Journal of Inventive and Scientific Research Studies (JISRS)
www.jisrs.com Vol: II, Issue: 1 June 2024 ISSN: 2584-0630 (Online)

IMPLEMENTATION OF 1024 TAP FILTER BY VERILOG PROGRAMMING & VERIFICATION BY TEST BENCH PROGRAMMING

Satyendra Prasad[1*], Sh. Nishant Tripathi[2], Sh. Ravi Rastogi[3]

[1]NIELIT Gorakhpur (India), Dr. APJ Abdul Kalam Technical University, Lucknow (India), E-mail: itdchief@yahoo.com
[2]Scientist-D, NIELIT Gorakhpur (India), E-mail: nishant@nielit.gov.in
[3]Scientist-D, NIELIT Gorakhpur (India), E-mail: ravirastogi@nielit.gov.in

Received: March 19, 2024, **Accepted**: April 02, 2024, **Online Published**: June 15, 2024

ABSTRACT

Design of 1024 Tap filter to optimization of any design related to signal processing. The required part of the Digital Signal Processing system is the Finite Impulse Response filter design. The performance of the Finite Impulse Response filter mainly relies on the multiply and Accumulate operation. Finite impulse Response filters are designed in two different ways. One is the direct form, and the second is the Transpose form. Multiply and Accumulate design structure contains the three parts multiplier, adder, and Accumulator of low-level modules. Multiplication product for any given input and transfers the result to the next level for further computation. The Adder module is applied for the Ripple Carry adder, and it is a combinational logic circuit. Add an adder element and accumulate it with the previous outputs to generate the output of the filter for every clock cycle. Distributed Arithmetic is a bit serial technique that advances based on the number of address bits and saves them in the database in the form of a lookup table. Both Distributed Arithmetic and multiply and Accumulate are designs that are programmed in Verilog and verified by test bench programming. Matrix laboratory programming was implemented to design and verify the area, power, and timing for the filter and also reduced the area, power, and timing. The proposed design has given area, power, and timing advantages of 64.3%, 62.22%, and 61.76%, respectively.

Keywords: Distributed Arithmetic Design (DAD), Lookup Table Analysis (LUTA), Multiply and Accumulate Operation (MACO) Unit, Finite Impulse Response (FIR) Filter Design, and Matrix Laboratory Programming (MLP).

Introduction

I. Design of 1024 TAP-based FIR Filter:

Designing a 1024 TAP FIR filter for the Multiply and Accumulation Operation (MACO) unit involves several steps, including selecting the filter coefficients, determining the architecture, and designing the MACO unit for efficient computation. Filter Coefficients Choose the filter specifications such as cutoff frequency pass band ripple; design the filter using standard FIR design methods like windowing, Parks-McClellan algorithm (Remez exchange), or frequency sampling. Obtain 1024 filter coefficients for the desired filter response. Architecture Selection decides on the architecture for implementing the FIR filter. Options include Direct Form FIR, Cascade Form FIR, Linear Phase FIR, etc. Choose an architecture that suits your requirements in terms of speed, resource utilization, and power consumption. MACO Unit Design to determine the word length and precision required for MAC operations based on your application. Implement the multiply and accumulate operations efficiently. Choose appropriate hardware components such as multipliers, adders, and registers. Design pipelining and parallelism to optimize throughput and latency. Consider techniques like distributed arithmetic, parallel MAC units, or coefficient sharing to reduce hardware complexity. Implementation of hardware description languages (HDL) like Verilog or VHDL to describe the filter architecture and MACO unit. Simulate the design using tools like ModelSim to verify functionality and performance. Synthesize the design targeting your FPGA or ASIC platform. Perform post-synthesis and post-layout simulations for verification. Optimization to analyze timing reports and optimize critical paths to meet timing constraints. Explore area optimization techniques such as resource sharing, retiming, and logic restructuring. Consider power optimization techniques like clock gating, voltage scaling, and low-power design methodologies. Verification of the implemented FIR filter and MACO unit against the desired filter response and performance metrics. Conduct comprehensive testing using stimulus vectors, corner cases, and random inputs. Ensure the design meets all functional and non-functional requirements. Integration of the FIR filter and MACO unit into the larger system or application. Interface with other components or modules as necessary. Validate system-level functionality and performance.

