iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** XI **Month of publication:** November 2017

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue XI November 2017- Available at www.ijraset.com

A Survey on Area Efficient Low Power High Speed Multipliers

R Kalaimathi[1], R Senthil Ganesh[2]
[1]*M.E VLSI Design,* [2]*Assistant Professor Institute of Engineering, Coimbatore, Tamilnadu*

Abstract: ***Digital arithmetic operations are the most important in the design of Digital Signal Processing (DSP) and application specific systems. Multiplication is the most basic arithmetic operations and the multipliers plays significant role in various arithmetic operations in digital processing applications such as digital filtering, digital communications and spectral analysis. A fast and accurate operation of a digital system is greatly influenced by the performance of different multipliers using different methodology. Emerging trends in low power made attention towards the search of low power, high speed and area efficient multipliers architecture. This paper presents the study on some important multiplier techniques.***
Keywords: High Speed, Low power, Area efficient, Multipliers, VLSI

I. INTRODUCTION

Computer industry creates every time faster-computing components due to the advances in Integrated Circuits (ICs) technology. Every time better chip fabrication and sophisticated computing design techniques are available to be used which create architectural innovation with an efficient use of technology improvements. However, with several improvement techniques of integrated circuits, parameters including latency, power consumption, silicon die and temperature of the chip are continuing to be major challenging of integrated circuit designers.

The design of high-speed, area efficient and low-power VLSI architecture needs efficient arithmetic processing units. In the design of systems using digital signal processing and other applications multiplier is an important basic building block. Statistical data shows that more than 70% of the instructions in microprocessor and DSP algorithms perform multiplication operations. So, these operations require more execution time. Therefore, there is the need of high speed multipliers. Many researchers are continuously trying to design multiplier with high speed, low power consumption, regular structure, such that it occupies less area for compact VLSI implementation. Many algorithms are proposed for efficient multiplication operation. Every algorithm offerings its own advantages and having trade-off between themselves by means of their speed, area, power consumption and circuit complexity [4].

This paper discussed the various multipliers such as Array, Booth Multiplier, Wallace Tree, Dadda multiplier.

II. MULTIPLIERS

Essential design targets of multiplier include high speed, low power consumption, regularity of layout and hence less area. In microprocessors, multiplication operation is performed in a variety of forms in hardware and software depending on the cost and transistor budget allocated for this particular operation. The basic multiplication algorithm which operates on positive n-bit long integers X and Y resulting in the product P which is 2n bit long given as [10],

$$P = XY = X * \sum_{i=0}^{n-1} yi\ ri \quad (1)$$

This expression indicates that the multiplication process is performed by summing n terms of a partial product P_i. The i^{th} term of P_i is obtained by arithmetic left shift of X value for I positions and multiplication by single digit y_i. For the binary radix (r=2), y_i is 0 or 1 and multiplication by the digit y_i is very simple to perform. The addition of 'n' terms can be performed by, sequentially adding partial products using an adder 'n' times or can be performed at once, by passing the partial products through a network of adders.

The straight forward way to implement a multiplication is based on an iterative adder-accumulator. This multiplier is called serial multiplier [4] and shown in Fig. 1. This solution is quite slow because the final result is available after 'n' clock cycles, where 'n' is the size of the operands. The serial multipliers are used where area and power are of utmost importance and increased delay can be tolerated. The multiplier and multiplicand inputs are arranged such that they synchronized with the circuit behavior. Depending on the length of the multiplier and multiplicand, the inputs can be presented at different rates. Two clock signals are used, one for data and other for reset operation. The drawback of this multiplier algorithm is not suitable for large values of multiplier and multiplicand.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue XI November 2017- Available at www.ijraset.com

