International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I5P114

Volume 74 Issue 5, 205-218, May 2026

Original Article

Advanced Design Approaches for Bit- and Digit-Level Systolic Array Multiplication

Pradnya Zode[1], Pravin Zode[2], Nilesh Ashtankar[3], Nilesh Shelke[4]

[1,2]*Department of Electronics Engineering, Yashwantrao Chavan College of Engineering, Nagpur, India.*
[3,4]*Symbiosis Institute of Technology, Nagpur Campus, Symbiosis International Deemed University, Pune, India.*

[1]*Corresponding Author : pradnyazode@gmail.com*

Received: 16 September 2025 Revised: 07 March 2026 Accepted: 12 March 2026 Published: 30 May 2026

Abstract *- This paper emphasizes advanced Architectural Designs of multipliers using systolic arrays explicitly augmented for Bit and Digit-Level Multiplication. These array multipliers are appropriate for high-speed, low area as well as parallel multiplication applications like Digital Signal Processing, Cryptography, and Artificial Intelligence. The array multipliers have a regular structure with efficient and very small interconnections so that the incoming real-time data can flow in a pipelined manner. With respect to this, different types of array multiplier with bit serial and digit serial processing methods are designed. The designed structures are assigned different mixtures of binary numbers as an input for finding the proper product bits. Advanced Bit-Level and Digit-Level with size two are developed and synthesized for 8 by 8 bits carry save array, Carry Ripple Array, and Baugh-Woolley multiplier. Towards this, dependence graphs for all the arrays of all multipliers are also designed in Verilog and synthesized on the board Xilinx ML605 (Virtex 6 XC6VLX240T-1FFG1156 FPGA). The detailed performance comparison between regular Carry Save, Carry Ripple, and Baugh-Wooley array multipliers examines the competence of these three different multiplier architectures using significant recital metrics such as delay, power consumption, and area utilization with the different word lengths from 4-bit to 128-bit. Out of these three, Baugh-Wooley array multipliers steadily deliver the maximum improvement in delay, with up to 55.02% at 4-bit and 38.51% at 128-bit. It also displays impactful enhancements in power up to 43.72% and area up to 35.24% than both carry save and Carry Ripple Array multiplier. Carry Ripple Array multiplier demonstrations reasonable enhancements, predominantly in power (up to 37.4% at 4-bit), but less significant than Baugh-Wooly. As word length intensifications, Baugh-Wooley's array multiplier proficiency governs across all three systems of measurement. These results highlight Baugh-Wooley's appropriateness for high-speed, low-power, and area-efficient VLSI designs.*

Keywords *- Systolic arrays, bit and digit level, Array multiplier, Carry save array, Carry ripple array, Baugh-Wooley multiplier dependence graph.*

1. Introduction

The noteworthy developments in the field of microelectronics and computing have been essential, enabling the extensive acceptance and advanced proficiencies of the field of Digital Signal Processing (DSP) [1]. The systematic and processing abilities of DSP techniques are progressively employed diagonally in numerous fields, which include engineering, science, medicine, and economics [2]. DSP is concerned with different investigating operations on real-time signals and data in experimental form. In addition, subtraction, delays, and multiplication are the most commonly used operations to develop any DSP working algorithm. Out of these operations, the multiplier is the circuit that consumes more power, area, and takes more time to perform its computations. Therefore, this work has focused on the design and development of power, speed, and area-efficient multiplier circuits for DSP applications. Bit-level architecture is a fundamental and regularly encountered concept in digital circuits, more specifically, signal processing systems and algorithms [3]. It facilitates very fast and more efficient processing of the real-time incoming data at the bottom probable granularity, i.e., the binary digit 0 or digit 1. These architectures have pushed the development of new modern daily needed processors and continue to be decisive in extraordinary-performance and focused computational fields. These architectures work or manipulate individual bits of data. Different bit-level architecture styles are bit-parallel, bit-serial, and digit-serial. Bit-parallel architectures are particularly suited for deployments requiring data processing with maximum speed, as they process one complete word of the real-time input sample per clock cycle. They can be used to implement data transfer procedures with a low difficulty level and high data rate. Next, bit-serial systems are ideal for area-efficient and low power-consuming applications, which

can be easily synthesized with the help of a scheduling approach, based on the mechanism of processing one digital bit of real-time sample input per clock cycle. These systems can be used to implement data transfer procedures with a medium difficulty level and low to medium data rate. Next, the architectures needed for high data rate, with crucial area and power consumption applications, digital serial systems are very useful. They work on the principle of processing a large number of bits of real-time input samples, which is called data size per clock cycle.

Although various multiplier architectures based on bit-parallel, bit-serial, and digit-serial methods have been described in the literature, most prevailing designs primarily optimize a single performance metric, such as speed or area, often at the cost of uncomplimentary trade-offs in power consumption or architectural complexity. Furthermore, many digit-serial designs reported in prior work introduce increased control complexity or limited scalability, which restricts their applicability in energy-efficient and high-performance DSP systems. This work focuses on optimization of the proposed design in all aspects of the Very Large-Scale Integration (VLSI) design current scenario.

For this paper, all real-time numbers are supposed to be represented in fixed-point 2's complement representation format. An N-bit number A can be represented as

$A = a_{N-1},a_{N-2},a_{N-3},\ldots..a_2,a_1,a_0.$

The bits in the range from 0 to N-1 can be 1 or 0, with the most significant bit always gives the sign bit, 0 as a positive and 1 as a negative number. The general formula for the number representation is given in Equation (1)

$$P = -p_{N-1} + \sum_{i=1}^{N-1} p_{N-1-i} \times 2^{-i} \quad (1)$$

An exact correct output of any arithmetic operation is the biggest advantage of the 2's complement representation of a number. The range for a number representation in fixed-point 2's complement with n bits is -2^{n-1} to 2^{n-1-1}. So, for an 8-bit system, the range of representation is from −128 to +127. To develop the bit serial and digit serial architecture for any multiplier, first, a Dependence Graph (DG) showing data processing is to be designed. An effective and very efficient graphical representation of any algorithm is known as DG which is a Directed Graph (DG). The primary concept of a DG is to represent the dependencies between the input signal and its computations on the next component of any digital system. Nodes and their connecting edges are the basic components of a DG, where nodes are the main computations of any algorithms, like adder, subtractor, multiplier, or divider. Moreover, the edges connecting two nodes give the flow of computational results from one node to another node. The principal advantage of DG is speed, as the distance between two nodes is very little, so the time needed for the input to go from one node to another is almost negligible. DGs are commonly used to implement any DSP algorithms in VLSI hardware. The DG of the 8×8 multiplier design shows that both partial product generation and carry-save addition stages are controlled by dependencies on prior input signals and intermediate results propagated through successive computational levels. This supports further parallel processing, pipelining, as well as optimization of speed, area, and power consumption. The novelty of this work lies in the systematic use of dependence graph–driven architecture to achieve improved speed, reduced area, and lower power consumption compared to existing bit-serial and digit-serial multiplier designs, while maintaining architectural simplicity and scalability. These features make the proposed multipliers well-suited for DSP-concerned with VLSI applications.

2. Literature Review

Adder, multiplier, and delay elements are the major components of any DSP algorithm. Out of which multiplier is the most powerful, area- and speed-intensive component. Accordingly, it is always better to design a high-speed and low-power-consuming multiplier circuit. To make the multiplier operation faster, an array of them is constructed, known as an array multiplier. It is a kind of combinational digital circuit developed for the multiplication of two binary numbers. An array multiplier has a wide range of applications in the field of DSP and VLSI hardware implementations because of its consistent construction and simplicity. Partial Product Generation with the help of AND gates and then its addition, are the two important operations performed by an array multiplier. Partial Product addition is performed by a half and full adder circuits using the shifting technique. Distinct types of Array Multipliers are the unsigned Array Multiplier and the Signed Array Multiplier. Carry-Save Array Multiplier (CSAM), Carry-Ripple Array Multiplier (CRAM), Baugh-Wooley Array Multiplier (BWAM), Booth's Multiplier, Wallace tree Multiplier are the different types of Signed Array Multiplier. In this paper, DG for 8 by 8 CSAM, CRAM, and BWAM are designed, and then the bit serial and digit serial architectures are developed. The developed designs are valuable for VLSI hardware implementation of real-time digital signals in high-speed or low-power applications. The innovative techniques were presented for accomplishing faster and more energy-efficient column reduction multiplication with minimal area used by [4]. 8-bit, 16-bit, 32-bit, and 64-bit Baugh Wooley (BW) multipliers, utilizing Wallace, Dadda, and HPM reduction trees, are designed with two methods: i.e., partitioning partial products for independent parallel reduction, and accelerating the final addition stage with new hybrid adder arrangements. An energy-efficient fast array multiplier is projected and designed by [5], where the least-significant half of the product is found with a carry-ripple adder through the curtailment. The BW multiplier with a marked number by two's complement shape is designed. A detailed review of different Architectures for Booth Multiplication Algorithms implemented in VLSI is