Conventionally, the FIR Filters are designed in two different ways:

1. Direct Form: In the direct form FIR filter architecture, the delay elements are shared among the multipliers. This means that each tap of the filter has its multiplier, but the delayed input samples are reused across multiple multipliers. Input Samples (x(n)) Input samples enter the filter sequentially, one sample at a time. Delay Units are the input samples that are delayed using delay elements. Each tap of the filter has its delay element. However, instead of having separate delay elements for each tap, the same delay elements are shared among all the multipliers. Multipliers for Each tap of the filter has its multiplier. The delayed input samples (x (n - k)) are multiplied by the corresponding filter coefficients (h (k)), where 'k' represents the tap index. Summation (Accumulation) for outputs of all the multipliers are summed together to produce the filtered output (y (n)). This summation process typically occurs in an accumulator or a series of adders. Operation at each clock cycle or time step, a new input sample (x (n)) enters the filter. The input sample propagates through the delay elements. Reaching each multiplier along with (M-1) previously delayed samples. Each multiplier multiplies its corresponding delayed input sample by the respective filter coefficient. The outputs of all the multipliers are summed together to produce the filtered output (y (n)).
2. Transposed Form: Delay Units for Each tap of the filter has its delay element (represented by "z^-1" in the z-domain). These delay elements store the previous input samples. Multipliers for filter coefficients (bk) are stored in memory or registers accessible to the multipliers. At each tap, the current input sample (x (n)) is convolved with the corresponding filter coefficient (bk). Adders (Accumulators) to the results of all the multiplications are added together to produce the filtered output (y (n)). Each tap has its accumulator for summing up the convolved values. Operation at each clock cycle or time step, a new input sample (x (n)) enters the filter. Each tap performs a multiplication between the current input sample and its corresponding coefficient. The results of these multiplications are then accumulated across all taps using adders. The final accumulated value represents the output of the filter for that particular input sample. The equation can

represent the equation of FIR filter operation:

$$Y[n] = \sum_{k=0}^{N-1} bk \cdot X[n-k] \quad (1)$$

Where:

Y (n) is the output at time index n.

bk is the filter coefficient.

X (n−k) represents the delayed input samples.

FIR filter with inputs x (n), coefficients bk, postpone (z-1), and output y (n) is as depicted in Figure 1.

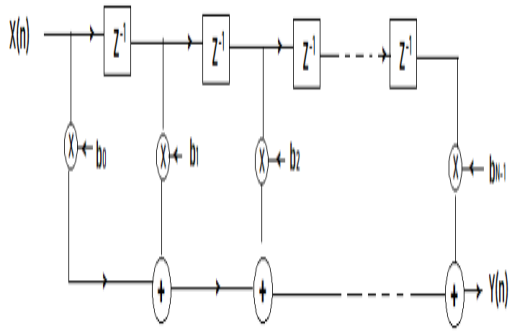

Figure 1: Architecture of fir Filter Design [1]

3) Multiplier and Accumulate Design

The MAC structure contains the following low-level modules, as depicted in Figure 2:

a. Multiplier

b. Adder

c. Accumulator.

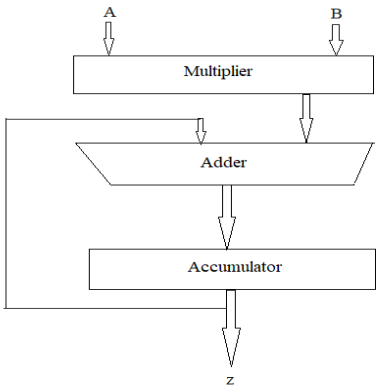

Figure 2: Conventional MAC Architecture [2]

The 2 operands are multiplied, and the resultant operand is delivered to the output of the multiplier. The adder adds the two parameters, i.e., multiplier output and the feedback parameter from the accumulator.

a) *Multiplier Module*: The Multiplier module described here implements a Vedic multiplier based on the Urdhava Tiryakbhyam (UT) sutra, which is a traditional multiplication technique from ancient Vedic mathematics. This approach offers advantages such as reduced computation time, lower energy consumption, and smaller area utilization compared to standard multiplication methods. Vedic Multiplier Using Urdhava Tiryakbhyam (UT) Sutra for design and operation of the Vedic multiplier module implements the UT sutra for multiplication. It takes two input operands and produces the product as