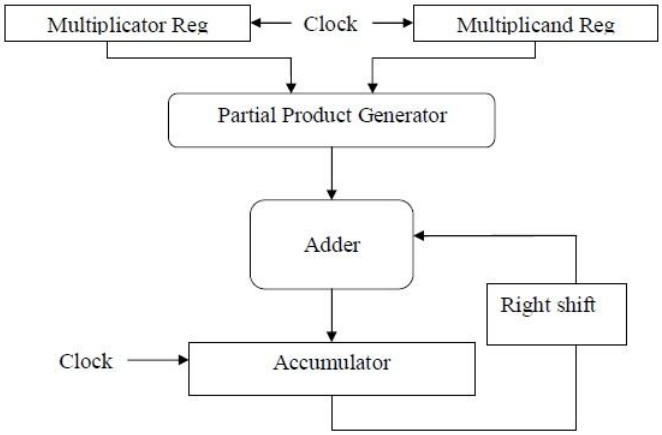


Fig. 1 Serial Multiplier

A faster version of iterative multiplier should add partial products at once. This could be achieved by unfolding the iterative multiplier and yielding a combinational circuit. The combinational circuit consists of several partial product generators with several adders that operate in parallel [6]. This multiplier is called parallel multiplier and shown in the Fig. 2.

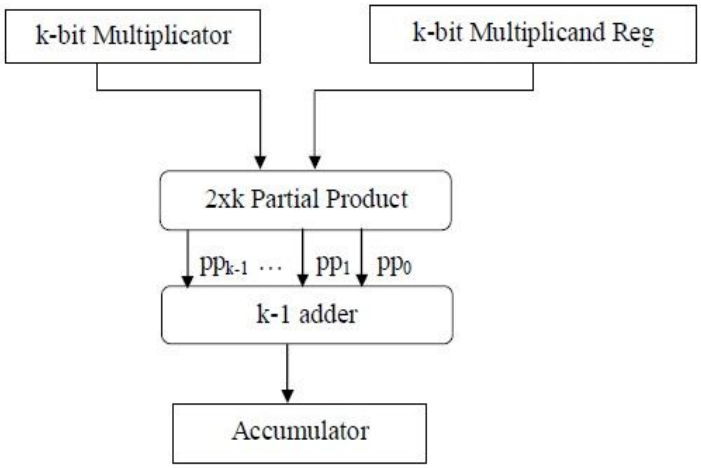


Fig. 2 Parallel Multiplier

The multiplication process has three main steps are partial product generation, partial product reduction and final addition of the sum and carry [9]. Many techniques are available to perform binary multiplication and the algorithm selection is based upon factors such as design complexity, throughput and latency. More efficient parallel approach uses some sort of array or tree of full adders to sum partial products. Array multiplier, Booth Multiplier, Wallace Tree and Dadda multipliers are some of the standard approaches to have hardware implementation of binary multiplier which are suitable for VLSI implementation at CMOS level.

A. Array Multiplier

Array multiplier is a form of combinational multipliers. Multiplication of binary numbers can be performed by using a combinational circuit that forms the product bit all at once and achieves fast way of multiplication. The reason is, delay is the only due to the signals to propagate through the gates that forms the multiplication array. It can be implemented by mapping the manual multiplication into hardware. The partial products are accumulated by an array of adder circuits. An n x n array multiplier requires n(n-1) adders and n^2 AND gates. It is a regular structure multiplier and performs the multiplication process in traditional way. Because of regular structure it is easy of layout and design. Add and shift algorithm is employed in an array multiplier. Implementation of the array multiplier is simple but it requires larger area with considerable delay than any other method [7]. In Fig. 3, PPx(y) denotes partial product, P denotes product, x denotes row number and y denotes bit number.

Array Multiplier consumes more power and requires larger number of gates. Due to larger gate components area is also increased. It is a fast multiplier but hardware complexity is high [7]. The array multiplier performance is slower compared to the Booth, Wallace and Dadda multipliers.