explained by [6]. A novel, precise, reconfigurable booth multiplier circuit is proposed by [7]. The design includes partial error adjustment with the addition of sign bits in a wrecker array multiplier. The formation of a small and high-performance BW multiplier with the help of Quantum Dot Cellular Automata is designed by [8]. BW and sign-magnitude are used by [9] as initial preparation processes to comprehend two's complement 8 by 8 bits multipliers with LUTs available in FPGAs. These two algorithms are employed as they facilitate the parallel implementation of signed multipliers. [10] presents high-speed, power-efficient signed fixed-point multipliers for DSP utilities on Xilinx (AMD) FPGAs. The design diminishes combinational path delay by combining partial product generation employing LUT-based Booth radix-4 with Bewick's sign extension, integrated into a Dadda-based concurrent architecture, and partial product decrease using carry-save adders. A new approach for array multipliers consuming approximate adders by cultivating efficiency in terms of resource usage and area to implement a 4-bit and 8-bit array multiplier by using 2-bit approximation adders is proposed by [11].

[12] has studied a comprehensive evaluation of different frequently used adders- Ripple Carry Adder (RCA), Carry Look-Ahead Adder (CLA), Carry Save Adder (CSA), Carry Select Adder (CSLA), and Carry Skip Adder (CSKA) with a 4-bit length to find the best performing adder. The selected best adder was then used in the design of array, Booth, and Vedic multipliers. Employment and performance study of different digital parallel array multipliers is presented by [13]. The multipliers are the conventional array multiplier, modified Braun's multiplier, Vedic multiplier, Booth's multiplier, and BW multiplier. It gives the plan principles, algorithms, and hardware implementation of all multipliers. An efficient multiplier is designed to use the computational power of LUTs by implementing complex logic functions with the lowest resource consumption and latency by [14].

A unique approximation method for signed multiplication on FPGAs is proposed by [15]. The method includes a novel segmentation technique useful to the BW multiplication algorithm. [16] designed a BW Multiplier integrating a CLA based on Quaternary logic with the help of the Wallace Tree algorithm to improve the functionality. Different multiplier architectures like Dadda, Array, Booth, Vedic, and Wallace Tree are implemented at the gate level in Verilog and synthesized by [17]. In paper [18], a pipelined multiplier using the Karatsuba formula for elliptic curve cryptography is designed. The implementation was also done on an FPGA. A Multiplier Circuit for Power Factor Correction Controllers circuit is proposed in [19]. The proposed circuit is analog in nature; if it is implemented with the proposed array multiplier, it will help reduce the noise and power consumption. In [20], a novel mixed-mode in-storage-computing architecture is proposed as an efficient and promising solution for large-scale matrix–vector multiplication operations. The paper [21] examines an efficient LUT6-based multiplier for Montgomery Modular Multiplication (MMM) on an FPGA. Primarily, the LUT6 cost of multipliers implemented using existing Radix-4, Radix-8, and Radix-16 Booth methods is analyzed and compared. Based on this analysis, an improved LUT6-based Radix-16 Booth method tailored for Coarsely Integrated Operand Scanning MMM is proposed. The paper [22] introduces Bit-Cigma, a hardware-centric architecture that exploits bit-sparsity to achieve efficient acceleration of generic matrix multiplication. The designed architecture has different features where processing elements are used to perform multiplication.

The authors in [23] propose efficient algorithms for modular reduction to design novel systolic and non-systolic architectures for polynomial basis finite field multipliers used in Reed–Solomon codecs. Their bit-parallel systolic design achieves nearly two-thirds the area of existing counterparts with the same latency, while non-systolic serial/parallel designs significantly reduce hardware complexity and area-time product. Further, balanced-tree bit-parallel multipliers with sub-expression sharing, achieving up to 65% of area savings compared to existing designs, are presented.

The rest of the paper is arranged as follows: In Section 2, a literature review regarding parallel multipliers is given. In Section 3, the working of the parallel carry save ripple multiplier is presented. Section 4 demonstrates the working of the parallel carry save array multiplier. Section 5 illustrates the working of the Baugh-Wooley array multiplier. Section 6 provides the mathematical representation of 8 by 8 signed-bit parallel multiplication. Section 7 shows the simulation-based results and provides a discussion on the comparison of the three different array multipliers, and the last section summarizes the conclusions.

3. Parallel Multipliers

The multiplication operation is performed by the multiplier component in the hardware. A multiplier is the component of any VLSI system that consumes large areas and power. Parallel multipliers are the components fabricated to perform multiplication in parallel by processing a number of real-time input sample bits concurrently to improve processing speed.

These designs are important for applications where speedy arithmetic operations are needed, such as the arithmetic logic unit of a computer, as well as DSP units. These designs are based on the primary theory of producing partial products from the multiplicand and multiplier, across the able number addition to get the proper result. Various types of parallel multipliers are available in the field of VLSI, such as Wallace Tree Multipliers, Dadda Multipliers, Array Multiplier, Booth Multiplier, CSAM, CRAM, Braun Multipliers, BW Multipliers, and Logarithmic Multipliers.

								p7	p6	p5	p4	p3	p2	p1	p0
								q7	q6	q5	q4	q3	q2	q1	q0
								-q0p7	q0p6	q0p5	q0p4	q0p3	q0p2	q0p1	q0p0
							-q1p7	q1p6	q1p5	q1p4	q1p3	q1p2	q1p1	q1p0	**
						-q2p7	q2p6	q2p5	q2p4	q2p3	q2p2	q2p1	q2p0	**	**
					-q3p7	q3p6	q3p5	q3p4	q3p3	q3p2	q3p1	q3p0	**	**	**
				-q4p7	q4p6	q4p5	q4p4	q4p3	q4p2	q4p1	q4p0	**	**	**	**
			-q5p7	q5p6	q5p5	q5p4	q5p3	q5p2	q5p1	q5p0	**	**	**	**	**
		-q6p7	q6p6	q6p5	q6p4	q6p3	q6p2	q6p1	q6p0	**	**	**	**	**	**
	-q7p7	q7p6	q7p5	q7p4	q7p3	q7p2	q7p1	q7p0	**	**	**	**	**	**	**
-p7q7	p6q7	p5q7	p4q7	p3q7	p2q7	p1q7	p0q7	**	**	**	**	**	**	**	**
M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Fig. 1 Eight by eight multiplication operation

Consider the 8-bit multiplicand and multiplier numbers as P and Q, then their representation as per Equation (1), where P and Q cover the value from -1 to +1 are given in Equations (2) and (3), which will vary from -1 to +1. Equation (4) gives the multiplication of P and Q numbers, denoted by the letter M.

$$P = p_{N-1}.p_{N-2} \ldots . p_2 p_1 p_0 + \sum_{i=1}^{N-1} p_{N-1-i} \times 2^i \quad (1)$$

$$Q = q_{N-1}.q_{N-2} \ldots . q_2 q_1 q_0 + \sum_{i=1}^{N-1} q_{N-1-i} \times 2^i \quad (2)$$

$$M = -m_{2N-2} + \sum_{i=1}^{2N-2} m_{2N-2-i} \times 2^{-i} \quad (3)$$

For multiplication to be done in parallel, the Horner's rule multiplication with sign extension is used, which for numbers P and Q is given in Equation (5):

$$M = P \times (-q_{N-1} + \sum_{i=1}^{N-1} q_{N-1-i} 2^{-i}) \quad (5)$$

The scaling operation by a factor of 2-1 is written in Equation (6):

$$M = -P.q_{N-1} + (P.q_{N-2} + (P.q_{N-3} + (P.q_{N-4} \ldots . + (P.q_1 + P.q_0 2^{-1}) \ldots .)2^{-1})2^{-1} \quad (6)$$

