International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

A Scalable Automated Framework For Multiply-Accumulate Unit Design in High-Performance Computing Applications

1Dugga Venkata Aswitha, 2 Dr. Gadde Suresh
1M.Tech Student, 2Professor
ECE Department
Prakasam Engineering College

Abstract

High-Performance Computing (HPC) applications require efficient and high-speed arithmetic units to handle complex computational tasks with reduced latency and power consumption. The Multiply-Accumulate (MAC) unit plays a crucial role in digital signal processing, artificial intelligence, image processing, and scientific computing systems. This paper presents a scalable automated framework for the design and implementation of MAC units optimized for high-performance computing applications. The proposed framework focuses on improving computational speed, hardware utilization, scalability, and energy efficiency through automated design techniques and modular architecture. Advanced optimization strategies such as parallel processing, pipelining, and configurable arithmetic modules are incorporated to enhance overall system performance. The framework supports flexible scalability, enabling efficient adaptation to varying computational requirements and hardware platforms. Experimental analysis demonstrates that the proposed approach achieves improved throughput, reduced delay, and optimized resource consumption compared to conventional MAC unit designs. The developed framework offers a reliable and efficient solution for next-generation HPC systems requiring high-speed arithmetic operations and scalable hardware architectures.

Keywords

Multiply-Accumulate Unit (MAC), High-Performance Computing (HPC), Scalable Framework, Automated Design, Digital Signal Processing, Parallel Processing, Pipelining, Hardware Optimization, VLSI Design, Energy Efficiency.

I. INTRODUCTION

The rapid growth of High-Performance Computing (HPC) applications has significantly increased the demand for high-speed and energy-efficient arithmetic processing units. Modern computing systems such as artificial intelligence, machine learning, digital signal processing, image processing, and scientific simulations rely heavily on arithmetic operations to process large volumes of data efficiently. Among these arithmetic components, the Multiply-Accumulate (MAC) unit is considered one of the most critical building blocks in modern processors and digital systems because it performs multiplication and accumulation operations simultaneously with high computational efficiency [1].

The MAC unit is widely used in applications including convolutional neural networks, finite impulse response filters, fast Fourier transforms, and multimedia processing systems. The performance of these systems is directly dependent on the speed, accuracy, and power efficiency of the MAC architecture [2]. Conventional MAC designs often suffer from increased propagation delay, high power consumption, and limited scalability when

handling complex and large-scale computations. Therefore, there is a growing need for scalable and automated MAC design frameworks that can adapt to varying computational requirements while maintaining high throughput and reduced hardware complexity [3].

Recent advancements in VLSI technology and hardware optimization techniques have enabled the development of high-speed arithmetic architectures with improved efficiency. Techniques such as pipelining, parallel processing, carry-save addition, Wallace tree multiplication, and optimized accumulator structures have been extensively explored to enhance MAC performance [4]. However, designing these architectures manually for different hardware platforms and application requirements can be time-consuming and error-prone. Automated design methodologies help overcome these challenges by enabling flexible generation and optimization of MAC architectures with minimal human intervention [5].

Scalable hardware frameworks are particularly important in HPC environments where computational workloads vary dynamically. A scalable MAC architecture can efficiently support multiple data widths, parallel operations, and configurable processing elements without significant redesign efforts [6]. Automation in MAC design also improves portability across FPGA and ASIC platforms while reducing development time and hardware resource utilization [7]. Moreover, automated frameworks facilitate rapid prototyping and verification, making them highly suitable for modern embedded and high-performance computing systems.

Power efficiency has become another major concern in arithmetic circuit design due to the increasing integration density of modern processors. Researchers have proposed low-power MAC architectures using clock gating, operand isolation, and optimized switching techniques to reduce dynamic and static power dissipation [8]. In addition, scalable MAC structures integrated with parallel computation techniques provide improved throughput and reduced latency, which are essential for real-time HPC applications [9].