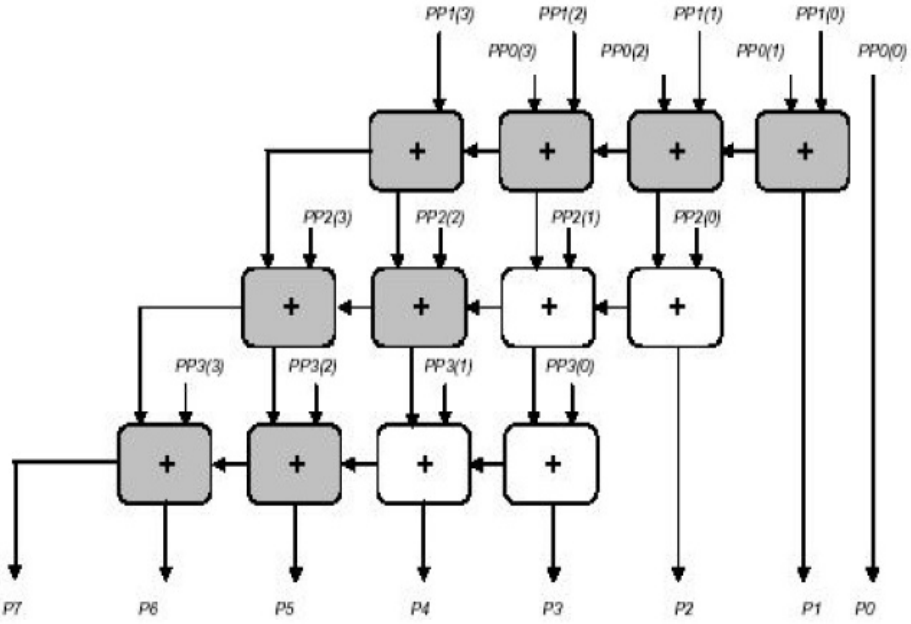


Fig.3 4 x 4 Array Multiplier

B. Booth Multiplier

Booth's algorithm is widely used to implement the software or hardware multipliers because it reduces the number of partial products generation. It can be used for both signed-magnitude numbers and 2's complement numbers with no need for a correction term. Modified version of the Booth algorithm is proposed by Mac Sorley, in which three bits are scanned. The Booth-Mac Sorley algorithm, also called as the Modified Booth algorithm or the Booth algorithm can be generalized to any radix. Radix-4 Booth encoding technique has the advantage of reducing the number of partial products by one half regardless of inputs. The recoding is performed within two steps such as encoding and selection. The encoding process scans the triplet of multiplier bits and defines the respective operation to be performed on the multiplicand. This method is an application of a sign-digit representation in radix-4. For example, a 3-bit recoding would require the following set of digits to be multiplied by the multiplicand: 0, +1, -1, +2, (-2). Booth multiplication preserves the sign of the result and also allows for smaller, faster multiplication [1].

The following methods are followed in Booth's multiplication process,

Right Shift Circulant (RSC) is nothing but shifting the bit position. In a string of binary numbers, shift to the right one bit position and takes the last bit in the string and append it to the beginning of the string. For example: 101101, after right shift circulant the result is 110110.

Right Shift Arithmetic (RSA) is where you add 2 binary numbers together and shift the result to the right 1 bit position. Then shift all bits right and put the first bit of the result at the beginning. For example: 0101 + 0011 = 1000 = 11000.

Booth Multiplier can be used in three distinct modes such as radix-2, radix-4, radix-8 etc. Of this, Radix-4 Booth's algorithm is most widely because of number of partial products is reduced to n/2 [1].

1) Radix-2 Booth Multiplier: In Booth's algorithm, the most important step is the booth recoding. Recode the multiplier Y, to a recoded value Z and the multiplicand X remain unchanged. Booth recoding, replace string of 1's by 0's. In Booth recoding, each digit of the multiplier can assume negative as well as positive and zero values. There is a special notation, called signed digit encoding to express the signed digits. In SD encoding +1 and 0 are expressed as 1 and 0, but -1 is expressed as 2's complement of 1. For example, the value of string of five 1's, 11111 = 25 -1 = 100001 = 32 -1 = 31. Hence if this number were to be used as multiplier in a multiplication, instead of five additions, only one addition and one subtraction is required. The recoding procedures are [1]

2) Move from LSB to MSB, replace each 0 digit of the original number with a digit 0 in the recoded number until digit 1 is encountered

3) When digit 1 is encountered, insert digit 1 at that position in the recoded number, and skip over any succeeding 1's until digit 0 is encountered.

4) Replace that digit 0 with digit 1 and continue. he algorithm is expressed in tabular form in Table 1. Pairs of numbers are Y_{i-1}, Y_i and recoded digit Z_i. Fig. 4 shows the example for bit pairing as per Booth recoding [11].