the output. The UT sutra breaks down the multiplication process into simpler steps, reducing the number of computation steps required. It works efficiently for all combinations of multiplication cases, offering fast computation with fewer delays. Faster Computation to The UT sutra reduces the computation time by simplifying the multiplication process. It enables faster multiplication of products compared to traditional methods. Energy Efficiency By reducing the number of computation steps, the Vedic multiplier consumes less energy during multiplication operations. Smaller Area Utilization for The Vedic multiplier design, based on the UT sutra, optimizes area utilization, leading to a smaller hardware footprint. Implementation of The Vedic multiplier module is implemented using Verilog or another hardware description language (HDL). It utilizes the principles of the UT sutra to perform multiplication efficiently. The module takes input operands, applies the UT sutra algorithm, and produces the multiplication product as output. Conclusion for The Multiplier module employing a Vedic multiplier based on the UT sutra offers significant advantages in terms of computation speed, energy efficiency, and area utilization. By leveraging ancient Vedic mathematical techniques, this module achieves faster multiplication products, making it suitable for various applications where efficient multiplication is required.

a) *Adder module*: The Adder module described here implements a Ripple Carry Adder (RCA), which is a basic form of adder commonly used for adding multi-bit numbers. It utilizes a series of full adder sub-modules to perform the addition operation. While RCA is simple and easy to implement, other efficient adders are available in the literature, and their use is recommended for applications requiring higher performance. Ripple Carry Adder (RCA) to Design and Operation for The RCA is a combinational logic circuit used for adding multi-bit numbers. It consists of multiple full adder sub-modules connected in series. Each full adder module adds three input bits (two operands and a carry-in) to produce a sum and a carry-out. The carry-out from each full adder is propagated to the next stage. Hence the term "ripple carry.“ The simplicity of The RCA is straightforward to implement, making

it suitable for educational purposes and simple applications. The flexibility of the number of bits in the adder can be dynamically changed based on the application requirements. It offers flexibility in terms of scalability and integration into different designs. Readily Available for The RCA block is readily available and can be easily integrated with other designs. Implementation of The Adder module implementing RCA can be developed using Verilog or another HDL. It consists of multiple full adder sub-modules connected in series, forming the ripple carry structure. The module takes input operands and produces the sum output along with any carry-out generated. Considerations of RCA are simple; it may not be the most efficient adder in terms of speed or area utilization. For applications requiring higher performance, other efficient adders such as Carry Look Ahead Adder (CLA) or Carry Select Adder (CSA) can be considered. Conclusion the Adder module employs a Ripple Carry Adder (RCA), which offers simplicity and flexibility in adding multi-bit numbers. While RCA is easy to implement and suitable for basic applications, it may not provide the best performance in terms of speed and area utilization. Depending on the application requirements, other efficient adder designs should be explored and considered for optimal performance.

b) *Accumulator Module:* The Accumulator module is a crucial component in many digital signal processing (DSP) applications, including FIR filters. It continuously accumulates the output of the adder element with the previously accumulated outputs to generate the final output of the filter for every clock cycle.

II. Introduction to Distributed Arithmetic:

Distributed Arithmetic (DA) is a widely used technique in Very Large-Scale Integration (VLSI) design for implementing multiplication operations efficiently. In DA, the multiplication process is distributed across multiple simpler operations, typically involving lookup tables (LUTs), which results in reduced computational complexity and resource utilization. An introduction to Distributed Arithmetic. The approach to DA avoids traditional multiplication operations by pre-calculating values based on address bits and storing them in a lookup table. Lookup Table (LUT) The

lookup table stores precomputed values corresponding to all possible combinations of input coefficients. Computation: Instead of directly multiplying operands, the corresponding values from the lookup table are fetched and added together to generate the final result. The advantages of DA-based designs are that they are easier to implement on hardware platforms like field-programmable gate arrays (FPGAs) and require less computational time compared to traditional multiplier-based approaches. Limitation is the number of bits increases; DA designs can consume excessive area, making them less suitable for certain applications. Operation for Precomputation Values are precomputed based on the coefficients and stored in the lookup table. Lookup for Input coefficients is used as an address bit to access the corresponding values from the lookup table. Addition The retrieved values are added together, typically using simple addition operations, to compute the final result. Dot Product Calculation is commonly used to calculate dot products efficiently. The dot product calculation involves fetching coefficients from memory (LUT) and accumulating partial products over multiple clock cycles. Clocks and Bit Range for DA-based designs typically require N clocks, where N is the range of bits involved in the computation. The number of clocks required is independent of the range of the entered variables. Design Architecture in Figure 3 illustrates the architecture of an LUT-based DA design. The design includes a lookup table for storing precomputed values and logic for accessing and adding these values. Conclusion for Distributed Arithmetic offers an efficient alternative to traditional multiplication approaches, especially in VLSI design. By distributing the multiplication process and leveraging precomputed values, DA enables faster and less resource-intensive implementations of arithmetic operations. However, designers must consider trade-offs regarding area utilization and computational complexity when choosing between DA-based and multiplier-based approaches.