For making a number negative in the 2's complement method, as a standard rule, first the 1's complement of the number is found out and then 1 bit is added to the least significant bit is done which is completely shown in Equation (7):

$$-P = p_{N-1} - \sum_{i=1}^{N-1} p_{N-1-i} 2^{-i}$$

$$-P = p_{N-1} + \sum_{i=1}^{N-1} (1 - p_{N-1-i}) 2^{-i} - \sum_{i=1}^{N-1} 2^{-i}$$

$$-P = p_{N-1} + \sum_{i=1}^{N-1} (1 - p_{N-1-i}) 2^{-i} - 1 + 2^{-N+1}$$

$$-P = -(1 - p_{N-1}) + \sum_{i=1}^{N-1} (1 - p_{N-1-i}) 2^{-i} + 2^{-N+1}$$

$$-P = -\bar{p}_{N-1} + \sum_{i=1}^{N-1} \bar{p}_{N-1-i} 2^{-i} + 2^{-N+1} \quad (7)$$

In Equation (7) (1-ai) gives the one's complement of the number ai, which is denoted by ai with the bar, which will range from 0 to N-1. Therefore, the term -P.pN-1 in the 8-bit multiplication can be written as given by Equation (8):

$$-P \times q_7 = -\bar{p}_7 q_7 + \bar{p}_6 q_7 2^{-1} + \bar{p}_5 q_7 2^{-2} + \bar{p}_4 q_7 2^{-3} + \bar{p}_3 q_7 2^{-4} + \bar{p}_2 q_7 2^{-5} + \bar{p}_1 q_7 2^{-6} + \bar{p}_0 q_7 2^{-7} + q_7 2^{-8} \quad (8)$$

The above multiplication operation can be performed with the help of the structure shown in Figure 1. The sign extension method must be used to perform the above multiplication, as direct addition is not possible due to the negative weights on the numbers, as shown in Equation (9):

$$P = -p_7 + p_6 2^{-1} + p_5 2^{-2} + p_4 2^{-3} + p_3 2^{-4} + p_2 2^{-5} + p_1 2^{-6} + p_0 2^{-7}$$

$$P = -p_7 2 + p_7 + p_6 2^{-1} + p_5 2^{-2} + p_4 2^{-3} + p_3 2^{-4} + p_2 2^{-5} + p_1 2^{-6} + p_0 2^{-7}$$

$$P = -p_7 2^2 + p_7 2 + p_7 + p_6 2^{-1} + p_5 2^{-2} + p_4 2^{-3} + p_3 2^{-4} + p_2 2^{-5} + p_1 2^{-6} + p_0 2^{-7} \quad (9)$$

The above Equation (9) gives the sign extension for number P by 1 and 2 bits, but its value remains unaffected.

The scaling operation of 2-i must be performed as a right shift with sign extension when a number is to be represented in 2's complement form. As an example, scaling of the number P by 2-1 is to be shown as given in Equation (10):

$$P.2^{-1} = (-p_7 + p_6 2^{-1} + p_5 2^{-2} + p_4 2^{-3} + p_3 2^{-4} + p_2 2^{-5} + p_1 2^{-6} + p_0 2^{-7}) 2^{-1}$$

$$= -p_7 2^{-1} + p_6 2^{-2} + p_5 2^{-3} + p_4 2^{-4} + p_3 2^{-5} + p_2 2^{-6} + p_1 2^{-7} + p_0 2^{-8}$$

$$= -p_7 + p_7 2^{-1} + p_6 2^{-2} + p_5 2^{-3} + p_4 2^{-4} + p_3 2^{-5} + p_2 2^{-6} + p_1 2^{-7} + p_0 2^{-8}$$

$$= -p_7 + p_7 2^{-1} + p_6 2^{-2} + p_5 2^{-3} + p_4 2^{-4} + p_3 2^{-5} + p_2 2^{-6} + p_1 2^{-7} \quad (10)$$

							p7	p6	p5	p4	p3	p2	p1	p0
							q7	q6	q5	q4	q3	q2	q1	q0
						1	-q0p7	q0p6	q0p5	q0p4	q0p3	q0p2	q0p1	q0p0
						-q1p7	q1p6	q1p5	q1p4	q1p3	q1p2	q1p1	q1p0	**
						pp_7^1	pp_6^1	pp_5^1	pp_4^1	pp_3^1	pp_2^1	pp_1^1	pp_0^1	
					-q2p7	q2p6	q2p5	q2p4	q2p3	q2p2	q2p1	q2p0	**	**
					pp_7^2	pp_6^2	pp_5^2	pp_4^2	pp_3^2	pp_2^2	pp_1^2	pp_0^2		
				-q3p7	q3p6	q3p5	q3p4	q3p3	q3p2	q3p1	q3p0	**	**	**
				pp_7^3	pp_6^3	pp_5^3	pp_4^3	pp_3^3	pp_2^3	pp_1^3	pp_0^3			
			-q4p7	q4p6	q4p5	q4p4	q4p3	q4p2	q4p1	q4p0	**	**	**	**
			pp_7^4	pp_6^4	pp_5^4	pp_4^4	pp_3^4	pp_2^4	pp_1^4	pp_0^4				
		-q5p7	q5p6	q5p5	q5p4	q5p3	q5p2	q5p1	q5p0	**	**	**	**	**
		pp_7^5	pp_6^5	pp_5^5	pp_4^5	pp_3^5	pp_2^5	pp_1^5	pp_0^5					
	-q6p7	q6p6	q6p5	q6p4	q6p3	q6p2	q6p1	q6p0	**	**	**	**	**	**
	pp_7^6	pp_6^6	pp_5^6	pp_4^6	pp_3^6	pp_2^6	pp_1^6	pp_0^6						
-q7p7	q7p6	q7p5	q7p4	q7p3	q7p2	q7p1	q7p0	**	**	**	**	**	**	**
pp_7^7	pp_6^7	pp_5^7	pp_4^7	pp_3^7	pp_2^7	pp_1^7	pp_0^7							
p7q7	-p6q7	-p5q7	-p4q7	-p3q7	-p2q7	-p1q7	-p0q7	**	**	**	**	**	**	**	**
-M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Fig. 2 Eight-by-eight multiplication operation with sign extension method

Equation (10) shows that all the input bits are shifted right by 1 bit position. The sign extended bit p7 is at the MSB position of the multiplicand P, and therefore, the LSB bit p0 is eradicated. The sign-extended binary number after scaling is denoted with the same word length of 8 bits. The multiplication operation at the bit level is performed with the sign extension method. An 8 by 8 multiplication is shown in the structure of Figure 2, where ppab is the pth bit of the partial product PPi. A continuous word length is preserved by discarding the least significant bits of the right-shifted partial products, which are shown in boxes.

As the partial products are generated one step at a time, one bit sign extension is enough to associate with the sign bits for the generated partial products at each respective step.

4. Parallel Carry Ripple Array Multiplier

It is a digital circuit which executes binary multiplication of two numbers with the help of Ripple Carry Adder's (RCA) array structure to find the sum of partial products. The structure of the 8 by 8 Parallel CRAM is shown in Figure 3.

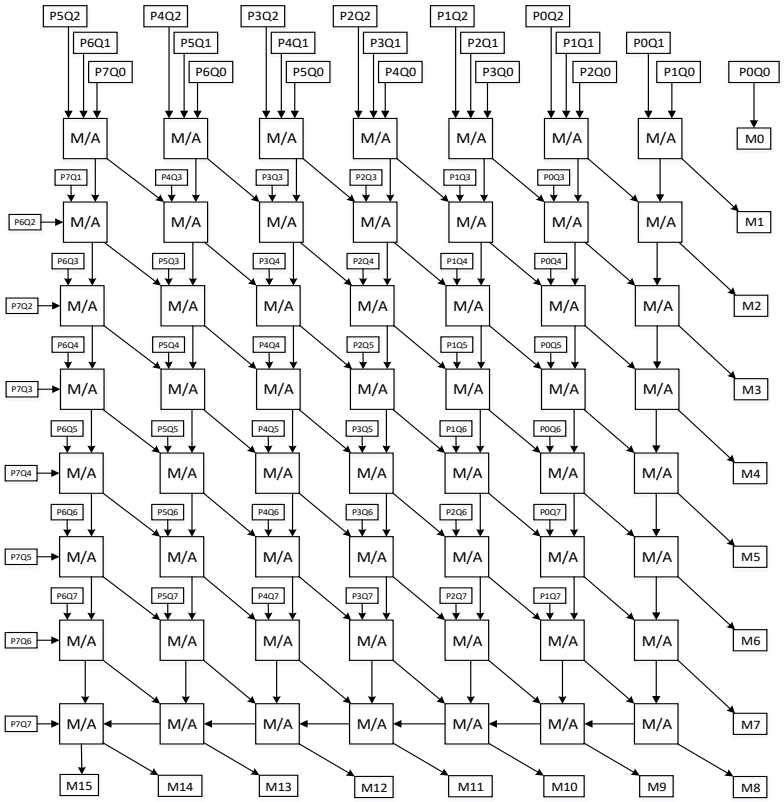

Fig. 3 Structure of the 8 by 8 Parallel CRAM

These multipliers are very useful in the field of VLSI hardware implementation, like FPGA design. The sequence of operations done by this multiplier is first partial product generation, then carry save addition, and followed by ripple carry addition.