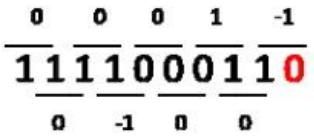


Fig. 4 Example for Bit Pairing as per Radix-2 Booth Recoding

TABLE 1
RADIX-2 BOOTH RECODING

Y_i	Y_{i-1}	Z_i	Multiplier Value	Comments
0	0	0	0	String of 0's
0	1	1	+1	End of String 1's
1	0	1	-1	Begin of string 1's
1	1	0	0	String of 1's

The drawbacks of radix-2 booth recoding are,
Difficult to design parallel multipliers because the number of add or subtract operation becomes variable.
Algorithm becomes inefficient if there are isolated 1's.

5) *Radix-4 Modified Booth Multiplier:* Using radix-4 booth multiplier, number of partial products can be reduced by half. MBE generates at most (N/2) partial products, where N is the number of bits. It compares three bits at a time with overlapping technique. The algorithm is as follows [1],

the LSB with one zero.
If n is even pad the MSB with two zeros else pad it with 1 zero.
Divide the multiplier into overlapping group of 3-bits.
Determine the partial products scale factor from the recoding table.
Compute the multiplicand multiples which is nothing but partial products.
The most common modified booth's recoding scheme is used with the digit set {-2, -1, 0, 1, 2} and shown in the Table 2. In radix-4, grouping starts from the LSB, and the first block uses only two bits of the multiplier and assumes a reference bit 0 for the third bit and shown in Fig. 5 [11].

Fig. 5 Example for Bit Pairing as per Radix-4 Modified Booth Recoding

TABLE 2 RADIX-4 MODIFIED BOOTH RECODING

Bits of Multiplier B			Encoding operation on Multiplicand A	Partial Product
y_{i+1}	y_i	y_{i-1}		
0	0	0	0	M * 0
0	0	1	+B	M * 1
0	1	0	+B	M * 1
0	1	1	+2B	M * 2

1	0	0	-2B	M * -2
1	0	1	-B	M * -1
1	1	0	-B	M * -1
1	1	1	0	M * 0

C. Wallace Tree Multiplier

C.S. Wallace (1964) proposed a fast technique to perform multiplication. Wallace tree multiplier offers faster performance for large operands. Unlike an array multiplier, the partial product matrix for a tree multiplier is arranged in a tree-like format, thereby reducing both the critical path and the number of adder cells. Wallace tree is an efficient hardware implementation of a digital circuit. Wallace trees are irregular structure. It consists of three stages. In the first stage, partial product matrix is formed. In the second stage, this partial product matrix is reduced to a height of two. In the third stage, these two rows are combined using adder [12]. Fig. 6 shows the tree structure of the partial product matrix. In this type of multiplier, partial products are reduced as soon as possible. But the number of half adders and full adders required for the partial product reduction is high. It uses more number of gates for partial product reduction.

The underlying principle of the Wallace tree multiplier is to achieve partial product accumulation by successively reducing the number of bits of information in each column using full adders or half adders. The full adder is known as (3:2) compressor and half adder as (2:2) compressor [5]. The Wallace tree consists of numerous levels of such column compression structures. Fig. 7 shows the dot diagram of 8-bit Wallace tree multiplier. To reduce the partial product matrix as quickly into the final product, this type of multiplier uses as much hardware as possible [3].