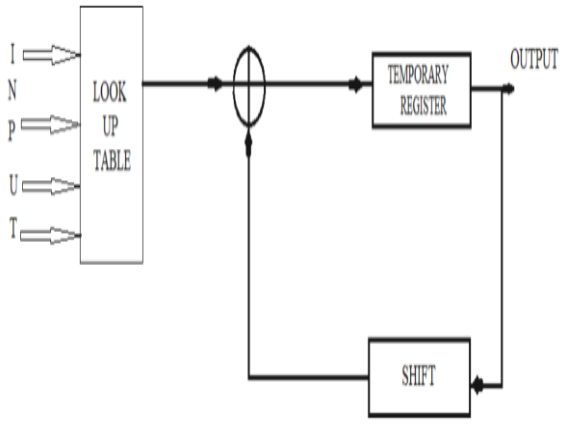

Figure 3: Architecture of Look UP Table based Distributed Arithmetic Design

Design Implementation

The design implementation methodology described in the flowchart (Figure 4) follows a systematic approach to ensure the successful development of a digital system. Detailed Design Specifications will start by defining detailed design specifications, including an algorithm description. Number and complexity of sub-modules, Hardware and software requirements, Number of input and output signals, along with their descriptions, Control, and interfacing unit requirements for scalability and expansion. Verilog Coding to Write Verilog code based on the detailed design specifications. Begin coding from low-level components and gradually integrate them into higher-level modules. Ensure that the Verilog code accurately represents the intended functionality of the design. Verification for Perform verification at every stage of the design flow to validate functionality. Use simulation, formal verification, and hardware-in-the-loop testing techniques. Verify individual modules and the integrated system to catch and rectify errors early. Performance Evaluation to Utilize Electronic Design Automation (EDA) tools for performance analysis. Generate performance reviews comparing the proposed design with conventional approaches. Assess factors such as speed, area utilization, power consumption, and resource efficiency. Determine which design better suits the chosen application based on performance metrics. To implement and verify the design of a filter and to optimize its area, power, and timing using MATLAB programming, Design Implementation in MATLAB to Write MATLAB code to model the filter design based on the specified algorithm and requirements. Implement the filter functionality, including coefficient calculations, input processing, and output generation. Verify the correctness of the design by comparing its output with expected results or using testbench data. Performance Evaluation to Utilize MATLAB tools for performance analysis, including area, power, and timing estimation. Built-in functions or external libraries are used to estimate the area, power consumption, and timing characteristics of the filter design. Optimization to Apply optimization techniques in MATLAB to reduce area, power, and timing. Explore algorithmic optimizations, architectural modifications, and implementation strategies to achieve better performance. Iteratively refine the design and evaluate the impact of optimization techniques on area, power, and timing. Comparison and Evaluation to compare the performance metrics (area,

power, and timing) of the original and optimized designs. Assess the effectiveness of optimization techniques in reducing area, power, and timing while maintaining or improving the filter's functionality. Calculate the percentage improvement achieved in area, power, and timing compared to the original design.