Partial products are produced by an array of AND gates that multiply each bit of the multiplicand with each bit of the multiplier. After that, Carry-Save Adders (CSA) are used to sum partial products in parallel, which diminishes upright propagation delays. The intermediate results are then added with a single RCA to get the final result product. For an n × n bit multiplication, n rows of partial products are generated using n2 AND gates. Each row is shifted one bit to the left.

For parallel execution, n(n−1) full adders are assembled into n−1 rows of CSA. At the end, one 2n-bit RCA is used to add all rows and get the final result. The DG, which gives the relation and flow of the input to the output of the 8 by 8 Parallel CRAP, is shown in Figure 4.

5. Parallel Carry Save Array Multiplier

A Parallel CSAM is a high-speed digital multiplier circuit that is frequently used in VLSI hardware implementations. Particularly designed to perform multiplication of large binary numbers competently, which has a substantial performance benefit over a CRAM. The disadvantage of the CRA is its speed, as in this, the carry bits are broadcast with each addition, which can be overcome by the use of CSAM. The composition of the 8 by 8 Parallel CSAM is shown in the Figure 5. It generates all partial products with the help of the AND operation of each bit of the multiplier with the multiplicand simultaneously.

It practices a sequence of CSAs to sum all partial products in parallel, generating sum and carry outputs at each stage, which are saved and used in the next row. Finally, the concluding sum and carry output bits are added to produce the result product.

The plus point of CSA is that the additions performed at different bit places in the same row are now independent of each other and can be executed in parallel, which in turn increases the speed of addition and completes the multiplication operation. The final addition operation of partial sum and partial carry is executed by the output integrating adder. The DG of the parallel CSAM is shown in Figure 6.