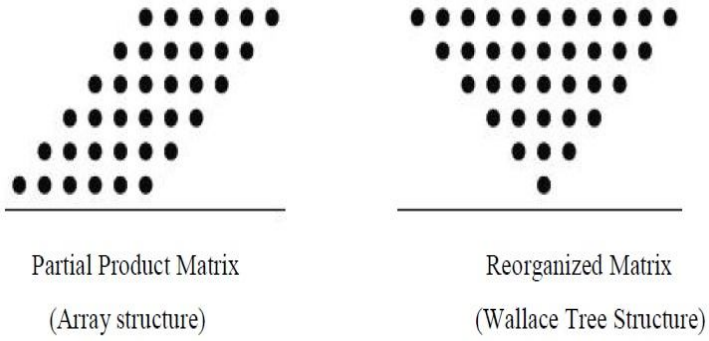


Fig. 6 Tree Structure of Partial Product Matrix

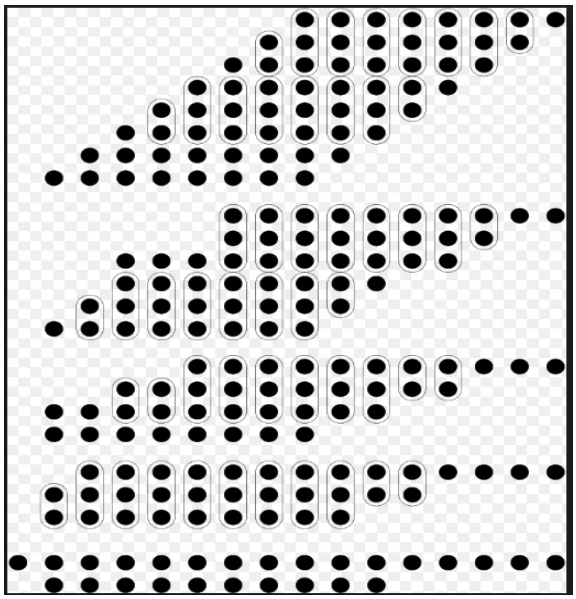


Fig. 7 Dot Diagram of 8-bit Wallace Tree Multiplier

D. Dadda Multiplier

Dada methodology is used to minimize the number of adder stages and therefore the delay can be greatly minimized. In the partial product addition stage, the partial product matrix is reduced to a height of two. In this stage, the Dadda methodology uses parallel (n, m) counters. A (n, m) parallel counter has n inputs and produce m outputs which gives a binary count of the number of 1's present at the inputs. Dadda multipliers use a minimal number of (3, 2) and (2, 2) counters at each level during the compression to achieve the required reduction [2].

The dot diagram is shown in Fig. 8 shows this algorithm implemented for an 8 by 8 multiplier. In this, four reduction levels are required with the matrix heights of 6, 4, 3, and 2. In the Fig. 8, two dots joined by a diagonal line indicate that these dots are the outputs from a (3, 2) counter. Similarly, two dots joined by a crossed diagonal line indicate that these two dots are the outputs from (2, 2) counter. It requires totally 35 (3, 2) counters and 7 (2, 2) counters to reduce the partial products. The number of (3, 2) and (2, 2) counters required for a Dadda multiplier depends on N, the number of bits of the operands [2]. Dadda multipliers require fewer (3, 2) and (2, 2) counters during the compression stage than the corresponding Wallace multipliers.

$$(3, 2) \text{ counters} = N^2 - 4N + 3 \quad (2)$$

$$(2, 2) \text{ counters} = N - 1 \quad (3)$$

The reduction procedure is in [2],

Let $d_1 = 2$ and $d_{j+1} = [1.5 * dj]$, d_j is the height of the matrix for the j^{th} stage. Repeat until the largest j^{th} stage is reached in which the original N height matrix contains at least one column which has more than d_j dots.

In the j^{th} stage from the end, place (3, 2) and (2, 2) counters as required to achieve a reduced matrix. Only columns with more than dj dots or which will have more than dj dots as they receive carries from less significant (3, 2) and (2, 2) counters are reduced.

Let j = j -1 and repeat step (b) until a matrix with a height of two is generated. This should occur when j = 1.

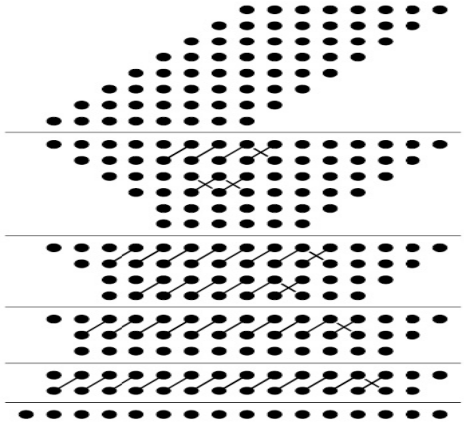


Fig. 8 Dot Diagram of 8-bit Dadda Multiplier

III. COMPARISON RESULTS

The comparative study of the above discussed multipliers is listed in the Table 3. The area, delay and power is calculated using Xilinx FPGA Spartan 2E, FG676 – XC2S600E. The respective values of area, delay, power, speed, PDP and EDP for 8-bit is given in the Table 4 and the parameter comparison chart diagram is also shown in the Fig.9 for Array, Modified Booth Encoding (MBE) multiplier, Wallace and Dadda multipliers.