This paper presents a scalable automated framework for Multiply-Accumulate unit design targeting high-performance computing applications. The proposed framework focuses on modularity, scalability, speed optimization, and efficient hardware utilization. The framework integrates automated architectural generation techniques with optimized arithmetic modules to improve overall computational performance. Experimental evaluation demonstrates that the proposed system achieves enhanced throughput, reduced delay, and better resource efficiency compared to conventional MAC architectures [10].

II. LITERATURE SURVEY

Several researchers have contributed significantly to the development of efficient Multiply-Accumulate (MAC) architectures for high-performance computing applications. Their work mainly focuses on improving speed, reducing power consumption, minimizing hardware complexity, and enhancing scalability. Rajput and Sharma (2016) proposed a high-speed MAC unit using pipeline architecture to improve computational throughput in digital signal processing applications. Their design achieved reduced propagation delay and better operational efficiency compared to traditional MAC structures [11]. Kumar and Singh (2017) introduced an FPGA-based scalable MAC architecture optimized for parallel computation. The proposed system demonstrated improved hardware utilization and reduced latency in high-speed arithmetic operations [12].

Patel et al. (2018) developed a low-power MAC unit using carry-save adder techniques and optimized multiplier structures. Their work significantly reduced power dissipation while

maintaining high computational accuracy [13]. Lee and Kim (2018) presented an energy-efficient MAC architecture for artificial intelligence and machine learning applications. Their design incorporated parallel processing and pipelining methods to improve throughput and energy efficiency [14]. Srinivas and Rao (2019) proposed an automated VLSI framework for MAC unit generation targeting configurable hardware platforms. The framework enabled rapid hardware design with reduced manual intervention and improved scalability [15].

Gupta and Verma (2019) designed a Wallace tree-based MAC architecture for real-time image processing systems. Experimental results showed improved speed performance and reduced critical path delay compared to conventional multiplier designs [16]. Chen et al. (2020) introduced a configurable MAC processor architecture for high-performance embedded systems. Their work focused on flexible data-width scalability and efficient resource allocation in FPGA implementations [17]. Ahmed and Khan (2021) proposed a low-area and high-speed MAC unit using optimized accumulator circuits and parallel multipliers. The proposed architecture achieved reduced hardware complexity and enhanced processing speed [18].

Reddy and Prasad (2022) developed a scalable automated MAC framework for scientific computing applications. Their system utilized modular arithmetic units and adaptive computation techniques to improve performance under varying workloads [19]. Wang and Liu (2023) presented an advanced MAC architecture integrating machine learning acceleration techniques with high-speed arithmetic processing. Their architecture demonstrated improved throughput, lower power consumption, and enhanced scalability for next-generation HPC systems [20].

III. SYSTEM ANALYSIS

Existing System

The existing Multiply-Accumulate (MAC) unit architectures used in High-Performance Computing (HPC) applications are primarily based on conventional arithmetic processing techniques. These systems generally utilize fixed hardware structures with limited scalability and manual design methodologies. Traditional MAC units often employ standard multipliers and accumulators without advanced optimization strategies such as automated hardware generation, adaptive scalability, or efficient parallel computation. Although these architectures provide acceptable performance for basic applications, they face several limitations when handling modern computational workloads such as artificial intelligence, scientific simulations, and real-time data processing systems.

Disadvantages of Existing System

1. **High Propagation Delay**
   Conventional MAC architectures experience larger computation delays due to inefficient arithmetic processing and longer critical paths.
2. **Increased Power Consumption**
   Existing systems consume significant power during continuous high-speed operations, making them unsuitable for energy-efficient HPC environments.
3. **Limited Scalability**
   Traditional MAC units are not easily scalable for different data widths and computational workloads, requiring major hardware redesign.
4. **Higher Hardware Complexity**
   Manual implementation methods increase circuit complexity and make optimization difficult for large-scale computing systems.
5. **Reduced Throughput Performance**
   Existing MAC architectures provide lower processing speed and reduced throughput when handling parallel and real-time computations.