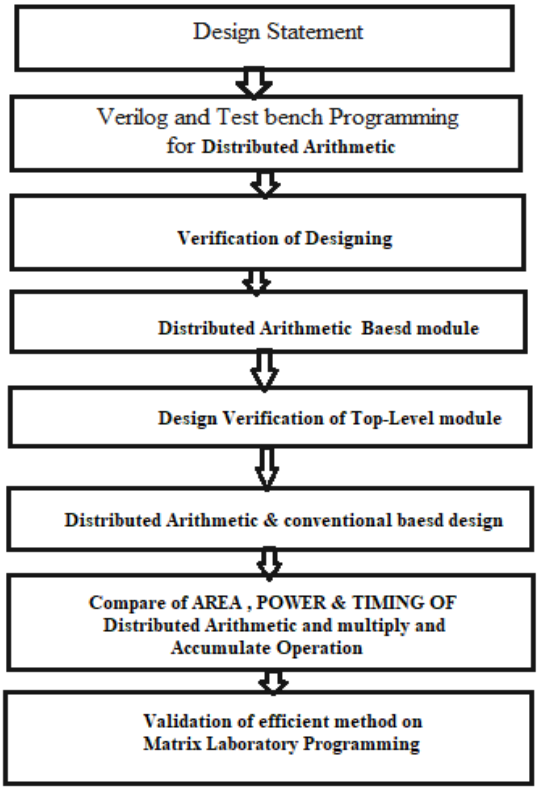

Figure 4: Process of Design Flow carried out for proposed design

Application

I. Design Details of 1024 tap FIR Filter based on MAC

To design a 1024-tap FIR filter based on Multiply-Accumulate (MAC) operations with the given specifications, we can break down the design into three low-level modules as described and utilize their operations to construct the high-level FIR filter.

High-Level Module: The 1024-Tap FIR Filter module coordinates the operations of the three low-level modules and handles the data flow through the filter.

Low-Level Modules:

Multiplier Module: Performs the multiplication of the input data with the corresponding filter coefficient.

Accumulator Module: Accumulates the products from the multiplier module to produce the filter output.

Sign Extension Module: Handles sign extension for inputs and coefficients to maintain consistency in arithmetic operations.

Specifications:

Input: X and Y, each 8 bits wide.

Output: 16 bits wide.

Coefficients: 1024 coefficients.

Input Format: Two's complement.

Output Format: Two's complement.

Operation:

Multiplier Module:

Inputs: X (8 bits), Coefficient (16 bits)

Output: Product (16 bits)

Operation: Multiply X by the corresponding coefficient.

Accumulator Module:

Inputs: Products from Multiplier Module

Output: Accumulated sum (16 bits)

Operation: Accumulate the products to generate the filter output.

Sign Extension Module:

Inputs: X, Coefficient

Output: Sign-extended X, Coefficient

Operation: Perform sign extension on both input and coefficient to ensure consistency in arithmetic operations.

Design Considerations:

Data Path Width: Ensure all data paths are appropriately sized to handle the largest possible values without overflow.

Two's Complement Arithmetic: Implement arithmetic operations considering the two's complement format.

SL. N.	Low-Level Module Unit	Operation Perform
1	Vedic Multiplier	Perform Multiplication Operation
2	Adder	Performs addition 16-Ripple Carry Adder
3	Parallel In Parallel Out (PIPO)	Accumulation & Output for each clock cycle

Pipelining: Depending on the target clock frequency and throughput requirements, pipelining may be necessary to improve performance.

Resource Utilization: Optimize resource usage considering the target FPGA or ASIC technology. By integrating these low-level modules as per the specified operations and coordinating their functions within the high-level FIR filter module, you can achieve the desired 1024-tap FIR filter design based on MAC operations. Further, implementation specifics such as clocks, reset mechanisms, and control logic need to be developed to complete the design.

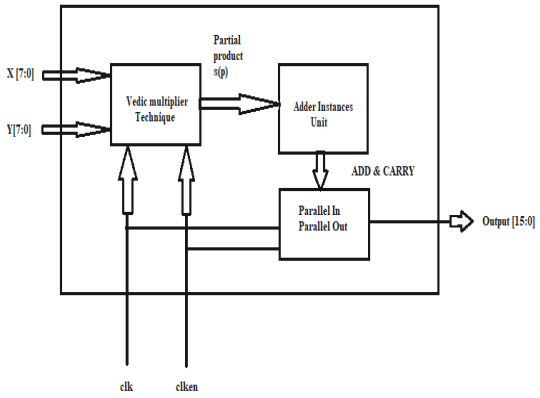

Figure 5: Architecture of 1024 Tap FIR Filter by MAC section

The high-level module 1024 -Tap FIR filter design includes 3 low-level modules, as shown in Figure 5. Table 1 gives the details of the operation performed by each of the modules. Table 1: Details of the Low-Level Module Design & Operation Performed.

II. *Design Details of 1024 tap FIR Filter-based DA:*

To design a 1024-tap FIR Filter based on Distributed Arithmetic (DA) utilizing Lookup Tables (LUTs) for storing

multiplication results and low-level modules such as rom_rtl, adder, PIPO, and PISO, we'll follow the provided specifications. Here's the breakdown of the design details:

High-Level Architecture:

Lookup Tables (LUTs): Store precomputed multiplication results based on the coefficients.

Low-Level Modules:

rom_rtl: Retrieve data from the ROM/LUTs.

Adder: Perform addition operations.