Speed is calculated from delay, Speed = (1/Delay) [8].

PDP is calculated from power and delay, unit is 'fWs' where f denotes 'femto'. PDP = (Power * Delay) [8].

EDP is calculated from PDP and delay, unit is 'yWs^2' where y denotes 'yocto'. EDP = (PDP * Delay) [8].

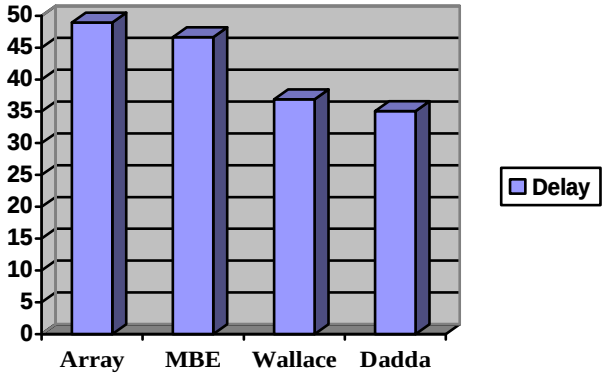
TABLE 3

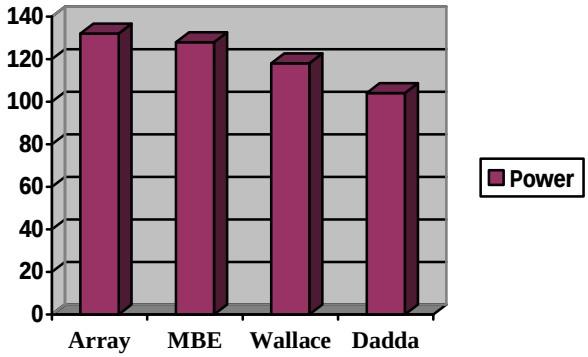
COMPARISON OF MULTIPLIERS

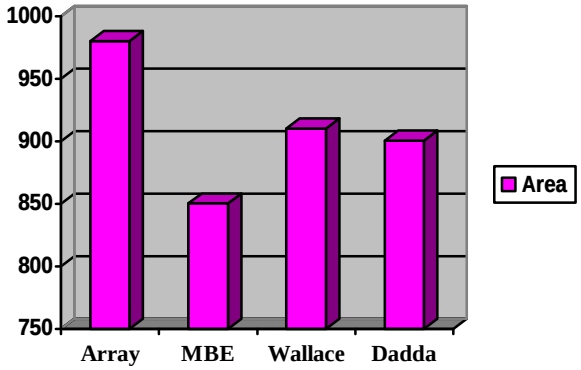
Type of Multiplier	Delay	Structure	Complexity	Area
Array	Linear	Regular	Low	High
MBE	Non-Linear	Regular	Medium	Low
Wallace	Logarithmic	Irregular	High	High
Dadda	Logarithmic	Irregular	High	High

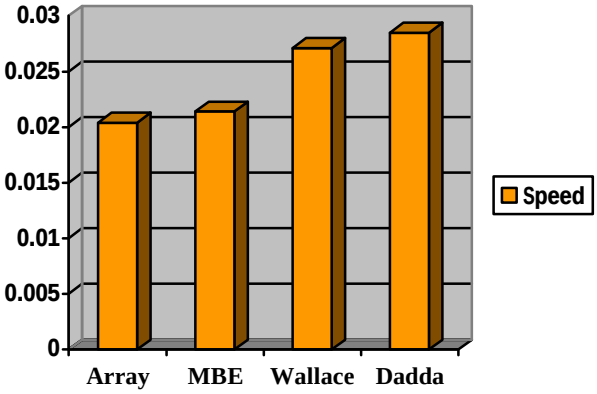
TABLE 4
PARAMETERS VALUE COMPARISON OF MULTIPLIERS

