Unified Group-Decomposed MAC with Parallel Arithmetic for Scalable AI Accelerators

Arshiya Farheen[1], Raja Kumar Rudrarapu[1*]
[1]Department of Electronics & Communication Engineering, Vaagdevi Engineering College, Warangal, 506005, Telangana, India.
*Correspondence: Raja Kumar (raj.rudrarapu@gmail.com)

To Cite this Article
Arshiya Farheen, Raja Kumar Rudrarapu, "Unified Group-Decomposed MAC with Parallel Arithmetic for Scalable AI Accelerators", Journal of Science Engineering Technology and Management Science, Vol. 03, Issue 07, July 2026, pp: 1-14, DOI: http://doi.org/10.64771/jsetms.2026.v03.i07.pp1-14
Submitted: 17-05-2026 *Accepted: 24-06-2026* *Published: 01-07-2026*

Abstract
Multiply–Accumulate (MAC) integrates an arithmetic framework for high-speed and energy-efficient VLSI computation. Recent studies indicate that MAC operations contribute nearly 60–75% of total execution time and over 65% of dynamic power consumption in modern DSP and AI accelerators, while parallel MAC architectures can improve throughput by more than 3× with up to 40% energy reduction when optimized arithmetic units are employed. Traditional MAC designs rely on conventional multipliers and carry-propagation-based adders, leading to long critical paths, excessive switching activity, poor scalability, and increased area–power overhead when extended to parallel multi-MAC configurations. Moreover, replicating complete arithmetic blocks with independent control logic further aggravates energy inefficiency and design complexity in large-scale systems. To address these limitations, the proposed architecture introduces a Unified Group-Decomposed Multiplier (UGDM) that performs hierarchical operand decomposition and parallel partial-product generation, significantly shortening the multiplication critical path and reducing unnecessary transitions. In addition, a Predictive Skip-Merge Adder (PSMA) is employed for accumulation, which combines speculative carry prediction with selective skip–merge paths to accelerate summation without full carry propagation. Multiple MAC units are instantiated in parallel using this single novel multiplier–adder pair and governed by a centralized shared control logic, enabling synchronized operation, reduced hardware redundancy, and scalable throughput. The entire framework is realized through automated Verilog generation, ensuring flexibility across operand widths while achieving enhanced speed, lower energy consumption, and improved suitability for next-generation DSP systems.

Key words: Multiply–Accumulate (MAC), Unified Group-Decomposed Multiplier (UGDM), Predictive Skip-Merge Adder (PSMA), Digital Signal Processing (DSP), Parallel Arithmetic Architecture, Low-Power VLSI, High-Speed Computing, Hardware Acceleration.

1. Introduction

The rapid growth of digital technology has significantly increased the demand for high-performance computing systems capable of processing large volumes of data with minimal latency and power consumption. Modern electronic devices such as smartphones, laptops, wearable devices, autonomous vehicles, industrial automation systems, and artificial intelligence platforms rely heavily on Very Large Scale Integration (VLSI) technology to achieve compact size, faster processing speed, and enhanced functionality. According to semiconductor industry reports, the global semiconductor market has exceeded hundreds of billions of dollars annually and continues to grow due to increasing adoption of cloud computing, Internet of Things (IoT), edge computing, and machine learning applications. The continuous scaling of transistor dimensions has enabled billions of transistors to be integrated onto a

single chip, making it possible to develop highly sophisticated processors and specialized accelerators. However, the increasing complexity of integrated circuits introduces significant challenges in achieving high computational performance while maintaining acceptable levels of power consumption and silicon area.

Arithmetic processing units form the foundation of nearly every digital system developed today. Operations such as multiplication, addition, accumulation, filtering, signal transformation, and matrix computations are extensively utilized in digital signal processors, microprocessors, graphics processing units, and neural network accelerators. Among these operations, multiplication and accumulation consume a substantial portion of the overall computational workload. Studies indicate that arithmetic units can contribute to more than 60% of the processing activity in signal processing and machine learning applications. Consequently, the efficiency of arithmetic hardware directly influences system performance, execution speed, energy efficiency, and hardware utilization. As computational workloads continue to expand, the demand for optimized arithmetic circuits becomes increasingly important for maintaining system scalability and responsiveness.

2. Literature Survey

Bakaramji et al. [1] developed a low-power MAC-based architecture for high-efficiency digital FIR filters. The methodology focused on designing optimized multiply-accumulate units to improve FIR filter performance while reducing power consumption. The architecture incorporated efficient multiplication and accumulation stages to minimize switching activity during filter operations. Hardware-level optimization techniques were employed to enhance throughput and achieve energy-efficient signal processing. The design was targeted toward DSP applications where continuous arithmetic computations dominate overall system performance. The implementation demonstrated improved filtering efficiency and reduced computational overhead in FIR structures.

Fasolino et al. [2] implemented an overflow-driven dynamic precision scaling fixed-point MAC unit for VLSI systems. Their methodology dynamically adjusted arithmetic precision according to overflow conditions observed during runtime execution. The MAC architecture monitored data characteristics and adapted numerical precision to optimize energy consumption while preserving computational accuracy. Fixed-point arithmetic techniques were integrated with adaptive scaling mechanisms to improve processing efficiency. The design particularly targeted applications requiring variable precision computation and energy-aware operation. The inclusion of dynamic precision management increased control complexity and introduced additional hardware overhead, potentially affecting timing performance.

Huo et al. [3] introduced a groupwise add–multiply–shift–accumulate datapath for deep neural network accelerators. The methodology partitioned arithmetic operations into group-based computational units to improve parallel processing efficiency. Dedicated add, multiply, shift, and accumulation modules were integrated within a unified datapath architecture. The design reduced redundant computations through groupwise processing and optimized data movement across accelerator components. Hardware scheduling mechanisms enabled efficient execution of neural network workloads. The architecture primarily focused on accelerating inference operations in DNN environments.