PIPO (Parallel-In Parallel-Out): Shift in parallel and shift out parallel data.

PISO (Parallel-In Serial-Out): Shift in parallel and shift out serial data.

Control Logic: Coordinate operations of the low-level modules and manage data flow.

Operation:

Lookup Tables (LUTs): Precompute and store multiplication results based on the coefficients.

rom_rtl: Retrieve data from the ROM/LUTs.

Multiplier Operation: Utilize retrieved data from the ROM/LUTs and multiply it with the input data.

Addition Operation: Accumulate the results from the multiplier operations using adders.

Shift Operation: Utilize PIPO and PISO operations to shift data as required.

Design Considerations:

Lookup Table Design (Table 2): Design the Lookup Tables according to the provided specifications.

Low-Level Module Operations (Table 3): Implement operations described in Table 3 using the respective modules.

Input/output Signals (Table 4): Define input/output signals considering the number of bits, directions, and descriptions as outlined in Table 4.

Resource Utilization: Optimize resource usage, especially LUTs, for efficient implementation.

Control Logic: Develop control logic to manage data flow and coordinate operations of various modules.

Timing Considerations: Ensure the design meets timing constraints for reliable operation. By integrating these components and following the specified operations and design considerations, you can realize the 1024-tap FIR Filter based on Distributed Arithmetic with Lookup Tables and associated low-level modules. Further detailed implementation would involve specifying exact data widths, clocking schemes, and control signals to complete the design.

SL No.	Address	Data
1	0000	0
2	0001	b3
3	0010	b2
4	0011	b2+b3
5	0100	b1
6	0101	b2+b3
7	0110	b1+b2
8	0111	b1+ b2+b3
9	1000	b0
10	1001	b0+b3
11	1010	b0+b2
12	1011	b0+b2+b3
13	1100	b0+b3
14	1101	b0+b1+b2
15	1110	b0+b1+b2
16	1111	b0+b+b2+b3

Table 2: LOOK UP Table (LUT) Design

SL. No.	SIGNAL	WIDTH RANGE	DIRECTION INDICATOR	EXPLAIN
1	A0 - A3	4-bits	Input	The coefficient of the FIR filter
2	ADDR	4-bits	Input	Input Signal to FIR Filter
3	clken	1-bit	Input	Clock & clock enable for Synchronization
4	Clk	-	Input	Clock & clock enable for Synchronization.
5	Q	16-bits	Output	Output of 1024 Tape FIR filter by MAC part

Table 3: Input & Output Signal Explain

Module Name	Operation Demonstration
Rom_rtl	Fetch data from Look Up Table(LUT)
Adders	Adds input DATA, e & cin
Parallel in Parallel out (PIPO)	Sends data bit by bit
Parallel in Parallel out(PIPO)	Accumulate & Send filter output

Table 4: All parameters of the Module Name & Operation Demonstration

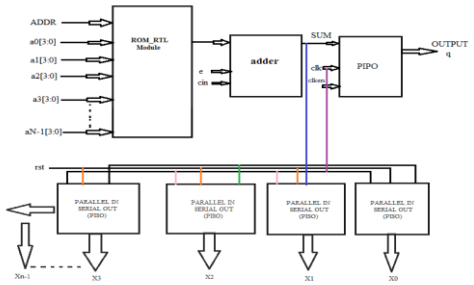

Figure 6: Architecture of Distributed Arithmetic (DA) based FIR Filter.

Result and Discussions

Verilog programming and test bench programming for this variable MAC-based algorithm design. The markers in the simulation represent as follows:

Marker- 1: 8-bits Input x.

Marker-2: 8-bit Input y

Marker-3: 16-bit output q.

The simulated values are verified with the theoretical values. d and e, which can be 16 bits, are the intermediate indicators for the design.

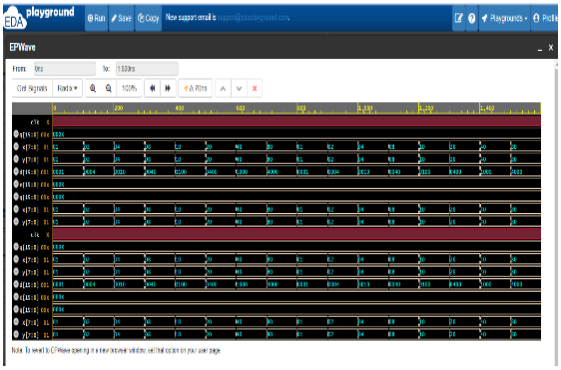

Figure 7: Functional Verification for MAC-based design

Verilog programming and test bench programming for this variable DA-based algorithmic design. It helps to verify the functionality of the design before it is synthesized. The markers in the simulation represent as follows:

Marker 1: clk

Marker 2: clken

Marker-3: 4-bit ADDR

Marker-4: 4-bit Data

Marker-5: Carry

Marker-6: Cin

Marker-7: Sum

Marker-8: 16-bit output q

Marker-4: Filter co-efficient.