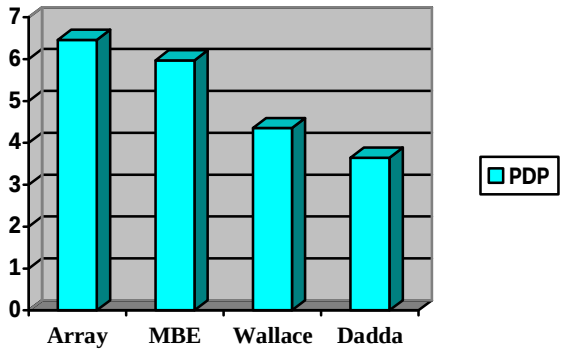
Types of Multiplier (8-bit)	Delay (ns)	Power (mW)	Area(μm^2)	Speed (GHz)	PDP (fWs)	EDP (yWs^2)
Array	48.952	132	979.96	0.0204	6.461	316.28
MBE	46.635	128	850.23	0.0214	5.969	278.36
Wallace	36.893	118	910.04	0.0271	4.353	160.60
Dadda	35.040	104	900.67	0.0285	3.644	127.69











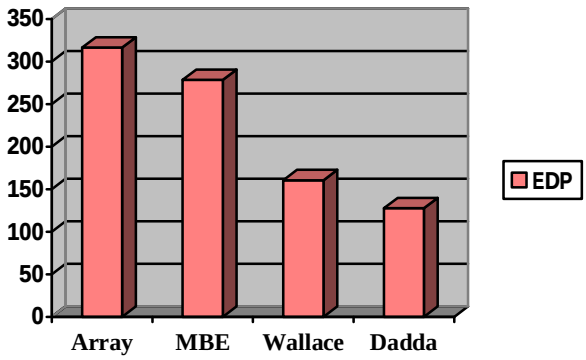


Fig. 9 Parameters Comparison Chart Diagrams for Multipliers

IV. CONCLUSIONS

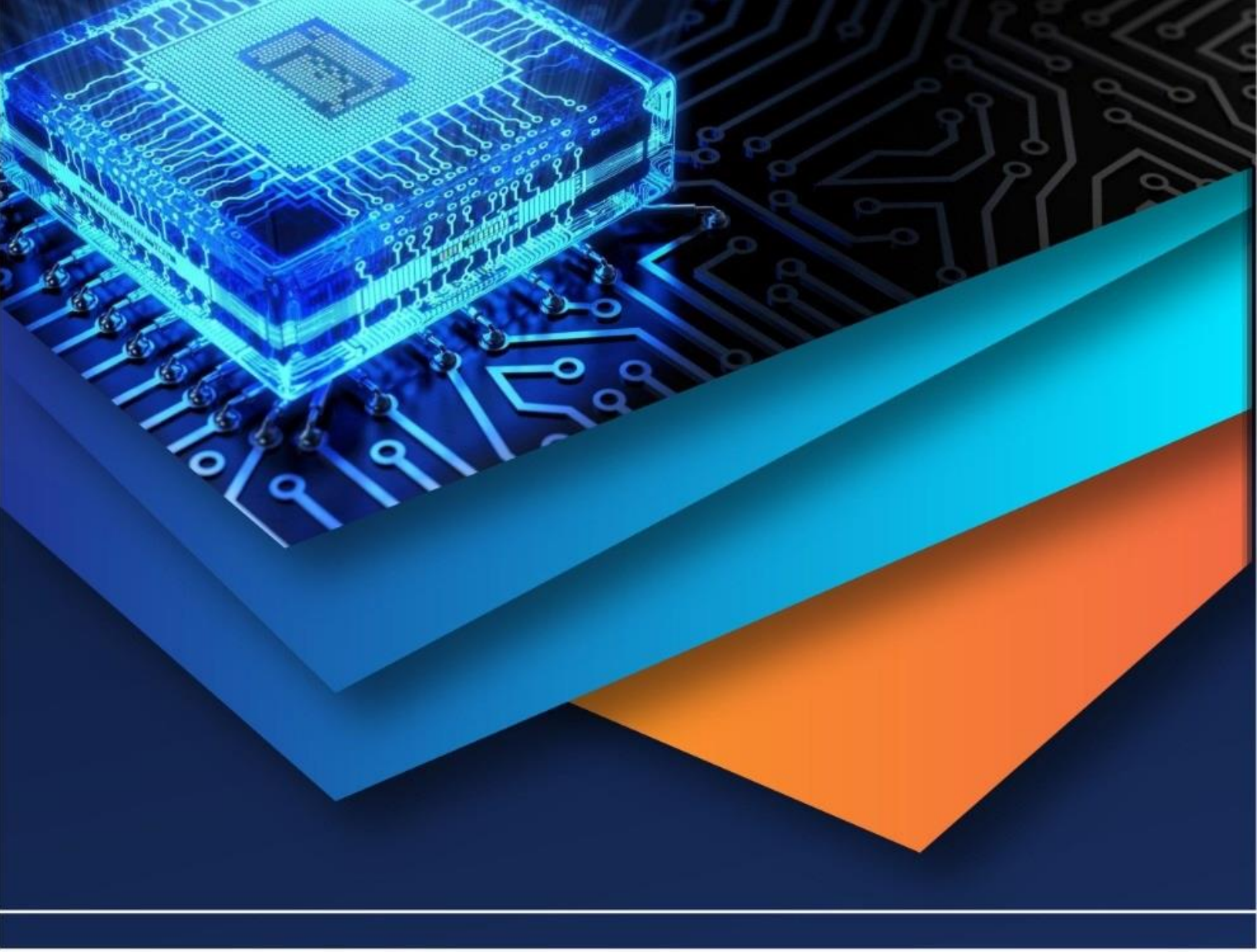
In this paper, various multiplier architectures are explained briefly. The Booth multiplier requires less area compared to other types of multipliers. For better power and delay, Dadda multiplier is more efficient than the other multiplier methods. To achieve better optimization, area efficient, delay and low power consumption, the combination of Booth and Dadda multiplier gives better result.

