iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** V **Month of publication:** May 2024

DOI: https://doi.org/10.22214/ijraset.2024.61704

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: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

Energy-Efficient Approximate Multiplier with Flexible Precision

Ayushi Vinodia
ME (Student Department Of Electronics and Telecommunication, Jabalpur Engineering College),482011, Jabalpur M.P.

Abstract: ***The method that can be utilized to increase accuracy and decrease energy use is approximate multiplication. A key component of many error-tolerant applications is multiplication. Approximate multipliers are increasingly utilized in energy-efficient computing for applications tolerant of inaccuracy. Apart from multiplier performance, determining the appropriate approximate multiplier is challenging due to considerations of area and delay. Therefore, selecting the type of approximate full adder (FA) becomes a crucial decision-making factor. These adders are employed for summing partial products in multipliers. This study presents the design and evaluation of an approximate multiplier employing four distinct approximate adders. The design undergoes simulation and synthesis using Xilinx and Model Sim. Compared to previously proposed approximate multipliers, the proposed circuits demonstrate substantial reductions in area, time delay, and power consumption. According to experimental data, the area, latency, and average power consumption of the suggested adjustable approximate multiplier can be lowered by 10%, 34.96 ns, and 300 mW when contrasted to the Wallace tree multiplier.***

Keywords: Comparative Study, Error Metrics, Approximate Multiplier, Power-Efficiency, Approximate Computing, Circuit Characteristics.

I. INTRODUCTION.

Multipliers play a vital role in various applications like digital signal processing (DSP), computer vision, multimedia processing, image recognition, and artificial intelligence. These applications often require numerous multiplications, leading to significant power usage, especially challenging for mobile devices. Many studies propose reducing power consumption in multiplier circuits to address this issue. One approach is to approximate multiplication, especially for applications where some level of error tolerance is acceptable, such as those related to human perception. Because human senses have limitations, precise computation results may not always be necessary. The approximation multipliers reduce cell space, time delay, and power consumption at the expense of precision [1].

The prevalence of modern computing systems, whether they are pervasive, portable, embedded, or mobile, has sparked a rising need for ultra-low power consumption, compact size, and superior performance. A developing paradigm called approximate computing provides a way to accomplish these goals while compromising arithmetic precision. Many domains, including multimedia and big data analysis, can tolerate a certain level of computational inaccuracies, making them prime candidates for employing approximate computing techniques. focuses primarily on the design of approximate arithmetic units in hardware, including adders and multipliers, at various abstraction levels, including transistor, gate, RTL (Register Transfer Level), and application [2].

An approximate multiplier in VLSI design usually has 3 stages.

1) Partial-product generation-input operands are decomposed into smaller sub-multiplications, which are combined to form the full product.
2) Accumulation-The input operands are combined to form the full product.
3) Final The output is produced depending upon the input operands.

The majority of approximation multipliers shorten the carry chains with configurable errors. When the operand's bit width rises, the algorithms utilized in the designs produce bigger magnitude errors for smaller numbers.

Power area and delay efficiency of approximate multiplier design-Removal partial product generation and accumulation for lower order input operands.

The computational method can therefore be applied to situations where an approximate answer is adequate for the intended purpose, but it also delivers a potentially incorrect result instead of a guaranteed one.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

II. METHODS FOR DIGITAL ARITHMETIC [RELATED WORK]

In practically all applications involving digital signal processing, multiplication is a fundamental arithmetic operation. Hardware multipliers are necessary for DSP systems to effectively implement DSP algorithms. The digital processing units' speed is directly impacted by the multiplier's "speed [3–5].

The literature has a large number of fast multipliers that can be utilized to" create an effective digital oscillator [6,7-10]. The addition is the fundamental function of a multiplication algorithm, regardless of the multiplicity technique. We examine the most typical adder architectures in this section. Multi-operand addition methods are utilized to better improve multiplication addition "processes [11].