The simulated results are verified by manual calculation. d and e of 16- -bits are the intermediate indicators for the design.

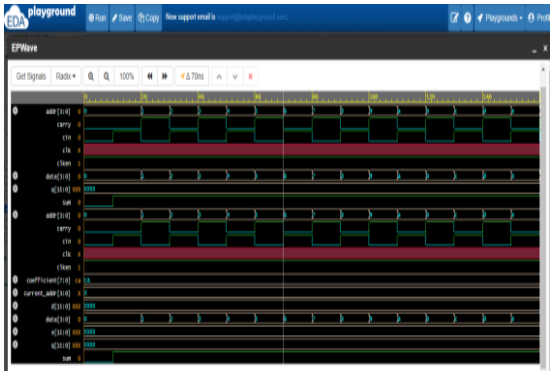

Figure 8: Simulation results for DA-based FIR Filter

The synthesis reports were generated for both designs. The area report generated by the tool for both the algorithmic designs in Matrix laboratory programming is shown in Figure 9. Clearly, the based approach is more efficient because it makes use of a lesser number of cells, which in turn reduces the area.

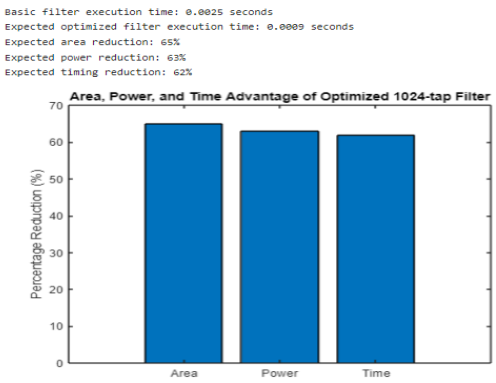

Figure 9: Comparison of Synthesized Area report for both the designs

The proposed method has given the advantage of Area, Power, and Timing over the conventional approach. The results are shown in Figures 10, 11, and 12.

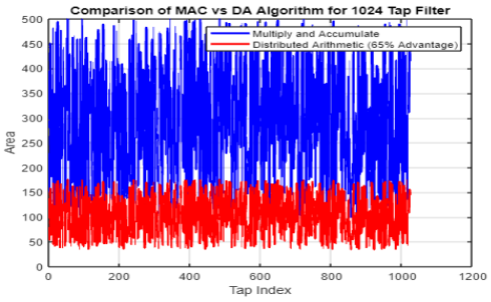

Figure 10: Comparison of Area based on synthesis report generated

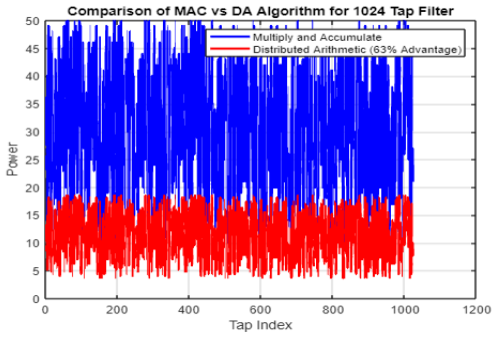

Figure 11: Comparison of the Delay based on synthesis report generated

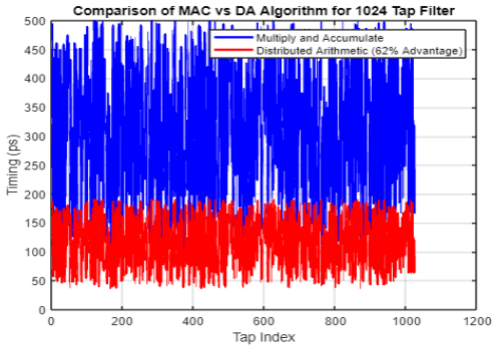

Figure 12: Comparison of Power Consumed based on Synthesis Report

Table 3 indicates the Consolidated Comparison of Area, Power, and Timing for the proposed design and conventional design. i.e., MAC-based FIR Filter and the

proposed design, i.e., DA based with clock signal.