V. ACKNOWLEDGMENT

I first thank my husband and my parents for their support and encouragement given to me throughout this journal work. I express my whole hearted thanks to Mr. R. Senthil Ganesh, Assistant Professor, for his valuable personal interest and guidance in completing this journal work. I avail this opportunity to express my profound sense of sincere and deep gratitude to who were responsible for the knowledge gained throughout the course of this journal work.

REFERENCES

[1] Booth. A.D, "A Signed Binary Multiplication Technique", Qarterly J. Mechan. Appl.Math, Vol. IV, 1951.
[2] Dadda. L, “Some Schemes for Parallel Multipliers”, Alta Frequenza, Vol.34, p.349-356, March 1965.
[3] Fadavi-Ardekani. J, "M x N Booth Encoded Multiplier Generator Using optimized Wallace Trees" IEEE trans. on VLSI Systems, Vol. 1, No.2, Jun 1993.
[4] Hwang. K, "Computer Arithmetic: Principles, Architecture and Design", John Wiley and Sons, 1979.
[5] Naveen Kr. Gahlan, Prabhat Shukla and Jasbir Kaur, “Implementation of Wallace Tree Multiplier Using Compressor”, ISSN: 2229-6093, Vol 3 (3), pp. 1194-1199.
[6] Oklobdzija. V.G, Villeger. D, Liu. S.S, “A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using and Alghoritmic Approach”, IEEE Transaction on Computers, Vol. 45, No 3, March 1996.
[7] Pezaris. S.D, "A 40 ns 17-bit array multiplier", IEEE Trans. on Computers., Vol. C-20, pp. 442-447, Apr. 1971.
[8] Saravanan, S. V, Senthil Ganesh R (2016), “Design of Low Power and Area Efficient Dynamic Comparator”, International Journal of Printing, Packaging & Allied Sciences, vol.04, no.02, pp.1464-1470.
[9] Stelling. P, Martel. C, Oklobdzija. V.G, Ravi. R, “Optimal Circuits for Parallel Multipliers”, IEEE Transaction on Computers, Vol. 47, No.3, pp. 273-285, March, 1998.
[10] Stenzel. W.J, "A Compact High Speed Parallel Multiplication Scheme", IEEE Transaction on Computers, Vol. C-26, pp. 948-957, Feb 1977.
[11] Sukhmeet Kaur, Suman and Manpreet Signh Manna, “Implementation of Modified Booth Algorithm (Radix 4) and its Comparison with Booth Algorithm (Radix-2)”. ISSN 2231-1297, Volume 3, pp. 683-690, Nov 2013.
[12] Wallace. C.S, “A Suggestion for a Fast Multiplier”, IEEE Transactions on Electronic Computers, EC-13, p.14-17, 1964.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY