low-Power VLSI Architecture for ECG Signal Detection Using Transpose Form Retimed Delayed LMS Filter with Improved Progressive Cross Sea Lion Graph Contextual Convolutional Attention Network." *Biomedical Materials & Devices* (2026): 1-23.
- [19]. Pereira, Pedro TL, Eduardo Da Costa, and Sergio Bampi. "HDL-DFGen: A Digital Filter Hardware Description Generation Framework." In 2026 27th International Symposium on Quality Electronic Design (ISQED), pp. 1-8. IEEE, 2026.
- [20]. Datta, Debarshi, and Mrinal Kanti Naskar. "Implementation of High-efficient FIR filter using reversible adders and multipliers for signal processing applications." *Computers and Electrical Engineering* 139 (2026): 111369.
- [21]. Banumathi, J., and G. Karthy. "High-performance FIR filter designs using Brent Kung Adder and pipelined Vedic multiplier." *Integration* (2026): 102645.
- [22]. Huang, Ziyang, Wei Deng, Haikun Jia, Lixue Kuang, Yue Liu, and Baoyong Chi. "An IEEE 802.15.4/4z-Compliant UWB All-Digital Transmitter With Hybrid FIR Filtering." *IEEE Transactions on Microwave Theory and Techniques* (2026).
- [23]. Anumothu, Murali, GM Anitha Priyadarshini, Ch Hima Bindu, Rajanikanth Aluvalu, Sai Prashanth Mallellu, Pankaj Kumar, Ghanshyam G. Tejani, and Seyed Jaleleddin Mousavirad. "An FSM-enabled reconfigurable debugging approach for area-optimized FIR filters on FPGA platforms." *IET Circuits, Devices & Systems* 2026, no. 1 (2026): 5545245.
- [24]. Krishnan, Thiruvankadam. "Optimized FPGA Architecture for Adaptive Fir Filter Using 5: 2 Compressor for Low Computational Complexity." In 2026 4th International Conference on Artificial Intelligence and Machine Learning Applications Theme: Healthcare and Internet of Things (AIMLA), pp. 1-5. IEEE, 2026.
- [25]. Goswami, Saksham, Aditi Keshri, Purnasha Dey, and P. Sudhanya. "High-performance Filter Design with Fast Binary Counters and Sorting." *VLSI, Microwave and Wireless Technologies: Select Proceedings of ICVMWT 2024* (2026): 399.
- [26]. Moryakova, Oksana, and Håkan Johansson. "Closed-form least-squares design of fast-convolution based variable-bandwidth FIR filters." *IEEE Open Journal of Signal Processing* (2026).
- [27]. Kumar, Vinod, and M. A. Majid. "High-Performance and Energy-Efficient FIR Filter Architecture Utilizing Modified Booth-Wallace Multiplier with Kogge-Stone Adder for Wireless Sensor Network in

- the IoT." In 2026 5th International Conference on Computer Networks, Big Data and IoT (ICCBI), pp. 241-247. IEEE, 2026.
- [28]. Shobha, K. R. "An Efficient FPGA-Based Architecture for Real-Time ECG Signal Acquisition and FIR Filtering." In 2026 10th International Conference on Inventive Systems and Control (ICISC), pp. 636-642. IEEE, 2026.
- [29]. Khorbotly, Sami, and Fayol Ateufack Zeudom. "Optimized Shift & Add Multipliers for Efficient Filter and Neural Network Implementations." In 2026 IEEE International Conference on Electro Information Technology (eIT), pp. 387-391. IEEE, 2026.
- [30]. Aarthy, S. T., and JL Mazher Iqbal. "Low-Area FIR Filter for Noise Reduction in ECG Signal using DADDA Multiplier and Parallel Prefix Adder." *Journal of Circuits, Systems and Computers* (2026): 2550442.