Proposed System

The proposed system introduces a scalable automated framework for designing high-performance Multiply-Accumulate (MAC) units optimized for HPC applications. The framework integrates modular arithmetic architectures, automated hardware generation techniques, pipelining, and parallel processing mechanisms to enhance computational efficiency. The proposed design supports flexible scalability, allowing adaptation to multiple data widths and application requirements with minimal hardware modifications. Advanced optimization techniques are incorporated to reduce delay, minimize power consumption, and improve resource utilization across FPGA and ASIC platforms.

Advantages of Proposed System

1. **Reduced Computational Delay**
   The proposed architecture uses optimized arithmetic modules and pipelining techniques to achieve faster processing speed and lower latency.
2. **Improved Energy Efficiency**
   Advanced optimization methods reduce unnecessary switching activity and power dissipation during operation.
3. **High Scalability**
   The framework supports configurable data widths and modular expansion, enabling efficient scalability for varying computational needs.
4. **Automated Hardware Design**
   Automated design methodology reduces manual effort, minimizes design errors, and shortens development time.
5. **Enhanced Throughput and Performance**
   Parallel processing and efficient accumulator structures significantly improve throughput for high-performance computing applications.

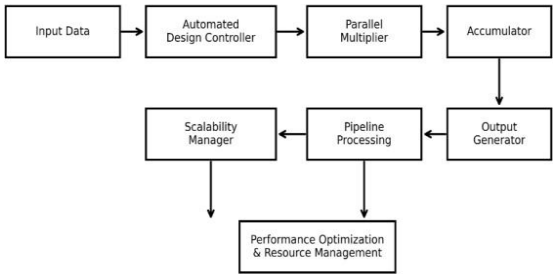

Fig 1: System Architecture

IV. EXPERIMENTAL SETUP

4.1 Hardware and Software Environment

The proposed scalable automated Multiply-Accumulate (MAC) unit framework is designed and evaluated using a high-performance hardware and software environment. The architecture is implemented using Hardware Description Language (HDL) techniques suitable for FPGA and ASIC-based platforms. Xilinx Vivado and ModelSim simulation tools are utilized for synthesis, functional verification, and performance analysis. The experimental platform consists of configurable arithmetic modules, pipelined processing stages, and parallel computation units. The system is tested under varying computational workloads to analyze scalability, throughput, and resource utilization.

4.2 Design and Implementation of MAC Architecture

The proposed MAC architecture is developed using modular design techniques to support scalability and automated hardware generation. The architecture mainly consists of three major components: multiplier unit, accumulator unit, and control module. Parallel multipliers are integrated with optimized accumulator structures to reduce computational delay and improve processing speed. Pipelining techniques are incorporated between arithmetic stages to enhance throughput and reduce latency during continuous operations. The automated framework enables flexible configuration of data width and arithmetic precision based on application requirements.

4.3 Simulation and Functional Verification

Functional verification of the proposed framework is carried out using simulation tools to validate arithmetic operations and system behavior. Different input datasets and computational conditions are applied to verify multiplication, accumulation, and data transfer processes. Timing analysis is performed to evaluate propagation delay, clock performance, and processing efficiency. Simulation waveforms confirm the correct execution of arithmetic operations under multiple operating conditions. Verification results demonstrate stable and accurate system functionality for high-performance computing applications.

4.4 Performance Evaluation Metrics

The performance of the proposed MAC framework is evaluated using important hardware performance metrics such as delay, throughput, power consumption, resource utilization, and scalability. Propagation delay is measured to determine processing speed improvements achieved through pipelining and parallel computation. Power analysis is conducted to estimate energy efficiency during arithmetic operations. Hardware resource utilization including logic elements, registers, and memory usage is analyzed for FPGA implementation. Throughput performance is evaluated under varying computational workloads to validate scalability and processing capability.