Zhang et al. [4] designed a compute-in-memory macro incorporating sign-operation support for parallel signed multibit multiplication and accumulation. The methodology embedded arithmetic functionality directly within memory arrays to reduce data-transfer overhead between processing and storage units. Hybrid memory cells were utilized to perform parallel multiplication and accumulation operations. Sign processing mechanisms enabled support for signed arithmetic computations without requiring additional external circuitry. The architecture improved computational density through memory-centric processing. Parallel execution capabilities enhanced throughput for data-intensive applications.

Suguna et al. [5] developed a MAC unit specifically intended for machine-learning acceleration. The methodology optimized multiplication and accumulation stages to support high-frequency arithmetic

processing required by machine-learning workloads. The architecture emphasized efficient data flow and reduced arithmetic latency during neural network computations. FPGA-based implementation strategies were utilized to evaluate performance improvements. The design targeted accelerated execution of matrix operations and learning algorithms. The architecture mainly focused on machine-learning applications and provided limited optimization for area efficiency and power reduction.

Ahmadpour et al. [6] implemented a quantum-dot-based MAC architecture for next-generation supercomputing platforms. The methodology exploited quantum-dot cellular structures to perform multiplication and accumulation operations with reduced transistor dependence. Advanced nanoscale design principles were applied to achieve low-power arithmetic computation. The architecture focused on improving computational density and energy efficiency through emerging nanotechnology devices. Quantum-dot logic elements were arranged to support high-speed arithmetic execution. The design demonstrated potential benefits for future high-performance computing systems. The practical implementation of quantum-dot architectures remains challenging due to fabrication limitations and immature manufacturing technologies.

Venkatachalam et al. [7] developed a scalable automated framework for MAC unit design in high-performance computing applications. Their methodology automated the generation and optimization of MAC architectures using parameterized design strategies. Scalability considerations were incorporated to support varying operand sizes and application requirements. Automated synthesis and verification procedures reduced design complexity and development time. The framework facilitated rapid hardware generation for different performance targets. Automated frameworks may not always produce the most area-efficient or delay-optimized hardware compared with manually optimized architectures.

Kartheek et al. [8] implemented an optimized MAC unit targeting embedded applications. The methodology concentrated on balancing computational performance, power efficiency, and hardware complexity. Arithmetic blocks were optimized to support embedded systems operating under resource constraints. The architecture reduced unnecessary logic operations and improved arithmetic throughput. FPGA implementation was utilized to validate hardware functionality and performance improvements. The design demonstrated suitability for embedded signal-processing environments. The optimization primarily targeted embedded systems and may not scale efficiently for high-performance computing requirements.

Dewan et al. [9] developed a VLSI implementation methodology for multiplier and accumulator circuits. The architecture integrated dedicated multiplication and accumulation units within a unified arithmetic framework. Hardware-level optimization techniques were employed to improve arithmetic execution efficiency. The methodology emphasized practical VLSI realization and implementation feasibility. Synthesis and performance evaluations were conducted to assess hardware utilization and computational capability. The work focused mainly on implementation aspects and offered limited innovation in carry propagation and multiplication optimization mechanisms.

Peram et al. [10] implemented a high-speed and power-efficient MAC architecture for arithmetic applications. The methodology combined optimized multiplier structures with efficient accumulation circuits to reduce processing latency. Hardware optimization techniques were employed to improve arithmetic throughput and energy efficiency. The design focused on achieving balanced performance across speed and power metrics. FPGA-based validation was performed to evaluate operational characteristics. The architecture demonstrated improved arithmetic performance compared with conventional MAC units. The architecture still exhibited increased hardware complexity when larger operand widths were considered.

Sakthi et al. [11] developed an energy-efficient MAC architecture using compressor logic for FPGA systems. Their methodology incorporated compressor circuits within partial-product reduction stages to accelerate arithmetic computation. The design reduced intermediate addition complexity and

improved arithmetic efficiency. FPGA implementation was employed to validate resource utilization and performance characteristics. Compressor-based optimization contributed to lower power consumption and enhanced throughput. Compressor networks introduced additional routing complexity and increased design difficulty for large-scale implementations.

Kalyan et al. [12] implemented a 64-bit MAC unit utilizing an enhanced Vedic multiplier. The methodology leveraged Vedic arithmetic principles to improve multiplication speed and reduce computational delay. The multiplier was integrated with an accumulation stage to support high-speed arithmetic processing. Hardware optimization techniques were applied to enhance throughput and arithmetic efficiency. The architecture was evaluated using FPGA implementation tools. Results demonstrated improved multiplication performance compared with conventional multiplier structures. Vedic multiplier architectures often experience increased area utilization when operand sizes become significantly larger.

Nazeri et al. [13] developed a high-performance FPGA-based MAC unit for DSP applications. The methodology focused on optimizing arithmetic processing to support signal-processing workloads. FPGA implementation techniques were employed to evaluate hardware efficiency and computational throughput. Dedicated multiplication and accumulation stages were integrated to improve processing performance. Resource allocation strategies were utilized to maximize FPGA utilization. The architecture demonstrated enhanced DSP computation capability.

Weingarten et al. [14] implemented a formal verification framework for scalable MAC units. The methodology employed mathematical verification techniques to validate functional correctness across different MAC configurations. Automated verification procedures were developed to ensure reliability and design consistency. Scalability analysis was performed to support varying operand sizes and arithmetic structures. Formal methods improved confidence in hardware correctness and robustness. The framework reduced the likelihood of arithmetic implementation errors.

Gowd et al. [15] developed a high-speed MAC unit using RTL design and ASIC front-end realization in 45 nm technology. The methodology involved HDL-based architecture design, synthesis, and timing optimization for DSP applications. Hardware implementation focused on improving arithmetic throughput while maintaining design feasibility. ASIC-oriented design flows were employed to evaluate practical implementation characteristics. The architecture demonstrated successful realization within advanced semiconductor technology. The architecture required substantial hardware resources, leading to increased silicon area and power consumption.