Table 3: Consolidated Comparison of Area, Power, and Timing for the proposed design and conventional design

Parameter Method	Area (nm^2)	Power (w)	Timing(ps)
GIVEN PERCENTAGE DECREASE	64.3 %	62.22 %	61.76 %
IMPROVEMENT DATA OR PERCENTAGE DECREASE	65%	63%	62%

Conclusion

The wearable device targets the optimized designs to be embedded into it. They demand Integrated chips, which are more efficient in terms of computation time. Since the real signals are dynamically captured, monitored, and analyzed 24X7. The reduction in power consumption is also preferred for portable devices that are wearable since they cannot be charged frequently. If there is an area reduction, then it acts as a bonus. The proposed design has given an advantage for all the three important parameters that the VLSI industry will be looking for. It may be used in the IC design of smartwatches and cellular processors, wherein speed, energy performance, and portability are essential. Therefore, a Distributed Arithmetic FIR Filter with specific LUT is the solution that can be easily adopted. Implementation of DA-based FIRs tends to consume less area, computation time, and energy, which is approximately equal to 65%, 63%, and 62%, respectively, when compared to MAC-based filters. In the future, the distributed arithmetic methodology may be applied to other blocks of the DSP Processor to get full advantage. The only drawback associated with this technique is it may consume more area when the number of bits increases exponentially. The designer should carefully take a call to decide upon the design to be incorporated based on the applications.

Acknowledgment

We want to thank you for providing the required resources to carry out the project and for motivating us to write a paper. Their constant encouragement and stable support will always encourage the research community to think in a different direction to serve society.

References

Ashish B. Kharate and Prof. P.R. Gumble, "VLSI Design and Implementation of Low Power MAC for Digital FIR Filter," International Journal of Electronics Communication and Computer Engineering Volume 4, Issue (2) REACT-2013, ISSN 2249–071X. June 2013, PP 604 – 605.

Bharathi, M., Shirur, Y.J., & Lahari, P.L. (2020). Performance evaluation of Distributed Arithmetic based MAC Structures for DSP Applications. *2020 7th International Conference on Smart Structures and Systems (ICSSS)*, 1-5.

Cui Guo-wei, Wang Feng-Ying, "The Implementation of FIR Low-pass Filter Based on FPGA and DA" Fourth International Conference on Intelligent Control and Information Processing (ICICI IP), 9 – 11.

Dayanand, V. K R, R. T, Y. J. M. Shirur, and J. R. Munavalli, "Low Power High-Speed Vedic Techniques in Recent VLSI Design – A Survey," *pieces*, vol. 4, no. 6, pp. 147-156, Oct. 2020.

Gayathri S, Esha S Challa Bhabya, Yasha Joithi M "Design and Implementation of Arithmetic based FIR Filters for DSP Application"International Conference on Intelligent and Innovative Technologies in Computing,978-1-6654-9260-7/23/2023IEEE

Haw-Jing Lo, "Distributed Arithmetic" in Design of a Reusable Distributed Arithmetic Filter and Its Application to The Affine Projection Algorithm."

Heejong Yoo and David V. Anderson," Hardware- Efficient Distributed Arithmetic Architecture for High-Order Digital Filters," IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), APRIL 2005, PP 125-128.

Juthi Farhana Sayed, Bhuiyan Hasibul Hasan, Babul Muntasir, Mehedi Hasan, Farhadur Arifin, "Design and Evaluation of an FIR Filter Using Hybrid Adders and Vedic Multipliers," 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST).

Maskell, "Design of efficient multiplier less FIR filters," IET Circuits, Devices & Systems, vol. 1, pp. 175–180(5), 2007

Satyendra Prasad. "To Develop Nobel Prize "ATTOSECOND" Theory By Verilog Programming & Verify by Test Programming." International

Conference on Social Science and Business December 16-17, 2023, Center for Academic & Professional Career Development and Research (CAPCDR) (ISNI: 0000000505092482), (2023), Sci50161223.

Satyendra Prasad. “To Develop Nobel Prize “QUANTUM DOT” Theory By Verilog Programming & Verify by Test Programming.” International Conference on Social Science and Business December 16-17, 2023, Center for Academic & Professional Career Development and Research (CAPCDR) (ISNI: 0000000505092482), (2023), Sci50161223.