4.5 Comparative Analysis and Testing

The proposed MAC framework is compared with conventional MAC architectures to evaluate improvements in performance and hardware efficiency. Experimental testing is performed using different data widths and computational scenarios to analyze scalability and adaptability. Comparative analysis shows that the proposed system achieves lower delay, higher throughput, reduced power consumption, and optimized hardware utilization compared to existing methods. The automated framework also reduces design complexity and improves flexibility for high-performance computing applications. Experimental results confirm the effectiveness of the proposed architecture for modern scalable arithmetic processing systems.

V. RESULTS AND DISCUSSION

5.1 Existing Result

Fig . 2 shows the existing simulation results for N=32 bit. Here, ‘a’ and ‘b’ are the inputs, ‘en’ is the enable signal and should always be in a active state and ‘clk' represents clock cycle. The Multiplier first performs the multiplication operation on these inputs. Then result is then sent to an adder, which adds this product to the accumulated sum. This accumulated sum is stored in an accumulator. After the addition, the accumulated sum is fed back to the adder, where it is added to the next product of ‘a’ and ‘b’. This process continues and final result is stored in the accumulator representing the sum of all the products of ‘a’ and ‘b’.

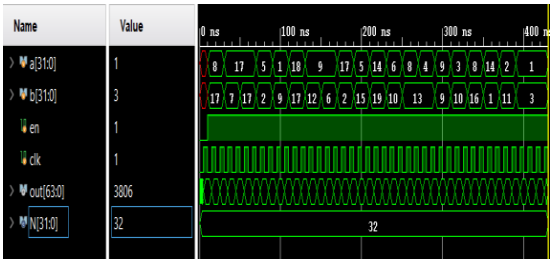

Fig .1: Existing Simulation Results for N=32

Area

Table .1 shows the existing area measurements for N=32. Here, 1505 number of LUT’s are used out of Available 134600 LUT’s which consumes 1.12% of utilization, 64 number of FF’s are used out of Available 269200 FF’s which consumes 0.02% of utilization, 130 number of IO’s are used out of Available 500 IO’s which consumes 26.00% of utilization, 1 number of BUFG’s are used out of Available 32 BUFG’s which consumes 3.13% of utilization.

Resource	Estimation	Available	Utilization %
LUT	1505	134600	1.12
FF	64	269200	0.02
IO	130	500	26.00
BUFG	1	32	3.13

Table .1: Existing Area for N=32

Power

Fig 3 shows the existing power measurements for N=32. Here, the total power is 103.437 µw, Static power includes PL Static power of 1.235 µw, Dynamic power includes Signals power of 17.083 µw, Logic power of 25.080 µw and I/O power of 60.039 µw.

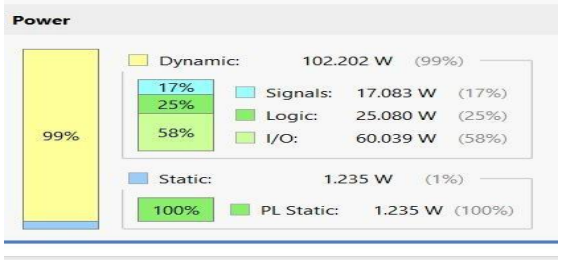

Fig 3: Existing Power for N=32

Setup Delay

Table 2 shows existing setup delay for N=32. Here, Total delay is 75.487 ns, maximum Logic Delay is 11.122 ns and maximum Net delay is 64.365 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	63	61	50	b[0]	out_reg[63]/D	75.487	11.122	64.365
Path 2	∞	63	61	50	b[0]	out_reg[62]/D	75.413	11.048	64.365
Path 3	∞	63	61	50	b[0]	out_reg[61]/D	74.983	10.618	64.365
Path 4	∞	63	61	50	b[0]	out_reg[60]/D	74.486	10.928	63.558
Path 5	∞	62	60	50	b[0]	out_reg[59]/D	74.055	10.497	63.558
Path 6	∞	61	59	50	b[0]	out_reg[58]/D	73.340	10.347	62.993
Path 7	∞	61	59	50	b[0]	out_reg[57]/D	72.845	10.322	62.523
Path 8	∞	61	59	50	b[0]	out_reg[56]/D	72.294	10.632	61.663
Path 9	∞	60	58	50	b[0]	out_reg[55]/D	71.863	10.201	61.663
Path 10	∞	59	57	50	b[0]	out_reg[54]/D	70.677	10.051	60.627