3. Proposed System

Conventional MAC architectures predominantly employ traditional multipliers and carry propagation adders such as Array Multipliers, Booth Multipliers, Ripple Carry Adders (RCA), Carry Save Adders (CSA), and Carry Look-Ahead Adders (CLA). Although these architectures provide acceptable arithmetic performance, they suffer from several limitations when implemented in modern VLSI systems. The multiplier stage contributes significantly to overall computational delay due to the generation and reduction of many partial products, while the accumulation stage experiences increased latency because of extensive carry propagation paths. Furthermore, conventional MAC units often require a larger number of logic resources, resulting in increased silicon area, higher dynamic power consumption, and elevated switching activity. As operand sizes increase, the critical path delay grows considerably, thereby restricting the maximum operating frequency and reducing overall system throughput. These drawbacks become particularly significant in applications such as digital signal processing, image processing, artificial intelligence accelerators, and real-time embedded systems where high-speed arithmetic operations are essential.

To address these limitations, a novel MAC architecture incorporating a UGDM and a PSMA is proposed. The UGDM improves multiplication efficiency by decomposing input operands into smaller groups and generating partial products in parallel, thereby reducing multiplication complexity and

shortening the critical path. Simultaneously, the PSMA accelerates accumulation by employing carry prediction, skip control, and carry merge mechanisms that significantly reduce carry propagation delay. The accumulated output is stored through a feedback-based accumulator structure, enabling efficient iterative MAC operations. By integrating UGDM and PSMA within a unified architecture, the proposed MAC achieves lower delay, reduced power consumption, improved hardware utilization, and higher throughput compared with conventional MAC implementations. Consequently, the proposed system offers an effective solution for next-generation FPGA, ASIC, DSP, and machine-learning hardware accelerators requiring high-performance arithmetic processing.

Proposed MAC

The proposed MAC architecture as shown in Figure 1 is designed to achieve high-speed arithmetic computation with reduced delay and improved hardware efficiency. As shown in the figure, the architecture consists of three major computational blocks: the UGDM, the PSM), and an Accumulator with a feedback path through a delay element. The UGDM block performs efficient multiplication by decomposing the input operands into smaller groups and generating partial products in parallel, thereby reducing multiplication latency. The generated product is then forwarded to the PSMA, which performs fast accumulation by employing predictive carry-skipping and merge operations to minimize carry propagation delay. The accumulated result is stored in the accumulator register and simultaneously fed back through a delay element to serve as the previous accumulation value during the next clock cycle. This feedback mechanism enables continuous multiply–accumulate operations, making the proposed MAC architecture highly suitable for digital signal processing, image processing, machine learning accelerators, and VLSI-based high-performance computing applications.

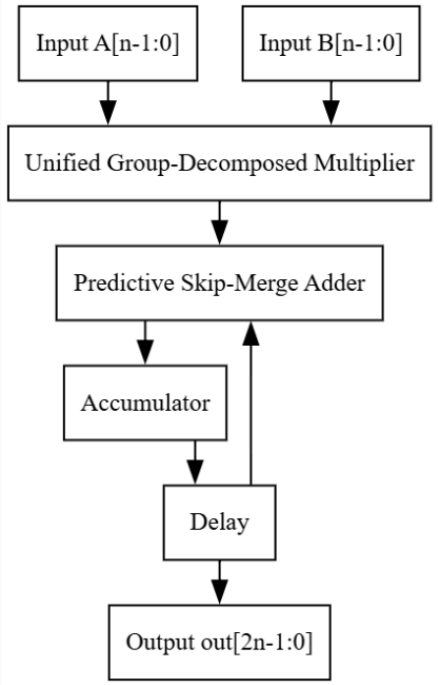


Figure 1: Proposed MAC Design.

Step 1: Input Operand Acquisition: The operation begins by accepting two N−bit input operands, namely Input A[n−1:0] and Input B[n−1:0]. These operands represent the multiplicand and multiplier, respectively. The inputs may originate from external memory, sensor interfaces, processing units, or previous computational stages. Both operands are simultaneously supplied to the multiplier block to initiate the MAC operation. The

parallel acceptance of operands eliminates unnecessary waiting cycles and improves computational throughput.

Step 2: Unified Group-Decomposed Multiplication: The received operands are processed by the UGDM. In this stage, the input operands are partitioned into smaller groups, allowing parallel generation and processing of partial products. Unlike conventional array or Booth multipliers, the UGDM architecture reduces the critical multiplication path by exploiting group decomposition and concurrent partial-product evaluation. The generated partial products are efficiently combined to obtain the multiplication result while minimizing switching activity and hardware complexity. Consequently, the UGDM block significantly reduces multiplication delay and power consumption while maintaining high computational accuracy.

Step 3: Predictive Skip-Merge Addition: The multiplication output generated by the UGDM is forwarded to the PSMA. Simultaneously, the delayed accumulated value from the previous cycle is supplied to the same adder through the feedback path. The PSMA predicts carry propagation patterns and employs skip-merge logic to bypass unnecessary carry computations. This predictive mechanism substantially shortens the carry propagation path compared with conventional ripple-carry, carry-select, or carry-lookahead adders. As a result, the adder performs high-speed accumulation of the current multiplication result and the previously stored accumulation value, thereby reducing overall MAC latency.

Step 4: Accumulation of Intermediate Results: The output generated by the PSMA is transferred to the Accumulator block. The accumulator functions as a storage register that preserves the current accumulated sum at every clock cycle. This accumulated value represents the running sum of all multiplication operations performed up to the current cycle. By storing intermediate results, the accumulator enables iterative MAC computations without requiring external memory access, thereby improving computational efficiency and reducing system overhead.

Step 5: Feedback Through Delay Element: The accumulated result is subsequently passed through a Delay block. This delay element acts as a clocked register that synchronizes the feedback path with the system clock. The delayed accumulation value is then routed back to the PSMA input for use during the next accumulation cycle. The delay block ensures stable timing behavior, eliminates race conditions, and provides proper synchronization between consecutive MAC operations. This feedback mechanism forms the core of the multiply–accumulates process.

Unified Group-Decomposed Multiplier

The UGDM is a high-performance multiplication architecture as shown in Figure 4.2 designed to reduce computational delay, hardware complexity, and switching activity in VLSI systems. Unlike conventional array multipliers that process all bits simultaneously through many partial-product generators, the UGDM architecture divides the input operands into smaller groups and performs multiplication in a structured hierarchical manner. This decomposition enables parallel processing of operand groups, resulting in shorter critical paths and improved operating frequency. The architecture typically consists of Input Registers, Group Decomposition Unit, Partial Product Generation Unit, Partial Product Reduction Network, Intermediate Merge Unit, and Final Product Generation Unit. By combining parallelism and efficient reduction mechanisms, UGDM achieves high-speed multiplication with reduced area and power consumption, making it suitable for DSP processors, MAC units, AI accelerators, FPGA implementations, and VLSI arithmetic processors.

Step 1: Input Register Unit: The multiplication process begins with two N−bit operands, A[N−1:0] and B[N−1:0], entering the UGDM architecture through input registers. These registers synchronize the incoming data with the system clock and provide stable inputs for subsequent processing stages. Registering the inputs minimizes timing uncertainties and ensures

reliable operation at higher clock frequencies. The registered operands are then forwarded to the decomposition stage.

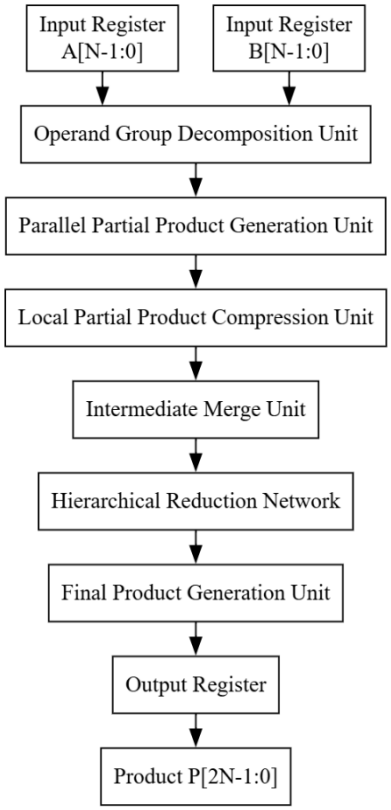


Figure 2. Proposed UGDM Design**.**

Step 2: Operand Group Decomposition Unit: In this stage, both operands are partitioned into multiple smaller groups of equal size. For example, an N-bit operand can be divided into K groups, where each group contains G bits. Each subgroup represents a smaller multiplication problem. The decomposition process reduces the complexity of large multipliers by converting them into several smaller multiplication blocks. Since these groups operate independently, they can be processed concurrently, significantly increasing computational speed.

Step 3: Parallel Partial Product Generation Unit: Each decomposed group pair generates partial products simultaneously. Dedicated AND-gate arrays or optimized partial-product generators produce the multiplication results for every group combination. Because all group multiplications occur in parallel, the multiplication latency is substantially reduced compared with sequential multiplication architectures. This stage forms the primary source of parallelism within the UGDM structure.

Step 4: Local Partial Product Compression Unit: The generated partial products are then processed by local compression blocks. These blocks employ hardware structures such as: Half Adders (HA), Full Adders (FA), Carry Save Adders (CSA), and Compressor Networks (3:2, 4:2, or 5:2 Compressors). The objective is to reduce multiple partial-product rows into fewer intermediate rows while avoiding long carry-propagation chains. Carry-save operations enable addition without immediate carry propagation, thereby reducing the critical path delay.

Step 5: Intermediate Merge Unit: After local compression, the compressed outputs from different operand groups are aligned according to their bit significance and merged together. This stage combines the outputs generated by individual decomposed multiplier blocks into a unified multiplication result.

Merged Product = PP00 + (PP01 << G) + (PP10 << G) + (PP11 << 2G)

Appropriate bit shifting ensures that each partial product contributes to the correct position in the final multiplication result. The merge unit effectively reconstructs the multiplication operation from the decomposed groups.

Predictive Skip-Merge Adder

The PSMA is a high-speed VLSI adder architecture as shown in Figure 4.3 designed to minimize carry propagation delay during arithmetic operations. Unlike conventional Ripple Carry Adders (RCA), where the carry signal propagates sequentially through every bit position, the PSMA predicts carry behavior and employs skip-merge logic to accelerate addition. The architecture combines Carry Prediction Units, Propagate-Generate Logic, Skip Control Networks, Merge Units, Sum Generation Blocks, and Output Registers to achieve faster arithmetic computation. By intelligently predicting carry paths and allowing carries to bypass unnecessary stages, the PSMA significantly reduces critical-path delay, power consumption, and switching activity. Consequently, it is highly suitable for high-performance MAC units, DSP processors, FPGA accelerators, AI hardware, and VLSI arithmetic systems where rapid accumulation operations are required.

Step 1: Input Register Unit: The addition process begins when two N-bit operands enter the PSMA through synchronized input registers. These registers stabilize the incoming data and align the operands with the system clock. Registering the inputs improves timing reliability and facilitates high-frequency operation. The registered operands are subsequently supplied to the carry prediction stage.

Step 2: Propagate–Generate Logic Unit: For every bit position, the PSMA computes the Propagate (P) and Generate (G) signals.

Propagate Signal: $P_i = X_i Y_i$

The propagate signal indicates whether an incoming carry will pass through the current bit position.

Generate Signal: $G_i = X_i Y_i$

The generate signal determines whether the current bit position independently generates a carry. These signals form the foundation of predictive carry computation and are generated simultaneously for all bit positions.

Step 3: Carry Prediction Unit: The generated propagate and generate signals are processed by the Carry Prediction Unit (CPU). Instead of waiting for carries to ripple through all bits, the CPU predicts the carry status of higher-order groups using local information. This prediction mechanism determines whether carries are likely to propagate, generate, or terminate within a particular group. Consequently, future carry values can be estimated before actual carry propagation completes, reducing arithmetic latency.

Step 4: Skip Control Network: The predicted carry information is forwarded to the Skip Control Network. This unit identifies regions where carry propagation can be safely bypassed. If all bits within a group exhibit propagate conditions, the incoming carry skips directly to the next group instead of passing through every intermediate stage. This skipping mechanism substantially shortens the critical path.

Step 5: Sum Generation Unit: The final carry signals are combined with the propagate signals to generate the output sum bits.

Sum Computation: $S_i = P_i C_i$

where:

- S_i= Sum bit
- P_i= Propagate signal
- C_i= Carry input

The sum generation process occurs in parallel across all bit positions, enabling rapid completion of the addition operation.

Step 6: Output Register Unit: The generated sum and carry-out signals are stored in output registers. These registers isolate the PSMA from subsequent processing stages and support pipelined operation in high-speed VLSI systems.

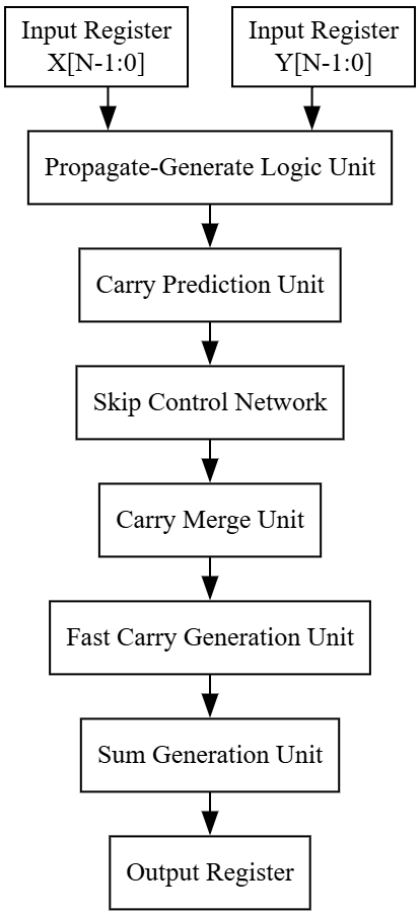


Figure 3. Proposed PSMA Design.

4. Results and Discussion

Figure 4 illustrates the functional simulation results of the proposed UGDM–PSMA MAC architecture. The waveform confirms the correct execution of multiply–accumulate operations for varying input operands. The input signal a[31:0] assumes values such as 8, 17, 5, 1, 18, 9, and 17, while b[31:0] changes through values including 17, 7, 17, 2, 9, 17, 12, 6, and 2. The enable signal remains asserted throughout the operation, allowing continuous arithmetic processing. The output signal out[63:0] exhibits a progressive accumulation pattern with values increasing from 0 to 136, 272, 391, 510, 799, 1088, 1098, 1108, 1117, 1126, 1432, 1738, 1846, 1953, 2008, and finally 2062, demonstrating the successful accumulation of multiplication results over consecutive clock cycles. The parameter N[31:0] = 32 confirms a 32-bit operand implementation. The smooth transition of accumulated values and the absence of unexpected glitches indicate the functional correctness and stable operation of the proposed MAC architecture under simulation conditions.

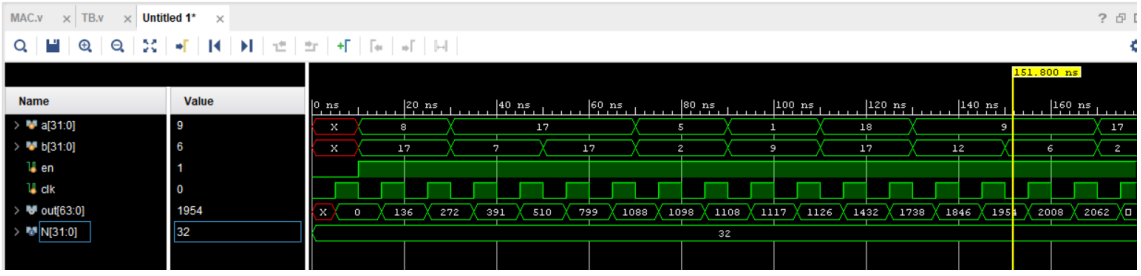


Figure 4. Proposed Simulation Outcome

Figure 5 presents the FPGA resource utilization summary of the proposed MAC architecture. The design utilizes 1324 LUTs out of the available 134,600 LUTs, resulting in a utilization of 0.98%, which is lower than the existing design utilization of 1.12%. The architecture consumes 64 Flip-Flops (FFs) from a total of 269,200 available FFs, corresponding to only 0.02% utilization, indicating minimal sequential storage requirements. The implementation uses 130 I/O ports out of 500 available resources, resulting in 26.00% utilization, while only 1 BUFG out of 32 available clock buffers is utilized, corresponding to 3.13% utilization. The reduction in LUT count from 1505 to 1324 LUTs demonstrates that the UGDM and PSMA structures effectively reduce combinational logic complexity while maintaining the same register and clocking requirements. So, the proposed architecture achieves improved area efficiency and better hardware utilization compared with the existing MAC implementation.

Resource	Estimation	Available	Utilization %
LUT	1324	134600	0.98
FF	64	269200	0.02
IO	130	500	26.00
BUFG	1	32	3.13

Figure 5. Proposed Area Outcome

Figure 6 illustrates the power consumption analysis of the proposed MAC architecture. The total dynamic power consumption is reported as 79.996 W, accounting for approximately 98% of the total power dissipation, while the static power consumption is only 1.235 W, representing approximately 2% of the total power. Within the dynamic power category, I/O power contributes 60.098 W (75%), signal power contributes 9.971 W (12%), and logic power contributes 9.926 W (12%). Compared with the existing architecture, where the dynamic power consumption was 101.677 W, the proposed design achieves a significant reduction of approximately 21.681 W. Furthermore, signal power decreases from 16.782 W to 9.971 W, while logic power decreases from 24.857 W to 9.926 W, indicating lower switching activity and improved arithmetic efficiency. These results demonstrate that the UGDM–PSMA architecture effectively minimizes internal power dissipation while maintaining high computational performance, thereby enhancing overall energy efficiency.

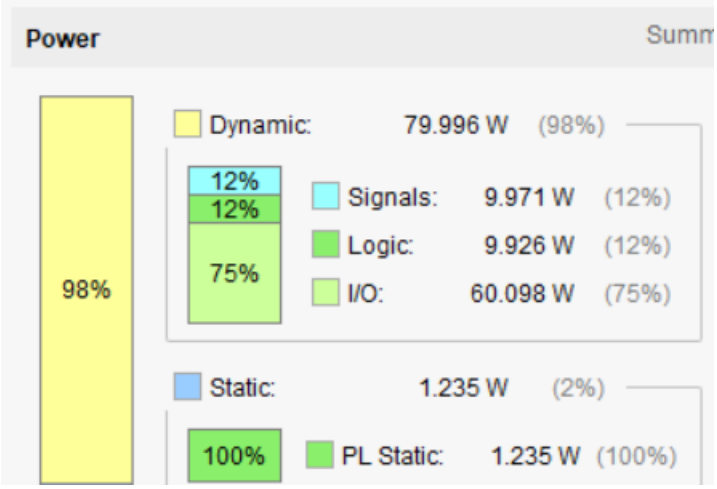
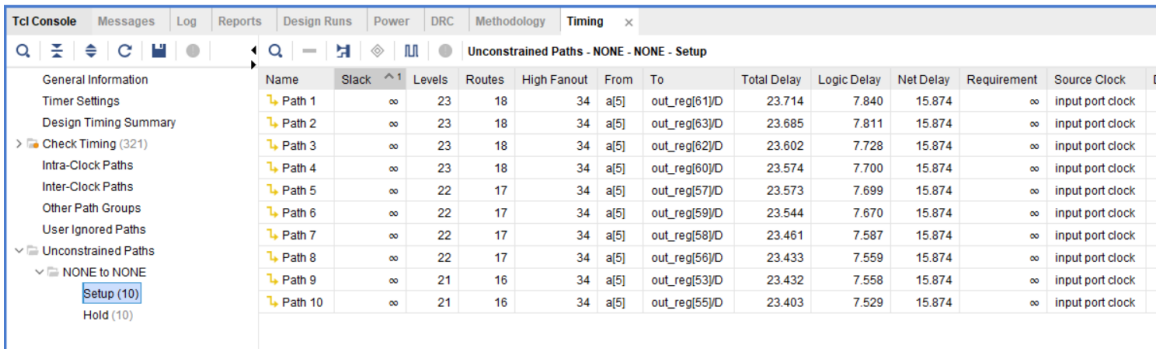


Figure 6. Proposed Power Summary

Figure 7 presents the setup timing analysis of the proposed MAC architecture. The critical timing paths exhibit total delays ranging from approximately 23.403 ns to 23.714 ns. The worst-case setup path, identified as Path 1, records a total delay of 23.714 ns, consisting of a logic delay of 7.840 ns and a net delay of 15.874 ns. Similarly, Path 2 exhibits a delay of 23.685 ns, while Path 10 records 23.403 ns. The critical paths contain between 21 and 23 logic levels, which is substantially lower than the 59–63

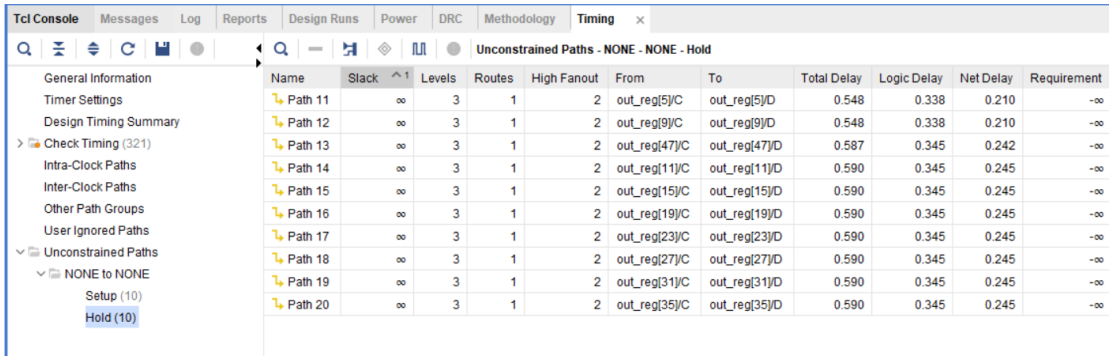
logic levels observed in the existing design. Moreover, the routing delays are significantly reduced compared with the existing architecture, where delays exceeded 70 ns. The reduction of the worst-case setup delay from 74.224 ns to 23.714 ns corresponds to an improvement of approximately 68.05%, demonstrating that the proposed UGDM–PSMA architecture significantly shortens the critical path and enables higher operating frequencies. This improvement directly contributes to enhanced throughput and faster arithmetic execution.



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	23	18	34	a[5]	out_reg[61]/D	23.714	7.840	15.874	∞	input port clock
Path 2	∞	23	18	34	a[5]	out_reg[63]/D	23.685	7.811	15.874	∞	input port clock
Path 3	∞	23	18	34	a[5]	out_reg[62]/D	23.602	7.728	15.874	∞	input port clock
Path 4	∞	23	18	34	a[5]	out_reg[60]/D	23.574	7.700	15.874	∞	input port clock
Path 5	∞	22	17	34	a[5]	out_reg[57]/D	23.573	7.699	15.874	∞	input port clock
Path 6	∞	22	17	34	a[5]	out_reg[59]/D	23.544	7.670	15.874	∞	input port clock
Path 7	∞	22	17	34	a[5]	out_reg[58]/D	23.461	7.587	15.874	∞	input port clock
Path 8	∞	22	17	34	a[5]	out_reg[56]/D	23.433	7.559	15.874	∞	input port clock
Path 9	∞	21	16	34	a[5]	out_reg[53]/D	23.432	7.558	15.874	∞	input port clock
Path 10	∞	21	16	34	a[5]	out_reg[55]/D	23.403	7.529	15.874	∞	input port clock

Figure 7. Proposed Setup Delay Outcome

Figure 8 shows the hold timing analysis results of the proposed MAC architecture. The hold paths exhibit delays ranging from 0.548 ns to 0.590 ns, which are identical to those observed in the existing implementation. The minimum delay is observed in Path 11, where the total delay is 0.548 ns, consisting of a logic delay of 0.338 ns and a net delay of 0.210 ns. Other paths, including Paths 14 through 20, exhibit total delays close to 0.590 ns, with logic delays around 0.345 ns and net delays around 0.245 ns. Each critical hold path consists of only 3 logic levels and a single routing stage, indicating very short minimum-delay paths. The absence of negative slack and the consistency of hold timing values demonstrate that the proposed architecture maintains proper synchronization between sequential elements. Therefore, while the proposed design significantly improves setup timing performance, it also preserves stable hold-time characteristics, ensuring reliable operation without introducing timing violations.



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 11	∞	3	1	2	out_reg[5]/C	out_reg[5]/D	0.548	0.338	0.210	-∞
Path 12	∞	3	1	2	out_reg[9]/C	out_reg[9]/D	0.548	0.338	0.210	-∞
Path 13	∞	3	1	2	out_reg[47]/C	out_reg[47]/D	0.587	0.345	0.242	-∞
Path 14	∞	3	1	2	out_reg[11]/C	out_reg[11]/D	0.590	0.345	0.245	-∞
Path 15	∞	3	1	2	out_reg[15]/C	out_reg[15]/D	0.590	0.345	0.245	-∞
Path 16	∞	3	1	2	out_reg[19]/C	out_reg[19]/D	0.590	0.345	0.245	-∞
Path 17	∞	3	1	2	out_reg[23]/C	out_reg[23]/D	0.590	0.345	0.245	-∞
Path 18	∞	3	1	2	out_reg[27]/C	out_reg[27]/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[35]/C	out_reg[35]/D	0.590	0.345	0.245	-∞

Figure 8. Proposed Hold Delay Outcome

Comparative Analysis

Table 1 presents the area utilization comparison between the existing MAC architecture and the proposed UGDM–PSMA MAC architecture. The existing design requires 1505 LUTs, whereas the proposed design utilizes only 1324 LUTs, resulting in a 12.03% reduction in LUT consumption. Similarly, LUT utilization decreases from 1.12% in the existing architecture to 0.98% in the proposed architecture, corresponding to a 12.50% improvement. Since LUTs are the primary resources responsible for implementing combinational logic in FPGA designs, the reduction in LUT usage indicates that the proposed architecture achieves better hardware efficiency and reduced logic complexity. The lower resource utilization also provides additional FPGA capacity for integrating other processing modules and contributes to improved area-delay characteristics. Overall, the proposed

architecture demonstrates superior area optimization while maintaining the required arithmetic functionality. LUT utilization is a key measure of FPGA logic efficiency and directly reflects the amount of combinational hardware required by a design.

Table 1. Area Comparison Between Existing and Proposed MAC Architectures

Resource	Existing MAC	Proposed MAC	Improvement
LUT	1505	1324	12.03% Reduction
LUT Utilization (%)	1.12	0.98	12.50% Reduction

Table 2 compares the power consumption characteristics of the existing and proposed MAC architectures. The existing design exhibits a dynamic power consumption of 101.677 µW, while the proposed architecture reduces it to 79.996 µW, achieving a 21.32% reduction. Signal power decreases significantly from 16.782 µW to 9.971 µW, resulting in a 40.58% reduction, whereas logic power decreases from 24.857 µW to 9.926 µW, corresponding to a substantial 60.07% reduction. Additionally, the dynamic power contribution decreases from 99% to 98%, indicating improved energy utilization within the arithmetic circuitry. These reductions demonstrate that the proposed UGDM–PSMA architecture effectively minimizes switching activity and internal logic transitions, thereby lowering overall energy consumption. Reduced dynamic power is particularly beneficial for battery-operated devices, FPGA accelerators, and high-performance computing systems where energy efficiency is a critical design objective. Dynamic power in digital circuits is strongly influenced by switching activity and interconnect capacitance, making reductions in signal and logic activity highly effective for power optimization.

Table 2 Power Comparison Between Existing and Proposed MAC Architectures

Power Parameter	Existing MAC (uW)	Proposed MAC (uW)	Improvement
Dynamic Power	101.677	79.996	21.32% Reduction
Signal Power	16.782	9.971	40.58% Reduction
Logic Power	24.857	9.926	60.07% Reduction
Dynamic Power Contribution (%)	99	98	Improved

Table 3 presents the timing performance comparison between the existing MAC architecture and the proposed UGDM–PSMA MAC architecture. The worst-case setup delay is reduced from 74.224 ns in the existing design to 23.714 ns in the proposed design, representing a significant 68.05% reduction. Similarly, the logic delay decreases from 11.340 ns to 7.840 ns, achieving a 30.86% improvement, while the net delay is reduced from 62.884 ns to 15.874 ns, corresponding to a remarkable 74.76% reduction. Furthermore, the number of logic levels decreases from 63 levels to 23 levels, resulting in a 63.49% reduction in logic depth. The reduction in logic levels and routing delay directly shortens the critical path, enabling higher operating frequencies and faster arithmetic execution. These results confirm that the proposed architecture substantially improves timing performance by reducing both combinational logic complexity and routing overhead, thereby enhancing throughput and computational efficiency. FPGA timing performance is heavily influenced by logic depth, LUT count within critical paths, and routing delays, all of which contribute to overall setup timing behavior.

Table 3 Delay Comparison Between Existing and Proposed MAC Architectures

Timing Parameter	Existing MAC (ns)	Proposed MAC (ns)	Improvement
Worst Setup Delay	74.224	23.714	68.05% Reduction

Logic Delay	11.340	7.840	30.86% Reduction
Net Delay	62.884	15.874	74.76% Reduction
Logic Levels	63	23	63.49% Reduction

5. Conclusion

The experimental results demonstrate that the proposed MAC architecture significantly improves performance over the existing design by reducing LUT utilization, power consumption, and critical path delay. The integration of UGDM and PSMA achieves notable reductions in area and dynamic power while greatly improving timing performance through shorter logic depth and faster accumulation. Simulation results confirm accurate arithmetic functionality and stable operation across multiple clock cycles. These enhancements make the architecture highly suitable for high-speed, low-power VLSI applications such as DSP processors, AI accelerators, and communication systems. Future work can focus on extending the design to higher precision arithmetic, incorporating adaptive optimization and power-management techniques, and implementing the architecture using advanced semiconductor technologies to further improve efficiency and support emerging computing applications.

References

[1]. Bakaramji, Kharate Ashish, and Satnam Singh. "VLSI DESIGN AND LOW-POWER IMPLEMENTATION OF MULTIPLY-ACCUMULATE UNITS FOR HIGH-EFFICIENCY DIGITAL FIR FILTERS." Transformative Education: A Multidisciplinary Journal (TEMJ) E-ISSN: 3117-8057 2, no. 2 (2026): 64-70.

[2]. A. Fasolino, R. Liguori, L. Di Benedetto, A. Rubino and G. D. Licciardo, "Overflow-Driven Dynamic Precision Scaling Fixed-Point Multiply-Accumulator Unit," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 34, no. 6, pp. 1814-1822, June 2026, doi: 10.1109/TVLSI.2026.3680254.

[3]. Z. Huo, W. Zhao, Y. Gong, T. Xia, Z. Wang and P. Ren, "A Groupwise Add–Multiply–Shift–Accumulate Datapath for Efficient DNN Accelerators," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, doi: 10.1109/TVLSI.2026.3689624.

[4]. Zhang, Jin, Zhongzhen Tong, Qiang Zhao, Chunyu Peng, Wenjuan Lu, Jian Zhou, Xiaoxiao Chen, Zhiting Lin, and Xiulong Wu. "A CIM Macro Embedded With Sign Operations for Parallel Signed Multibit Multiplication-and-Accumulation Using Hybrid Cell Array." IEEE Transactions on Very Large Scale Integration (VLSI) Systems (2026).

[5]. Suguna, T., Bharath Kumar Suresh, and Afrid Malim Malim. "Design and Performance Analysis of Multiply and Accumulate (MAC) Unit for Machine Learning Acceleration." JOURNAL OF COMPUTER SCIENCE (ISSN NO: 1549-3636) 19, no. 3 (2026).

[6]. Ahmadpour, Seyed-Sajad, Muhammad Zohaib, Hadi Rasmi, and Nima Jafari Navimipour. "High-performance and low-power quantum-dot-based multiply–accumulate design for next-generation supercomputing platforms." The Journal of Supercomputing 82, no. 2 (2026): 106.

[7]. Venkatachalam, Adithya, Umadevi Seerengasamy, and Abraham Sudharson Ponraj. "A scalable automated framework for multiply-accumulate unit design in high-performance computing applications." Scientific Reports 15, no. 1 (2025): 42713.

[8]. Kartheek, S. V., T. V. Sudeep, P. Harish, and M. Vinodhini. "An Optimized Multiply Accumulate Unit for Embedded Applications." In 2025 Fifth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), pp. 1-6. IEEE, 2025.

[9]. Dewan, Suvrodeep Bagchi, and Abhishek Basu. "VLSI Implementation of Multiplier and Accumulator." In International Conference on Sustainable Communication, Machine Intelligence and Metaverse, pp. 15-28. Singapore: Springer Nature Singapore, 2025.

[10]. Peram, Ravindra Reddy, and Sarda Sharma. "A Power-Efficient and High-Speed Multiply-Accumulate Unit for Arithmetic Operations." In 2025 IEEE 2nd International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS), pp. 1-6. IEEE, 2025.

[11]. Sakthi, G., Abhishek N. Tripathi, Jagana Bihari Padhy, Indrasen Singh, and Vivek Rajpoot. "Energy-Efficient Multiply-Accumulate Architecture Using Compressor Logic for FPGA-Based Systems." In 2025 IEEE 3rd International Symposium on Sustainable Energy, Signal Processing and Cybersecurity, pp. 1-6. IEEE, 2025.

[12]. Kalyan, P. Santhosh, and M. Vinodhini. "64-Bit Multiplier Accumulator Unit using Enhanced Vedic Multiplier." In 2025 6th International Conference on Electronics and Sustainable Communication Systems (ICESC), pp. 7-13. IEEE, 2025.

[13]. Nazeri, Nazirul Hafiz Mohd, Siti Zarina Md Naziri, and Rizalafande Che Ismail. "Development of High-Performance Multiply-Accumulate (Mac) Unit Design on Fpga for DSP Applications." In TENCON 2025-2025 IEEE Region 10 Conference (TENCON), pp. 248-252. IEEE, 2025.

[14]. Weingarten, Lennart, Kamalika Datta, and Rolf Drechsler. "ForMAt: Formal Verification of Scalable Multiply and Accumulate Units." In 2025 Forum on Specification & Design Languages (FDL), pp. 1-7. IEEE, 2025.

[15]. Gowd, Anil, and S. Jayaprakash. "RTL-Based Design and ASIC Front-End Realization of a High-Speed Multiply-Accumulate (MAC) Unit for DSP Applications Using 45 nm Technology." In 2025 IEEE International Conference on Communication Networks and Computing (CNC), pp. 1312-1316. IEEE, 2025.