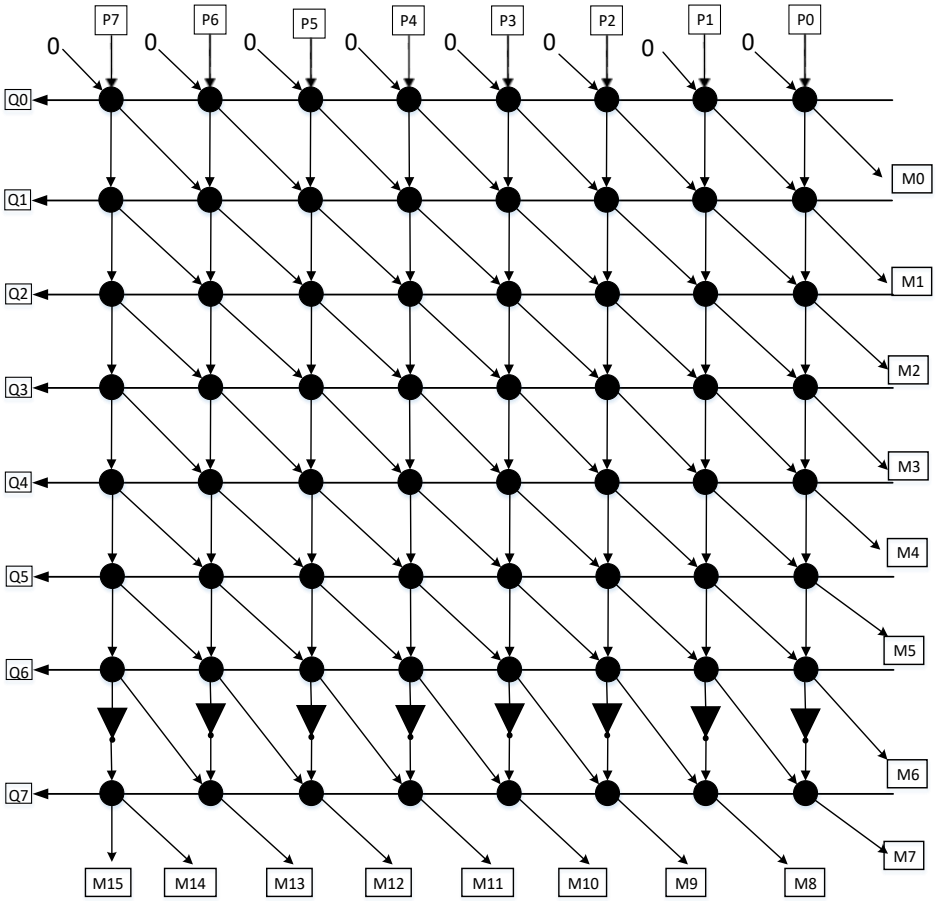

Fig. 4 Dependence graph of the 8 by 8 Parallel CRAM

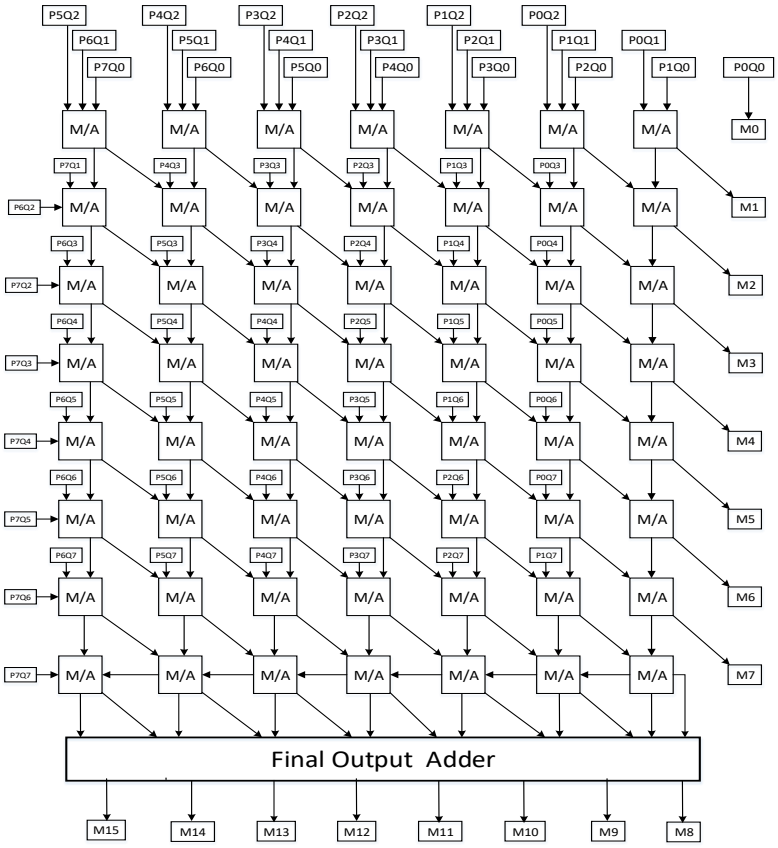

Fig. 5 Structure of the 8 by 8 Parallel CSAM

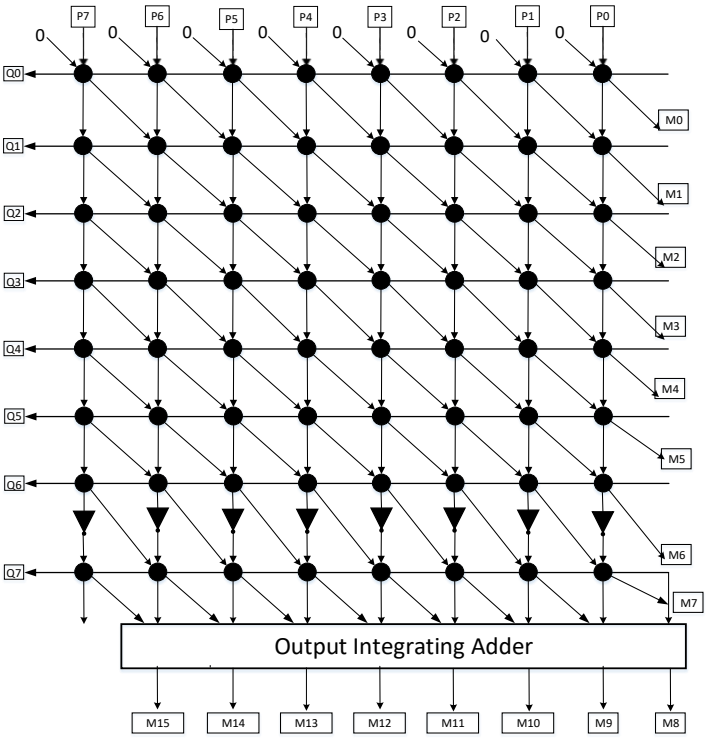

Fig. 6 Dependence graph of the 8 by 8 Parallel CSAM

6. The Baugh Wooley Array Multiplier

It is very tough to perform multiplication using the 2's complement method because of the sign bits of the multiplier and multiplicand when implemented in VLSI hardware. This problem can be solved by using a very efficient and fast multiplier known as the BW multiplier. It can be designed with CSAM as well as CRAM. A carry save BWAM is shown in Figure 7. The BW multiplier is a hardware-efficient and broadly implemented algorithm in VLSI for multiplication of two signed binary numbers for real-time digital circuits with the 2's complement method. It is crafted to handle negative numbers competently by rearranging and inverting particular partial product bits, creating the multiplication execution steady and appropriate for array multiplier designs. The algorithm is very effective and simply scalable to N × N-bit multipliers. The multiplier is ideal for low power, small area, and low delay applications when compared to other array multipliers. The formation of the hardware-efficient partial-product generated array of an N-bit BW multiplier encompasses various steps. Starting from the Most Significant Bit (MSB) of each of the first N − 1 partial-product rows, along with all bits of the last partial-product row except its MSB, are inverted. Next, '1' is added to the Nth column. Finally, the MSB of the final result is inverted. The DG of the BWAM is shown in Figure 8, which graphically presents the generated partial products, the sign inversion and sign multiplication, as well as the real-time input to output data movement. The detailed gate-level structure of the BWAM is shown in Figure 9.

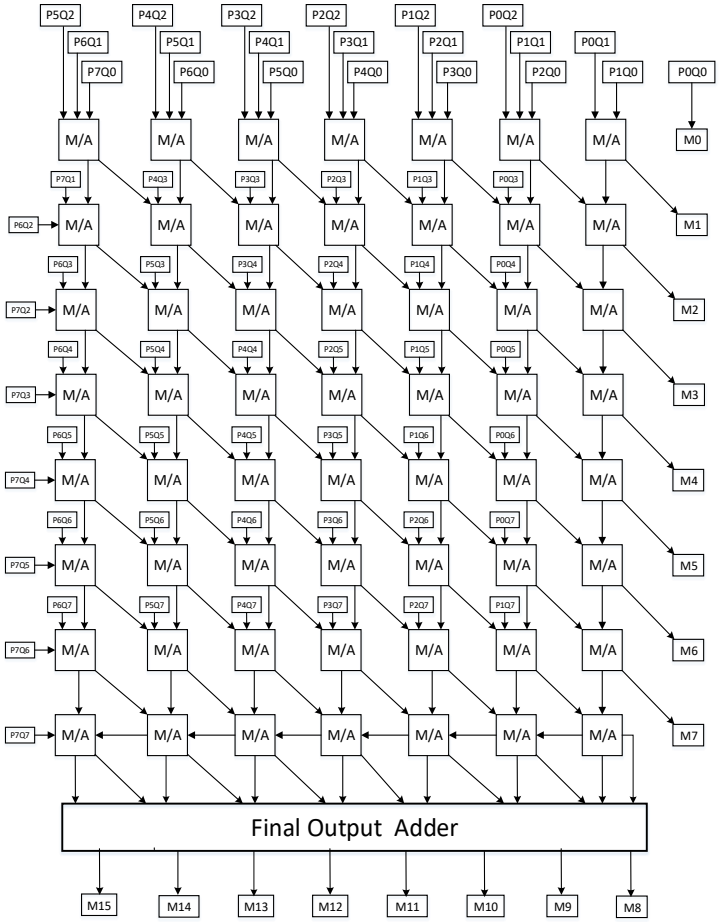

Fig. 7 Structure of the 8 by 8 BWAM

7. Mathematical Representation

For the two n-bit 2's complement numbers P and Q, shown in Equations (11) and (12):

$$P = -p_{N-1}2^{N-1} + \sum_{i=0}^{N-2} p_i 2^i \quad (11)$$

$$Q = -q_{N-1}2^{N-1} + \sum_{j=0}^{N-2} q_j 2^j \quad (12)$$

The multiplication P cross Q is as given in Equation (13):

$$M = p_{N-1}q_{N-1}2^{2N-2} + \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} p_i q_j 2^{i+j} + 2^{N-1}\sum_{i=0}^{N-2}\overline{p_i q_{N-1}}\,2^i + 2^{N-1}\sum_{j=0}^{N-2}\overline{p_{N-1}q_j}\,2^j - 2^{2N-1} + +2^N \quad (13)$$

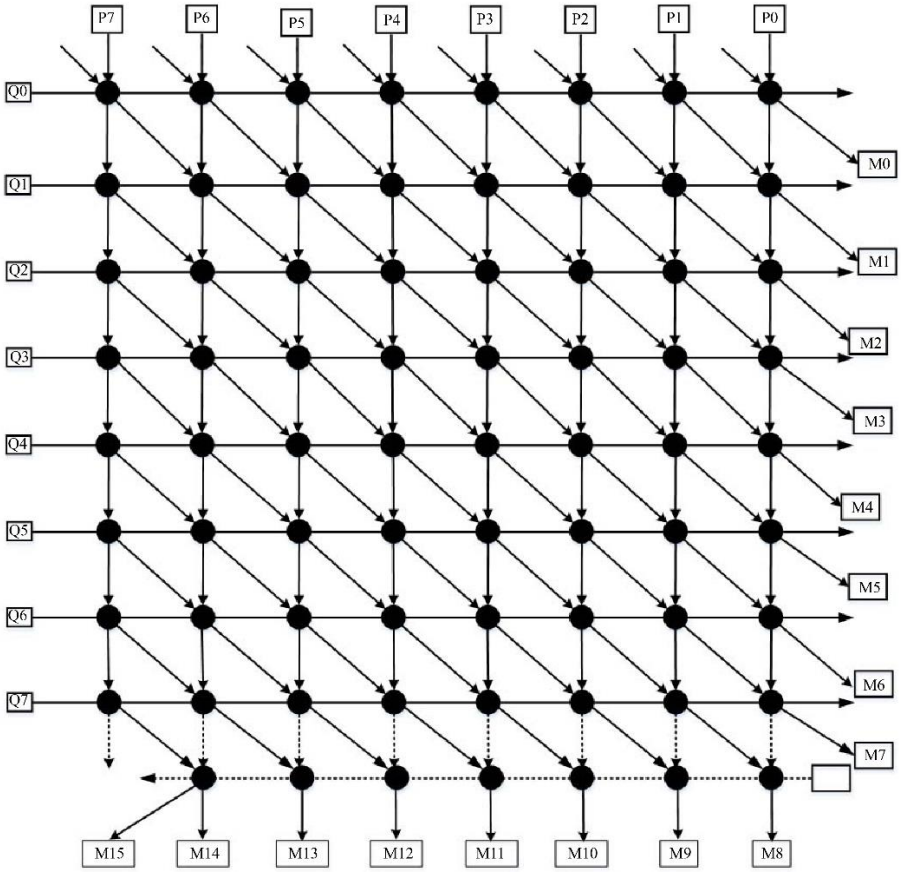

Fig. 8 Dependence graph of the 8 by 8 BWAM

The multiplication algorithm is shown in Figure 10 for an 8-by-8-bit number multiplication. The BW multiplier is designed with CSA to generate the intermediate partial products and CRA to get the final result by merging the partial products. The bit serial and digital serial with digit size 2 architectures for CRAM are shown in Figures 11 and 12, respectively. The bit serial and digital serial with digit size 2 architectures for CSAM are shown in Figures 13 and 14, respectively. The bit serial and digital serial with digit size 2 architectures for BWAM are shown in Figures 15 and 16, respectively.

8. Results and Discussion

Three different array multipliers, as carry save, carry ripple, and BW multipliers, are compared in this work. DGs for all three array multipliers are designed for 8 by 8-bit real-time data input signals. The designed DGs are modeled in Xilinx using Verilog hardware descriptive language and synthesized with the XST tool. The board employment platform is the Xilinx ML605 (XC6VLX240T1FFG1156 FPGA) embedded development kit. The total logic cells available in this device are 241,152, and the DSP slices are 768. The performance comparison between regular CSAM, CRAM, and BWAM is listed in Table 1 to examine the competence of these three different multiplier architectures using significant recital metrics such as delay, power consumption, and area utilization with the different word lengths from 4-bit to 128-bit, the table clearly shows that the delay between input and output of the signal, power consumption, and resources utilization of the FPGA by the array multiplier are less for efficient BWAMs, making it perfect for low-area and high-speed applications. CRAM keeps a balance between delay, power consumption, and area utilization. CSAMs proposed faster performance than CRAM, compromising power consumption and area utilization, particularly at larger word lengths. BWAM steadily delivers the maximum improvement in delay, with up to 55.02% at 4-bit and 38.51% at 128-bit.