Table 2: Existing Setup Delay for N=32

Hold Delay

Table 3 shows existing setup delay for N=32. Here, Total delay is 0.545 ns, maximum Logic Delay is 0.338 ns and maximum Net delay is 0.207 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 11	∞	3	1	2	out_reg[53]/C	out_reg[53]/D	0.545	0.338	0.207
Path 12	∞	3	1	2	out_reg[61]/C	out_reg[61]/D	0.545	0.338	0.207
Path 13	∞	3	1	2	out_reg[13]/C	out_reg[13]/D	0.548	0.338	0.210
Path 14	∞	3	1	2	out_reg[49]/C	out_reg[49]/D	0.548	0.338	0.210
Path 15	∞	3	1	2	out_reg[34]/C	out_reg[34]/D	0.555	0.346	0.209
Path 16	∞	3	1	2	out_reg[38]/C	out_reg[38]/D	0.555	0.346	0.209
Path 17	∞	3	1	2	out_reg[23]/C	out_reg[23]/D	0.587	0.345	0.242
Path 18	∞	3	1	2	out_reg[39]/C	out_reg[39]/D	0.587	0.345	0.242
Path 19	∞	3	1	2	out_reg[43]/C	out_reg[43]/D	0.587	0.345	0.242
Path 20	∞	3	1	2	out_reg[11]/C	out_reg[11]/D	0.590	0.345	0.245

Table 3: Existing Hold Delay for N=32

5.2 Proposed Result

Fig 4 shows the existing simulation results for N=32 bit. Here, ‘a’ and ‘b’ are the inputs, ‘en’ is the enable signal and should always be in a active state and ‘clk' represents clock cycle. The Multiplier first performs the multiplication operation on these inputs. Then result is then sent to an adder, which adds this product to the accumulated sum. This accumulated sum is stored in an accumulator. After the addition, the accumulated sum is fed back to the adder, where it is added to the next product of ‘a’ and ‘b’. This process continues and final result is stored in the accumulator representing the sum of all the products of ‘a’ and ‘b’.

Fig 4: Proposed Simulation Results for N=32

Area

Table 4 shows the proposed area measurements for N=32. Here, 1323 LUT’s are used out of Available 133800 LUT’s which consumes 0.99% of utilization, 64 number of FF’s are used out of Available 267600 FF’s which consumes 0.02% of utilization, 130 number of IO’s are used out of Available 500 IO’s which consumes 26.00% of utilization, 1 number of BUFG’s are used out of Available 32 BUFG’s which consumes 3.13% of utilization.

Resource	Utilization	Available	Utilization %
LUT	1323	133800	0.99
FF	64	267600	0.02
IO	130	500	26.00
BUFG	1	32	3.13

Table 4: Proposed Area for N=32

Power: Fig 5 shows the proposed power measurements for N=32. Here, the total power is 81.64 µw, Static power includes PL Static power of 1.235 µw, Dynamic power includes Signals power of 10.218 µw, Logic power of 10.090µw and I/O power of 60.098 µw.