It also displays impactful enhancements in power up to 43.72% and area up to 35.24% than both CSAM and CRAM. CRAM demonstrates reasonable enhancements, predominantly in power (up to 37.4% at 4-bit), but less significant than BWAM. As word length intensifications, BW's array multiplier proficiency governs across all three systems of measurement. These results highlight BWAM's appropriateness for high-speed, low-power, and area-efficient VLSI designs. The performance comparison between the designs with bit-parallel CSAM, CRAM, and BWAM is listed in Table 2. This table also shows that the BWAM is the best

performer one for a bit-parallel input signal also. BWAM is better upto 60% for input-output delay time, while 40% and 41% in terms of power consumption and area utilization, respectively. The performance comparison between the designs with digit-serial CSAM, CRAM, and BWAM is listed in Table 3. Here also, BWAM has better performance in all respects than the remaining two array multipliers. The intensive FPGA resource utilization analysis is given in Table 4. Table 5 gives a percentage-wise analysis of FPGA resource utilization for the three array multipliers. From the table, BWAM takes the minimum logic cells, the number of LUTs, the number of flipflops, DSP slices and number of IOBs. So, BWAM is more efficient in terms of overall logic development.

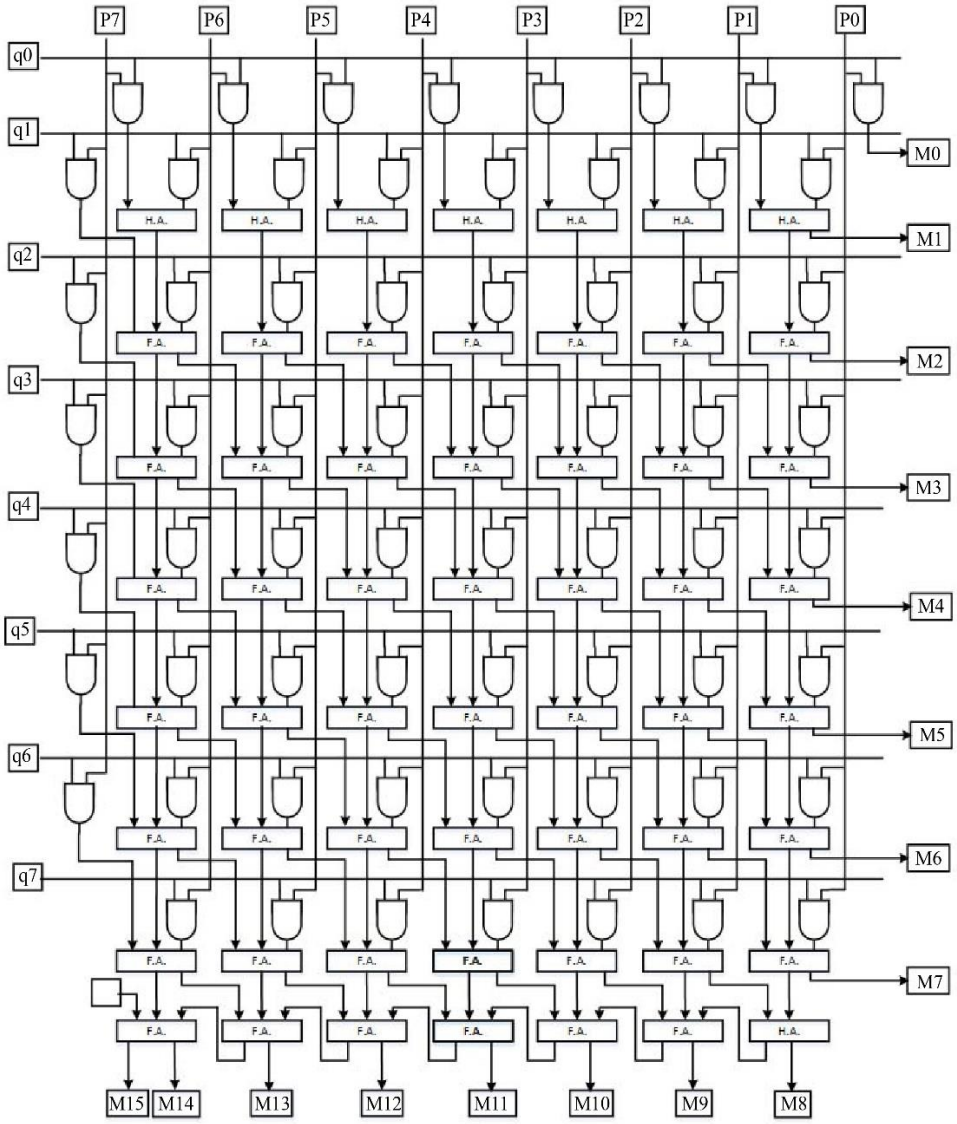

Fig. 9 Gate level structure of the BWAM

								p7	p6	p5	p4	p3	p2	p1	p0
								q7	q6	q5	q4	q3	q2	q1	q0
							1	$\overline{p_7q_0}$	p6q0	p5q0	p4q0	p3q0	p2q0	p1q0	p0q0
							$\overline{p_7q_1}$	p6q1	p5q1	p4q1	p3q1	p2q1	p1q1	p0q1	**
						$\overline{p_7q_2}$	p6q2	p5q2	p4q2	p3q2	p2q2	p1q2	p0q2	**	**
					$\overline{p_7q_3}$	p6q3	p5q3	p4q3	p3q3	p2q3	p1q3	p0q3	**	**	**
				$\overline{p_7q_4}$	p6q4	p5q4	p4q4	p3q4	p2q4	p1q4	p0q4	**	**	**	**
			$\overline{p_7q_5}$	p6q5	p5q5	p4q5	p3q5	p2q5	p1q5	p0q5	**	**	**	**	**
		$\overline{p_7q_6}$	p6q6	p5q6	p4q6	p3q6	p2q6	p1q6	p0q6	**	**	**	**	**	**
	p7q7	$\overline{p_6q_7}$	$\overline{p_5q_7}$	$\overline{p_4q_7}$	$\overline{p_3q_7}$	$\overline{p_2q_7}$	$\overline{p_1q_7}$	$\overline{p_0q_7}$	**	**	**	**	**	**	**
$\overline{M15}$	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0

Fig. 10 The multiplication table for the BW multiplier

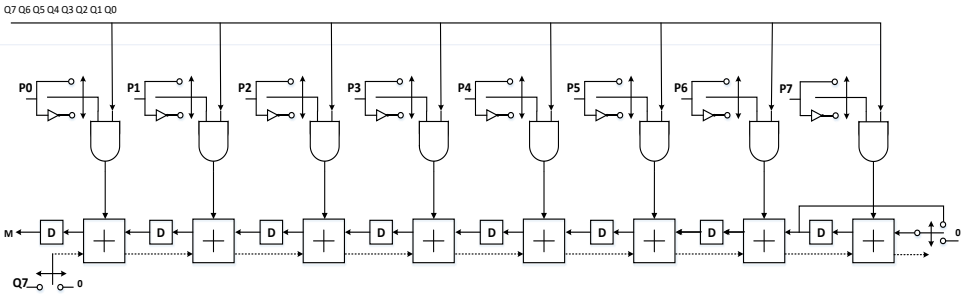

Fig. 11 The bit serial architecture for CRAM

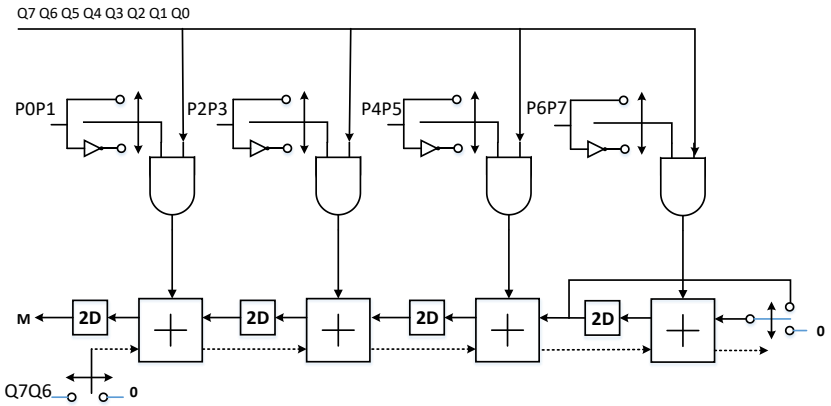

Fig. 12 The digit serial architecture for CRAM

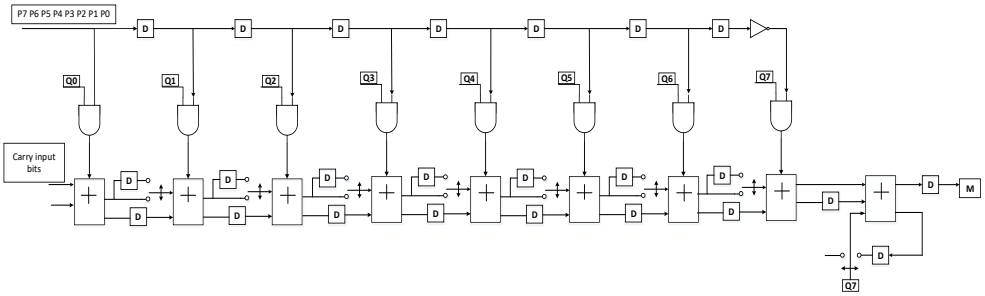

Fig.13 The bit serial architecture for CSAM

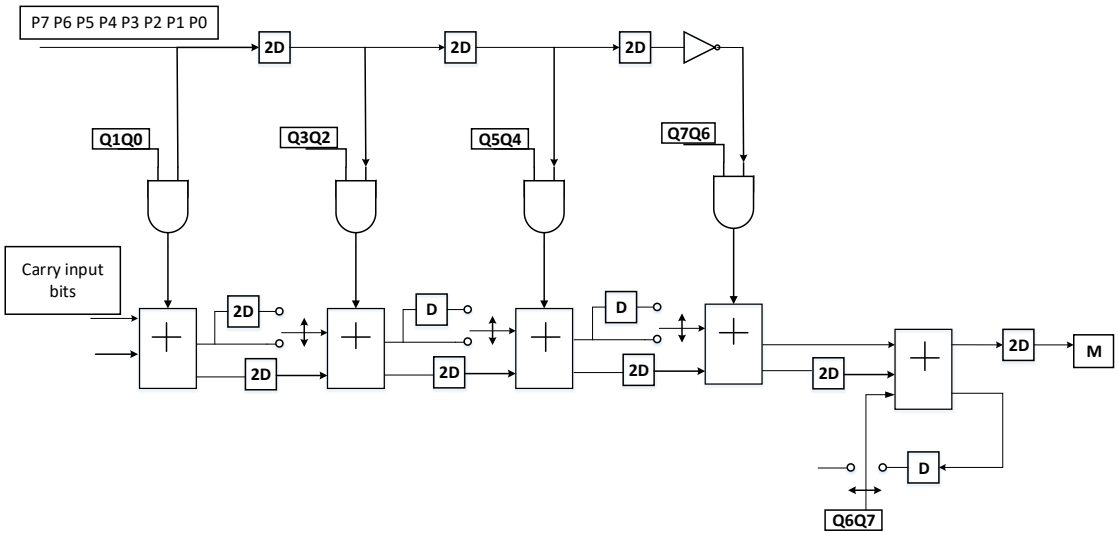

Fig. 14 The digital serial architecture for CSAM

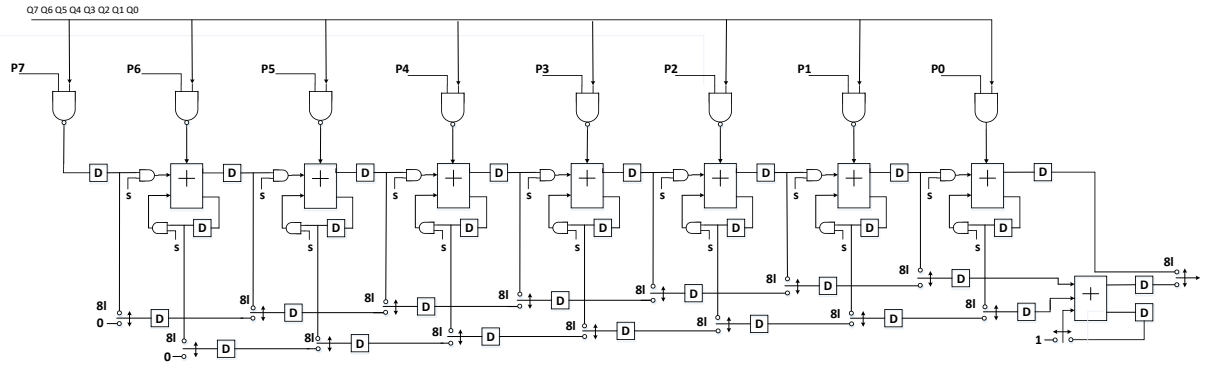

Fig. 15 The bit serial architecture for Baugh-Wooley array multiplier

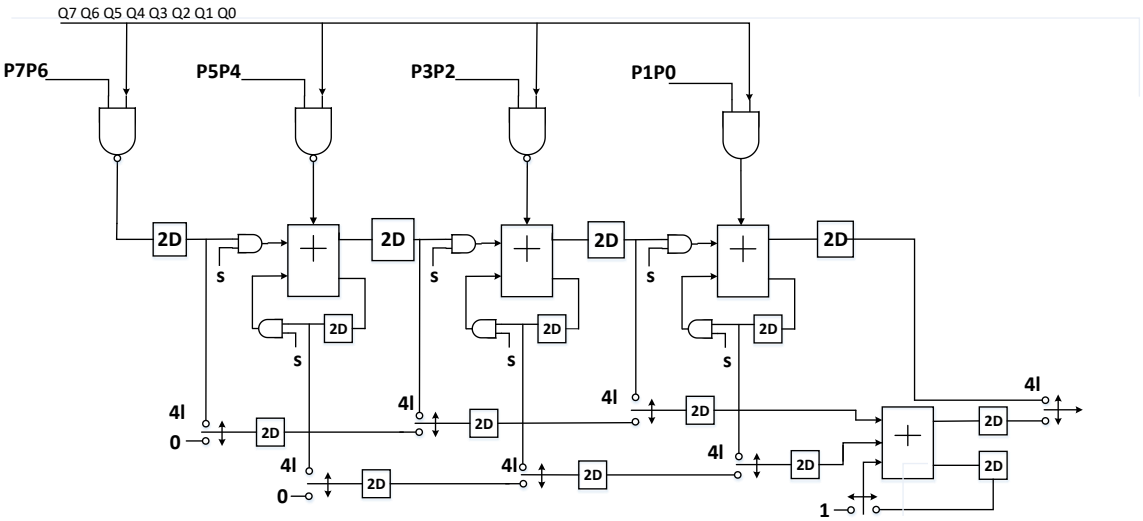

Fig. 16 The digital serial architecture for Baugh-Wooly array multiplier

Table 1. Performance comparison between regular carry save, carry ripple, and BW array multipliers

	Carry save array multiplier.			Carry ripple array multiplier			Baugh Wooley array multipliers		
Word Length	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**
4-bit	5.98	32.32	10,898	4.88	20.23	9,506	2.69	18.19	7,058
8-bit	8.06	40.23	22,565	7.23	32.96	19,505	5.87	25.55	16,045
16-bit	10.56	52.23	34,203	8.56	42.02	28,898	7.02	35.55	23,580
32-bit	14.02	66.76	42,232	10.25	57.89	36,987	9.03	50.01	29,023
64-bit	21.56	80.26	51,233	18.36	71.26	45,210	15.64	62.62	38,027
128-bit	32.85	92.63	59,102	26.56	88.88	50,257	20.20	80.08	43,580

Table 2. Performance comparison between the designed bit-parallel carry save, carry ripple, and BW array multipliers

	Carry save array multiplier.			Carry ripple array multiplier.			Baugh Wooley array multipliers		
Word Length	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**
4-bit	3.23	26.32	8,897	2.08	20.23	7,500	1.28	16.23	5,989
8-bit	5.69	32.02	14,505	4.23	30.96	12,501	3.03	23.36	9,998
16-bit	8.56	40.23	26,255	6.50	36.02	20,808	4.59	32.63	15,597
32-bit	11.32	51.76	37,239	8.29	45.89	28,907	6.56	41.02	22,189
64-bit	16.06	63.29	45,238	13.36	58.26	38,212	11.03	54.23	32,520
128-bit	22.65	71.23	52,107	18.58	67.88	46,251	15.89	60.02	40,253

Table 3. Performance comparison between the designed bit-serial carry save, carry ripple, and BW array multipliers

	Carry save array multiplier.			Carry ripple array multiplier.			Baugh Wooley array multipliers		
Word Length	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**	**Delay (ns)**	**Power (µW)**	**Area (µm2)**
4-bit	3.98	22.32	10,897	2.38	19.23	9,024	1.28	16.23	8,982
8-bit	5.56	28.33	16,501	4.29	25.06	15,236	3.03	23.36	13,028
16-bit	9.56	38.03	28,005	6.98	34.92	26,525	4.59	32.02	22,586
32-bit	12.32	49.02	35,565	9.98	45.09	31,898	6.56	48.98	25,560
64-bit	15.69	55.40	49,285	13.96	50.26	46,412	11.03	56.23	42,586
128-bit	22.99	66.23	56,258	19.02	60.08	52,626	15.89	78.92	47,025

Table 4. Intensive FPGA resources utilization analysis

	Available resources	**Carry save array multiplier(128-bits)**	**Carry ripple array multiplier (128 bits)**	**Baugh Wooley array multipliers (128 bits)**
Number of Logic cells	241,152	4568	4258	4001
Number of LUTs used	150,720	1507	1328	1175
Number of Flipflops used	301,440	3489	3196	2520
Number of DSP slices	768	16	16	16
Number of IOBs	600	257	208	176

Table 5. Percentage-wise analysis of FPGA resources utilization for the three array multipliers

	% of Utilization		
	Carry save array multiplier(128-bits)	**Carry ripple array multiplier (128 bits)**	**Baugh Wooley array multipliers (128 bits)**
Number of Logic cells	1.89%	1.77%	1.66%
Number of LUTs used	1.00%	0.88%	0.78%
Number of Flipflops used	1.16%	1.06%	0.84%
Number of DSP slices	2.08%	2.08%	2.08%
Number of IOBs	42.83%	34.67%	29.33%

9. Conclusion

The design of the advanced architecture for bit-level and digit-level systolic array multiplication is presented in this paper. The emphasis is on the high-performance and area-efficient array multiplier designs. A comprehensive study of the performance of the designed array multiplier is done. The employment of digit-level architecture results in a noteworthy decrease in interconnection overhead and critical path delay, while bit-level architecture preserves modularity and accuracy of the systems. The incorporation of techniques like Carry-save, carry ripple, and BW algorithms additionally improves computational competence and appropriateness for signed and high meticulous procedures. Further simulation and synthesis results of the designed architectures show that they outperform the old-styled array multiplier circuit designs in terms of latency, throughput, and adaptability, particularly when implemented in FPGAs, reconfigureurable devices. These conclusions support the real-world feasibility of systolic arrays in real-time DSP applications. Finally, the obtained architecture provides meaningful contributions to the development of parallel arithmetic systems and offers a strong substance for future research in adaptive, low-power, and application-specific array multiplier designs. The proposed systolic array multiplier architectures are validated mainly on an FPGA platform; henceforth, the stated delay, power, and area metrics may differ under ASIC implementations due to technology-dependent effects. Therefore, potential extensions of this work should focus on ASIC-based validation in adaptive and variable-precision systolic architectures.

Conflicts of Interest

No

Acknowledgments

Nil

References

[1] A.V. Oppenheim, R.W. Schafer, and C.K. Yuen, "Digital Signal Processing," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 8, no. 2, 1978. [CrossRef] [Publisher Link]

[2] John G. Proakis, and Dimitris G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications, 4/e*, Pearson Education, 2007. [Google Scholar] [Publisher Link]

[3] Keshab K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, John Wiley and Sons Publishing, 1999. [Google Scholar] [Publisher Link]

[4] B. Ramkumar, and Harish M. Kittur, "Faster and Energy-Efficient Signed Multipliers," *VLSI Design*, vol. 2013, no. 1, pp. 1-12, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[5] Wen Yan, Miloš D. Ercegovac, and He Chen, "An Energy-Efficient Multiplier with Fully Overlapped Partial Products Reduction and Final Addition," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 1954-1963, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[6] B. Hareesh, John Moses C, and M.V.V. Prasad Kantipudi, "VLSI Architectures of Booth Multiplication Algorithms--A Review," *International Journal of Computing and Digital Systems*, vol. 11, no. 1, pp. 266-275, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Avishek Sinha Roy, Hardik Agrawal, and Anindya Sundar Dhar, "ACBAM-Accuracy-configurable Sign Inclusive Broken Array Booth Multiplier Design," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 4, pp. 2072-2078, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[8] P. Kishore et al., "Implementation of Braun and Baugh-Wooley Multipliers using QCA," *2023 2nd International Conference for Innovation in Technology (INOCON)*, Bangalore, India, pp. 1-4, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[9] Noureddine Chabini, and Rachid Beguenane, "FPGA-based 8x8 Bits Signed Multipliers using LUTs," *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Regina, SK, Canada, pp. 366-370, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Mitul Sudhirkumar Nagar et al., "High-Speed Energy-Efficient Fixed-Point Signed Multipliers for FPGA-based DSP Applications," *IEEE Embedded Systems Letters*, vol. 16, no. 4, pp. 417-420, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[11] R.V. Nithyashree et al., "Design and Implementation of Array Multiplier using 2-Bit Accurate and Approximate Adder," *2024 IEEE 6th PhD Colloquium on Emerging Domain Innovation and Technology for Society (PhD EDITS)*, Bangalore, India, pp. 1-2, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[12] Wai Leong Pang et al., "Optimizing Multiplier Performance Through Verilog Hardware Description Language Design," *2024 Multimedia University Engineering Conference (MECON)*, Cyberjaya, Malaysia, pp. 1-6, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] Yeshudas Muttu et al., "Implementation and Performance Analysis of 8-bit Digital Parallel Array Multipliers," *2024 International Conference on Intelligent Computing and Sustainable Innovations in Technology (IC-SIT)*, Bhubaneswar, India, pp. 1-5, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] S. Saranya et al., "Design of Performance Improved 4-Bit Baugh Wooley Multiplier using LUT," *2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, Chennai, India, pp. 1-4, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[15] Shervin Vakili, "A Cost-Effective Baugh-Wooley Approximate Multiplier for FPGA-based Machine Learning Computing," *2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, Abu Dhabi, United Arab Emirates, pp. 367-371, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[16] Kapil Juneja, Anupriya Jangra, and Dhiraj Khurana, "Design of a Quaternary Component and Wallace Tree Integrated Baugh-Wooley Multiplier," *International Journal of Networked and Distributed Computing*, vol. 13, no. 1, pp. 1-14, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[17] L. Kavana, and G.N. Keshava Murthy, "Power and Area-Efficient Multiplier Architectures: A Comparative Study of Array, Dadda, Booth, Wallace Tree, and Vedic Multipliers," *2025 3rd International Conference on Smart Systems for Applications in Electrical Sciences (ICSSES)*, Tumakuru, India, pp. 1-6, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[18] Asep Muhamad Awaludin et al., "A High-Performance ECC Processor Over Curve448 based on a Novel Variant of the Karatsuba Formula for Asymmetric Digit Multiplier," *IEEE Access*, vol. 10, pp. 67470-67481, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Qiang Wu et al., "A Novel Multiplier Circuit for PFC Controllers with Low Total Harmonic Distortion and High Power Factor," *IEEE Transactions on Power Electronics*, vol. 40, no. 11, pp. 16105-16110, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[20] Yu-Yu Lin et al., "A Serial-Parallel Mixing-Mode Matrix-Vector Multiplication Architecture for Large-Scale In-Storage Computing with Ultrahigh Parallelism in nand Flash Memories," *IEEE Transactions on Electron Devices*, vol. 72, no. 9, pp. 4837-4843, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[21] Yujun Xie, and Yuan Liu, "An Efficient LUT6-based Montgomery Modular Multiplication using Radix-16 Booth Method," *IEEE Transactions on Computers*, vol. 74, no. 9, pp. 3223-3237, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[22] Zixuan Zhu et al., "Bit-Sparsity Aware Acceleration with Compact CSD Code on Generic Matrix Multiplication," *IEEE Transactions on Computers*, vol. 74, no. 2, pp. 414-426, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[23] Pramod Kumar Meher, "Systolic and Non-Systolic Scalable Modular Designs of Finite Field Multipliers for Reed-Solomon Codec," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 6, pp. 747-757, 2009. [CrossRef] [Google Scholar] [Publisher Link]