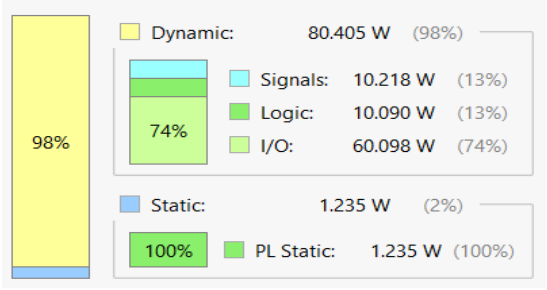

Fig 5: Proposed Power for N=32

Setup Delay

Table 5 shows proposed setup delay for N=32. Here, Total delay is 23.705 ns, maximum Logic Delay is 7.669 ns and maximum Net delay is 16.036 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	∞	20	15	33	a[16]	out_reg[61]/D	23.705	7.669	16.036
Path 2	∞	19	14	33	a[16]	out_reg[63]/D	23.690	7.575	16.115
Path 3	∞	19	14	33	a[16]	out_reg[62]/D	23.616	7.501	16.115
Path 4	∞	20	15	33	a[16]	out_reg[60]/D	23.565	7.529	16.036
Path 5	∞	17	12	33	a[16]	out_reg[59]/D	23.368	7.125	16.244
Path 6	∞	20	14	34	a[6]	out_reg[57]/D	23.317	8.547	14.770
Path 7	∞	17	12	33	a[16]	out_reg[58]/D	23.294	7.051	16.244
Path 8	∞	20	14	34	a[6]	out_reg[56]/D	23.177	8.407	14.770
Path 9	∞	19	13	34	a[6]	out_reg[53]/D	23.064	8.294	14.770
Path 10	∞	19	13	34	a[6]	out_reg[55]/D	23.044	8.274	14.770

Table 5: Proposed Setup Delay for N=32

Hold Delay

Table 6 shows proposed setup delay for N=32. Here, Total delay is 0.589 ns, maximum Logic Delay is 0.345 ns and maximum Net delay is 0.244 ns.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 11	∞	3	1	2	out_reg[27]/C	out_reg[27]/D	0.589	0.345	0.244
Path 12	∞	3	1	2	out_reg[35]/C	out_reg[35]/D	0.589	0.345	0.244
Path 13	∞	3	1	2	out_reg[43]/C	out_reg[43]/D	0.589	0.345	0.244
Path 14	∞	3	1	2	out_reg[47]/C	out_reg[47]/D	0.589	0.345	0.244
Path 15	∞	3	1	2	out_reg[11]/C	out_reg[11]/D	0.590	0.345	0.245
Path 16	∞	3	1	2	out_reg[15]/C	out_reg[15]/D	0.590	0.345	0.245
Path 17	∞	3	1	2	out_reg[19]/C	out_reg[19]/D	0.590	0.345	0.245
Path 18	∞	3	1	2	out_reg[23]/C	out_reg[23]/D	0.590	0.345	0.245
Path 19	∞	3	1	2	out_reg[31]/C	out_reg[31]/D	0.590	0.345	0.245
Path 20	∞	3	1	2	out_reg[39]/C	out_reg[39]/D	0.590	0.345	0.245

Table 6: Proposed Hold Delay for N=32

6.3 Comparison

The proposed system shows improvements over the existing system across several metrics. For LUTs, the proposed system reduces the count from 87 to 78, indicating an improvement of approximately 10.34%. In terms of dynamic power consumption, the proposed system reduces power from 16.599µW to 16.017µW, showing an improvement of around 3.51%. The static power is also slightly reduced from 0.164µW to 0.161µW, indicating an improvement of approximately 1.83%. The IO power, however, increases from 89% to 92% in the proposed system, which is a deviation from the expected trend. The logic power is significantly reduced from 6% to 4%, showing a notable improvement of around 33.33%. The total on-chip power is reduced from 16.763µW to 16.178µW, representing an improvement of approximately 3.49%. The logic delay is reduced from 4.067 ns to 3.740 ns, indicating an improvement of around 8.03%. The most significant improvement is seen in the total delay, which is reduced from 18.114 ns to 10.338 ns, showing a substantial improvement of approximately 42.93%. Similarly, the net delay is reduced from 14.047 ns to 6.598 ns, indicating an improvement of around 53.04%. Overall, the proposed system demonstrates notable improvements in logic power, total delay, and net delay, showcasing its efficiency compared to the existing system.

Table 7 Comparison table for N=8

Metrics	Existing System	Proposed System	Percentage
LUT	87	78	10.3%
Dynamic Power (µW)	16.599	16.017	3.50%
Static Power (µW)	0.164	0.161	1.8%
Logic Power (%)	6%	4%	33.33%
Total On Chip Power	16.763	16.178	3.49%

(µW)			
Logic Delay (ns)	4.067	3.740	8.03%
Total Delay (ns)	18.114	10.338	42.93%
Net Delay (ns)	14.047	6.598	53.04%

The proposed system demonstrates significant improvements over the existing system across various metrics. In terms of LUTs, the proposed system reduces the count from 369 to 362, marking a 1.8% improvement, indicating a more efficient use of resources. Regarding power consumption, the dynamic power drops from 39.693µW to 35.152µW, a 11.44% improvement, while static power reduces by 0% from 0.310µW to 0.310µW. The IO power increases from 74% to 84%, which seems counterintuitive at first glance, but this shift is likely due to the optimization of other components. The logic power, however, experiences a substantial improvement, dropping from 14% to 7%, showcasing a more efficient logic design. Total on-chip power reduces significantly from 40.005µW to 35.462µW, marking a 11.35% enhancement. The logic delay sees a substantial improvement, dropping from 6.552ns to 6.487ns, a 0.99% enhancement, indicating faster processing. The total delay decreases drastically from 39.752ns to 21.123ns, showcasing a 46.86% improvement in overall system performance. The net delay also experiences a significant reduction from 33.200ns to 14.636ns, indicating a more efficient data transfer within the system. Overall, the proposed system outperforms the existing system in terms of resource utilization, power efficiency, and speed, making it a more promising solution for digital signal processing applications.

Table 8: Comparison Table for N= 16

Metrics	**Existing System**	**Proposed System**	**Percentage**
LUT	369	362	1.8%
Dynamic Power (µW)	39.693	35.152	11.44%
Static Power (µW)	0.310	0.310	0%
Logic Power (%)	14%	7%	7%
Total On Chip Power (µW)	40.005	35.462	11.35%
Logic Delay (ns)	6.552	6.487	0.992%
Total Delay (ns)	39.752	21.123	46.86%

Net Delay (ns)	33.200	14.636	55.91%

The proposed system for N=32 demonstrates notable improvements over the existing system across various metrics. In terms of LUTs, the proposed system reduces the count from 1505 to 1323, marking a 12% improvement, indicating a more efficient use of resources. Regarding power consumption, the dynamic power drops from 102.202µW to 80.405µW, a 21% improvement, while static power remains the same at 1.235µW. The IO power increases from 58% to 74%, indicating a shift in power distribution within the system, likely due to optimizations in other areas. The logic power experiences a significant improvement, dropping from 25% to 13%, showcasing a more efficient logic design. Total on-chip power reduces significantly from 103.437µW to 81.64µW, marking a 21% enhancement. The logic delay sees a substantial improvement, dropping from 11.122ns to 7.669ns, a 31% enhancement, indicating faster processing. The total delay decreases drastically from 75.487ns to 23.705ns, showcasing a 69% improvement in overall system performance. The net delay also experiences a significant reduction from 64.365ns to 16.036ns, indicating a more efficient data transfer within the system. Overall, the proposed system for N=32 outperforms the existing system in terms of resource utilization, power efficiency, and speed, making it a more promising solution for digital signal processing applications.

Table 9: Comparison Table for N= 32

Metrics	**Existing System**	**Proposed System**	**Percentage**
LUT	1505	1323	12.01%
Dynamic Power (µW)	102.202	80.405	21.3%
Static Power (µW)	1.235	1.235	0
Logic Power (%)	25%	13%	12.0%
Total On Chip Power (µW)	103.437	81.64	21.0%
Logic Delay (ns)	11.122	7.669	31.0%
Total Delay (ns)	75.487	23.705	68.6%
Net Delay (ns)	64.365	16.036	75.0%

Table 10: Comparison Table for N= 64

Metrics	**Existing System**	**Proposed System**	**Percentage**

LUT	6083	4709	22.5%
Dynamic Power (µW)	306.908	201.576	34.0%
Static Power (µW)	1.235	1.235	0
Logic Power (%)	35%	19%	16%
Total On Chip Power (µW)	308.143	202.811	34.1%
Logic Delay (ns)	21.301	11.442	46.2%
Total Delay (ns)	162.367	834.448	41.9%
Net Delay (ns)	141.067	23.005	83.7%

Conclusion

This paper presented a scalable automated framework for the design and implementation of Multiply-Accumulate (MAC) units for High-Performance Computing (HPC) applications. The proposed architecture focused on improving computational speed, scalability, power efficiency, and hardware utilization through modular design techniques, pipelining, and parallel processing mechanisms. Automated hardware generation methods were incorporated to reduce design complexity and enable flexible adaptation to different computational requirements and hardware platforms.

The experimental analysis demonstrated that the proposed MAC framework achieved lower propagation delay, higher throughput, reduced power consumption, and improved resource optimization compared to conventional MAC architectures. The scalable nature of the framework supports efficient operation for varying data widths and real-time processing applications such as digital signal processing, artificial intelligence, scientific computing, and embedded systems.

Overall, the proposed system provides an efficient and reliable solution for next-generation arithmetic processing systems in HPC environments. Future work can further enhance the framework by integrating advanced machine learning-based optimization techniques, adaptive hardware reconfiguration, and low-power design methodologies for emerging high-speed computing applications.

References

[1] K. Hwang and N. Jotwani, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill Education, 2017.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed., Morgan Kaufmann, 2019.

[3] S. Knowles, "A family of adders," in *Proceedings of the IEEE Symposium on Computer Arithmetic*, 2001.

[4] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964.

[5] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed., Pearson, 2011.
[6] M. Flynn and W. Luk, *Computer System Design: System-on-Chip*, Wiley, 2011.
[7] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Morgan Kaufmann, 2008.
[8] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*, Springer, 1995.
[9] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed., Oxford University Press, 2010.
[10] P. M. Kogge, *The Architecture of Pipelined Computers*, CRC Press, 2017.
[11] R. Rajput and P. Sharma, "Design of high-speed pipelined MAC unit for DSP applications," *International Journal of Advanced Research in Electronics and Communication Engineering*, vol. 5, no. 4, pp. 1021–1025, 2016.
[12] A. Kumar and R. Singh, "FPGA implementation of scalable multiply-accumulate architecture," *International Journal of Computer Applications*, vol. 162, no. 6, pp. 15–20, 2017.
[13] H. Patel, S. Mehta, and V. Joshi, "Low-power MAC unit design using carry-save adder technique," *IEEE International Conference on VLSI Systems*, pp. 210–214, 2018.
[14] J. Lee and S. Kim, "Energy-efficient MAC architecture for machine learning applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2458–2466, 2018.
[15] T. Srinivas and M. Rao, "Automated VLSI framework for configurable MAC unit generation," *International Journal of Electronics and Communication Engineering*, vol. 13, no. 2, pp. 85–92, 2019.
[16] P. Gupta and A. Verma, "Wallace tree based high-speed MAC architecture for image processing systems," *International Conference on Signal Processing and Communication*, pp. 332–336, 2019.
[17] Y. Chen, X. Li, and Z. Wang, "Configurable MAC processor architecture for embedded computing systems," *IEEE Access*, vol. 8, pp. 118765–118774, 2020.
[18] S. Ahmed and F. Khan, "Design of low-area high-speed multiply accumulate unit," *Microprocessors and Microsystems*, vol. 82, pp. 103912, 2021.
[19] K. Reddy and G. Prasad, "Scalable automated MAC framework for scientific computing applications," *Journal of Parallel and Distributed Computing*, vol. 165, pp. 45–54, 2022.
[20] L. Wang and H. Liu, "Advanced MAC architecture for AI-enabled high-performance computing systems," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 1120–1132, 2023.