A. Carry Save Adder (3:2 Adder/3:2 Counter)

The Carry Save Adder (CSA) belongs to the category of redundant number system adders. It is employed for adding multiple binary vectors, such as x, y, and z, to produce 2 binary vectors—S and C—where x + y + z = S + C. This addition operation can be executed in constant time (O(1)) as the carry bits are preserved in the C vector, eliminating the need for carry propagation. The final binary vectors (S + C) are added using a conventional number system adder, such as the carry look-ahead adder, to yield the result of the multi-operand addition. The FA can be viewed as a 1-bit CSA, as" Fig. 2 illustrates.

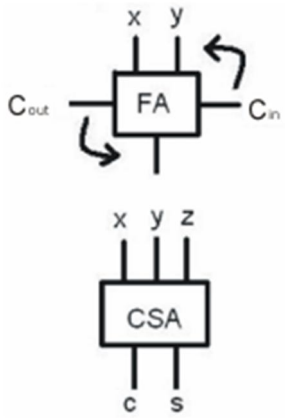


Fig 1. Full adder to carry save adder.

B. 4:2 Compressor

The "4-2 carry-save module" "was suggested in [12]. In comparison to the conventional CSA Adder, 3:2 Adder, it comprises a mixture of FA cells that are coupled to produce the output signal quicker. 5 bits, or input signals, are compressed into 3 bits, or output signals, by the structure; however, four of the inputs originate from the same bit position and one from the bit position before it (referred to as carry-in, Cin). One bit in the subsequent bit position (referred to as perform, Cout) and 2 output bits in the same bit location constitute the output of such a 4:2 compressor. The basic construction of a 4:2 compressor is depicted in Figure 3. Due to its ability to compress 4 partial product bits into two bits and output one carry bit, this" compressor is frequently utilized in multiplier units. At every level, the compressor's design cuts the quantity of partial product bits in half. Three series XOR gates' speed represents the speed of a 4:2 compressor of this kind [7,11].

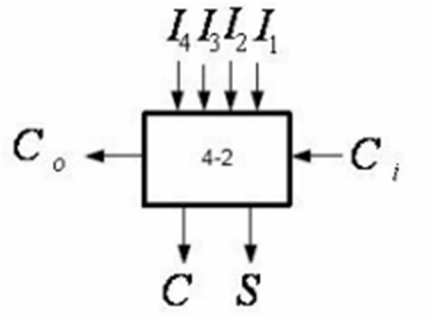


Fig. 2. 4:2 Compressor

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

C. Multiplier Design

The three primary steps of digital arithmetic multiplication techniques are final addition, partial product summation, and partial product generation. The several second stage implementations are explained in detail in this section. Because it significantly affects the multiplier's overall speed, the second phase is essential. In this case, a multiplier with little surface area and low latency can be created by using multi-operand addition algorithms such as the Wallace tree, Compressor tree, and linear tree. But adding "utilizing one of the fastest conventional adders—the carry look-ahead adder—occurs in the last step [5-12].

D. Array of Full Adder (FA)

The partial products are added in the multiplier employing an array of full adders. The unsigned integers multiplier (X) and multiplicand (A) have the same word length (n bits). As (1) illustrates, the product needs 2n bits to be represented (we call such multiplications n-by-n multiplications).

$$P = X \times A$$

$$P = \sum_{i=0}^{n-1} x_i \cdot 2^i \times \sum_{j=0}^{n-1} a_j \cdot 2^j$$

$$P = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(x_i \cdot a_j \right) \cdot 2^{i+j}$$

E. Linear Carry-Save Adder (CSA)

Tree The multiplier uses the modified Booth's recoding algorithm, which was suggested in [14], to produce partial products. As demonstrated in Figure 4, the produced partial products are added" together utilizing a Linear Carry-Save Adder Tree (LCSAT). We produce two components of the finished product at each step. The Look-ahead Carry In the last step, adder (CLA) is employed to produce the remaining parts of the outcome [7].

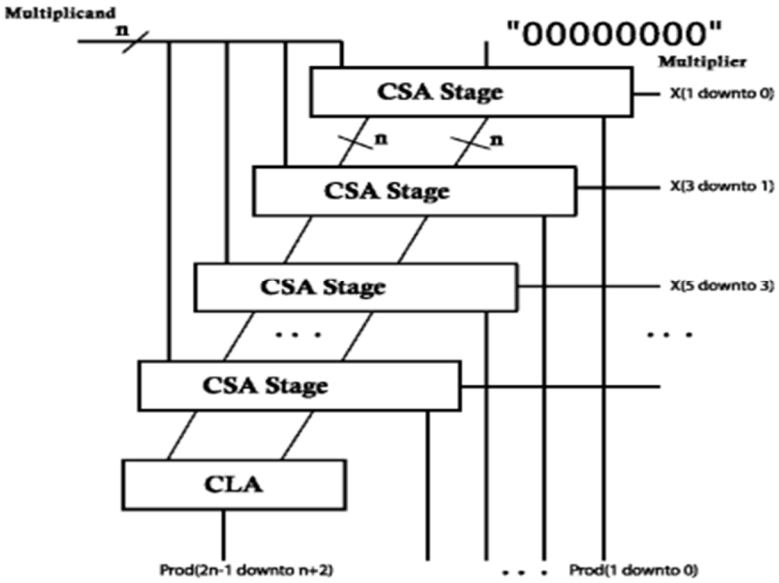


Fig.3. Multiplier based on the Linear Carry-Save adder

F. Wallace Tree Multiplier

The Wallace tree multiplier is a hardware circuit designed to efficiently multiply binary numbers. It was developed by Australian computer scientist Chris Wallace in 1964. Unlike regular structures, Wallace tree multipliers reduce the number of partial products and utilize carry select adders for adding these partial products. While the structure of a Wallace tree is irregular, it effectively adds partial products in parallel, improving efficiency. This method involves a three-step process for multiplying integers and can be implemented using two different designs: one employing half adders and full adders, and the other using a more advanced carry skip adder (CSA).

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Initially, partial products are formed by multiplying each bit of the multiplicand with each bit of the multiplier. Half adders are used for columns with two bits, while full adders are used for columns with three bits. Single bits in a column are passed to the next stage without alteration. This reduction process continues in successive stages until only 2 rows remain. These remaining rows are then added in the final stage, resulting in an 8-bit output. The compressor's efficiency directly affects the multiplier's speed, area, and power consumption. Using a 3:2 compressor in a Wallace tree speeds up the overall multiplier operation.

In a Wallace tree employing a 3:2 compressor, three partial products are directed to a one-bit full adder, forming a three-input Wallace tree circuit. The output is then fed into the next stage's full adder, positioned one bit higher. The arrangement of carry-save adders (CSAs) in a Wallace tree resembles a tree structure, with full adders forming the nodes. CSAs improve worst-case path delay by saving carry vectors to combine with sums later. This method effectively increases the maximum frequency and occupies minimal area. However, because of its irregular structure, laying out the circuit becomes more complex, despite its high operational speed.

				A_3	A_2	A_1	A_0
				B_3	B_2	B_1	B_0
				P_{03}	P_{02}	P_{01}	P_{00}
			P_{13}	P_{12}	P_{11}	P_{10}	
		P_{23}	P_{22}	P_{21}	P_{20}		
	P_{33}	P_{32}	P_{31}	P_{30}			
	P_{33}	P_{23}	S_{04}	S_{03}	S_{02}	S_{01}	P_{00}
		C_{04}	C_{03}	C_{02}	C_{01}		
	P_{33}	P_{32}	P_{31}	P_{30}			
	P_{33}	S_{14}	S_{13}	S_{12}	S_{11}	S_{01}	P_{00}
	C_{14}	C_{13}	C_{12}	C_{11}			
C_{out}	S_{24}	S_{23}	S_{22}	S_{21}	S_{11}	S_{01}	P_{00}

Fig.4. 4x4 Schematic design of Wallace Tree Multiplier.

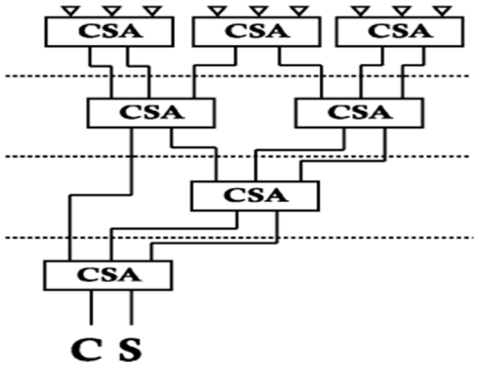


Fig.5. Wallace Tree Multiplier 3:2 Compressors.

G. Array Multiplier

An array of full as well as half adders is used in an array multiplier, a type of digital combinational circuit, to multiply two binary values. It is similar to a conventional structure multiplier. This multiplier uses the common add and shift method to achieve multiplication.[13] The multiplicand is multiplied by each bit of the multiplier to get the partial product. After that, the partial product is relocated by its bit order before being appended at the very end. The number of multiplier bits and the number of partial products that are produced are equal. The construction of an array multiplier is quite regular and methodical, and as word length increases, so does its delay. Dual array tree structure was used in a novel design that was introduced by Min C. Park et al. (1993) [14]. introduced a novel dual array tree structure design. which involves a traditional array multiplier's dual partial product array split from a single partial product plane. An array multiplier requires the maximum number of components and uses more power. The space also grows since a greater number of gates are needed. As a result, the array multiplier has subpar area performance [15]. Although it is a fast multiplier, the high number of gates causes an increase in hardware complexity. An illustration of a traditional array multiplier using CSA.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

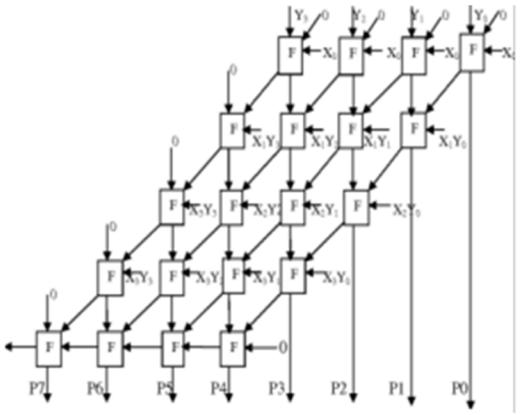


Fig.6. Array Multiplier

H. Booth Multiplier

The algorithm credited to Andrew Donald Booth in 1950 was developed during his research on crystallography. Booth utilized reception desk calculators, which offered faster-shifting capabilities than addition, prompting him to devise this algorithm to enhance computational speed. Notably, it accommodates both signed and unsigned numbers, as depicted in the flowchart provided in Figure 4. This algorithm is particularly influential for multiplying signed numbers and handling positive and negative values uniformly. It multiplies signed binary values using the 2's complement representation, where each multiplier bit produces a multiple of the multiplicand that needs to be added to the partial product. Consequently, the multiplier's delay is mostly depends on the number of additions required. Strategies aimed at reducing the number of additions can significantly enhance performance. Widely adopted in chip design, the Booth algorithm offers considerable improvements over traditional long multiplication methods. The schematic flowchart of the Booth algorithm is illustrated step by step in Fig 8.

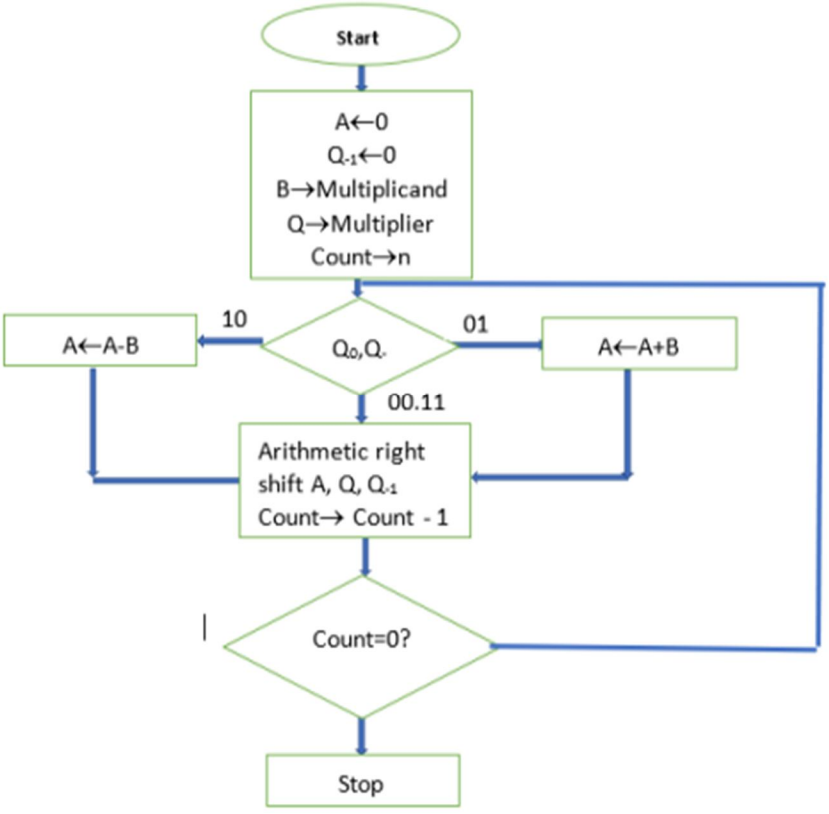


Fig.7. Flow Chart of Booth Multiplier.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue V May 2024- Available at www.ijraset.com

III. PROPOSED APPROXIMATE MULTIPLIER

A. Architecture Of Proposed Multiplier

The proposed approximate multiplier is made for n-bit multiplication in which we are taking 32-bit input and producing 64-bit output figure represents the architecture of the proposed approximate multiplier in which we are providing x(31:0) that is 32-bit input at x and y(31:0) that is 32-bit input at y and getting p(63:0) that is 64-bit output at p. The fig represents the internal structure of the proposed approximate multiplier.

B. Parameter Table For Proposed System With Results

Area Consumption is less	19200 (10% Utilization)
Power Consumption should be less	300mW
Time should be less	34.96nm

Area consumption is less than 19200 only 10% utilization, Power consumption of the proposed multiplier is also < 300 mW and the time delay is also less than 34.96nm of our proposed approximate multiplier.

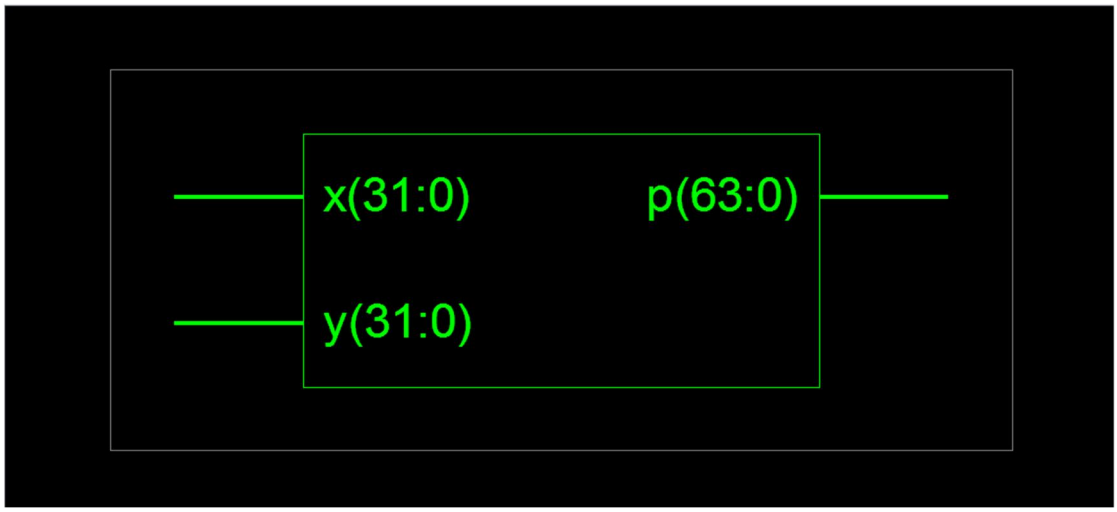


Fig.8. Architecture of proposed approximate multiplier.

In the proposed multiplier architecture we are taking 32-bit input and we are getting 64-bit input
X=32-bit, Y=32 bit input and P=64 bit input

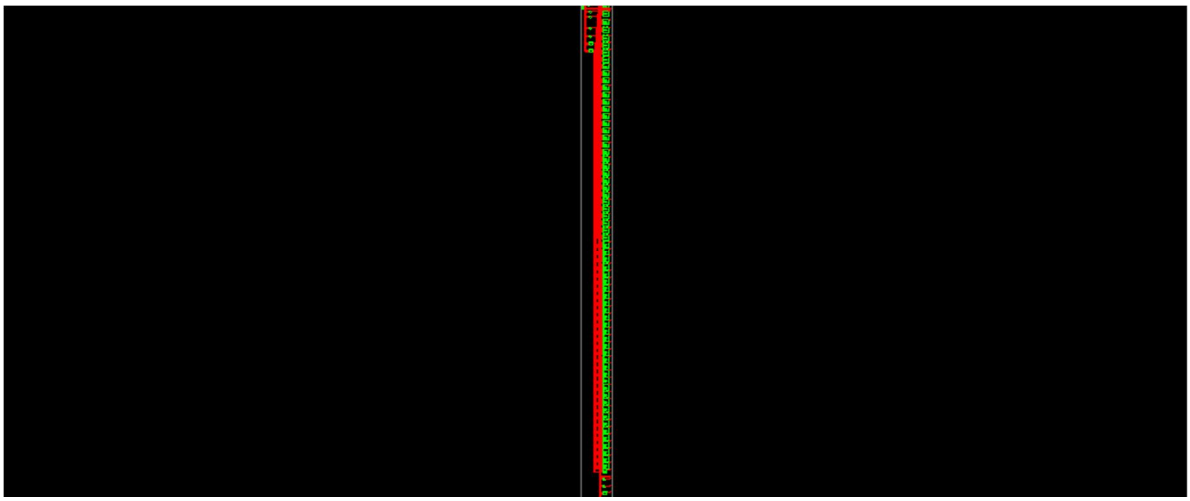


Fig.9. This is the internal structure of the proposed multiplier

©IJRASET: All Rights are Reserved

IV. RESULTS AND DISCUSSION

The multiplication process of the proposed approximate multiplier is explained with the help of Modlesim. The proposed multiplier has two inputs x (0:31) i.e. 32-bit and y (0:31) i.e. 32-bit and one output p(0:63) i.e. 64-bit. The time in which the waveform generates is 100ps. Fig 1. Represents the time in which the waveform generates.

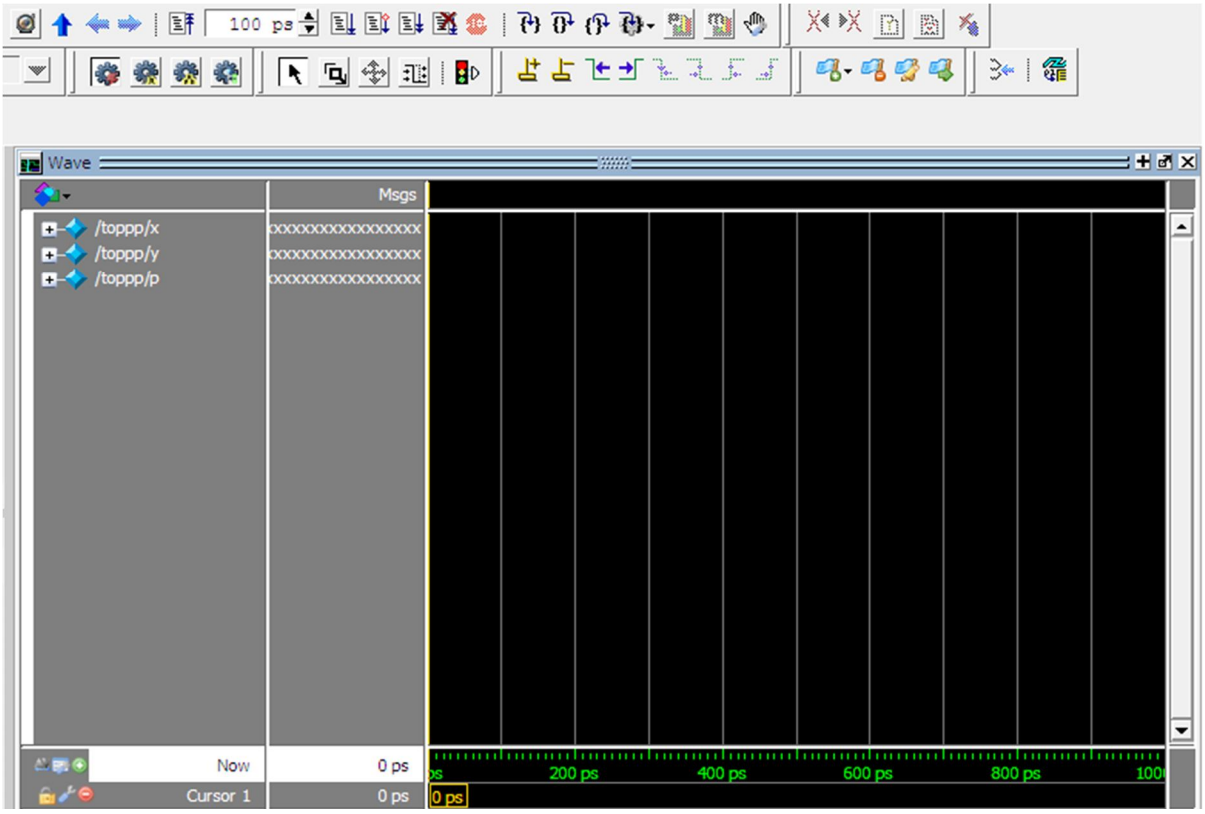


Fig. 10. Time in which the waveform generates 100ps.

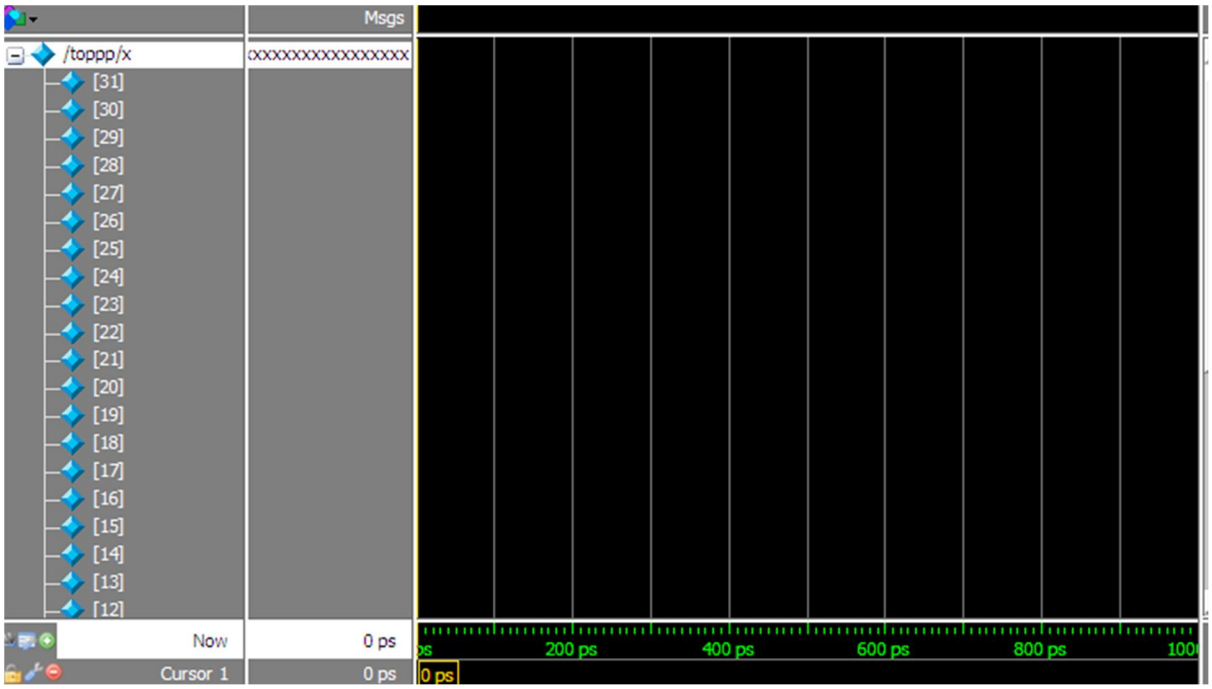


Fig. 11. x contains 32-bit input.

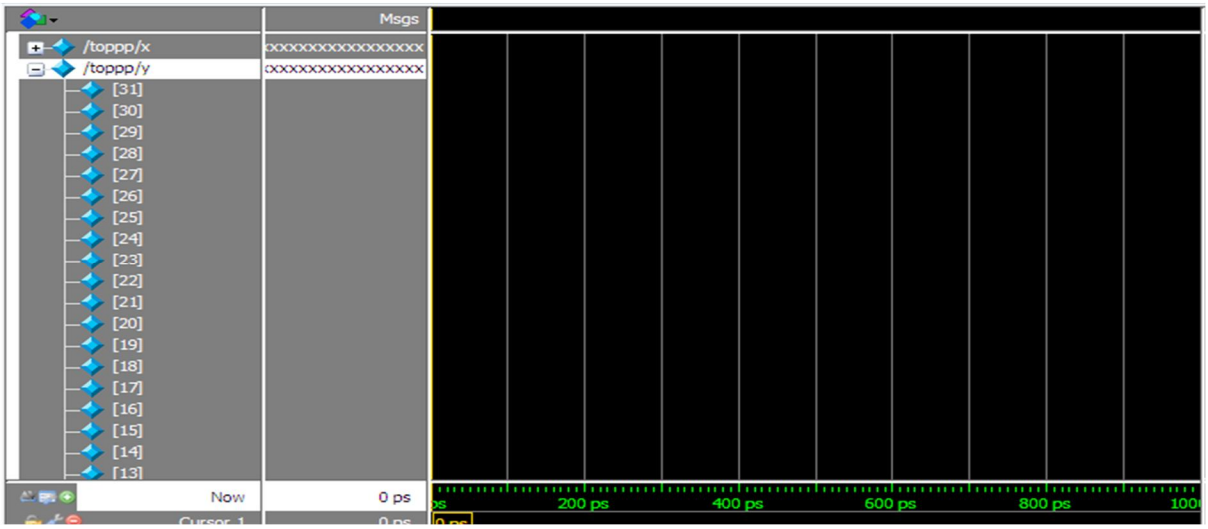


Fig. 12. y contains 32-bit input.

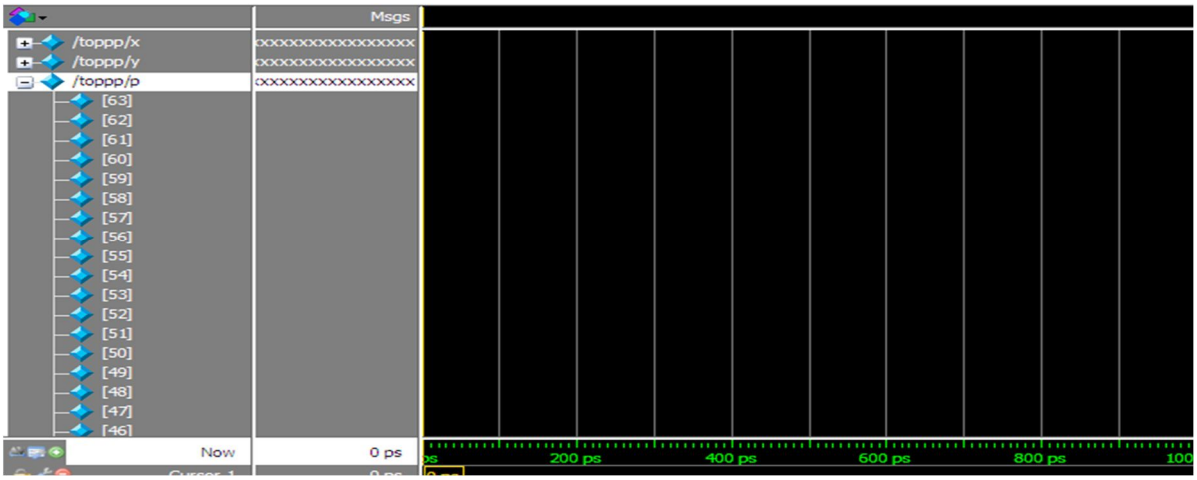


Fig.13. p contains 64-bit output

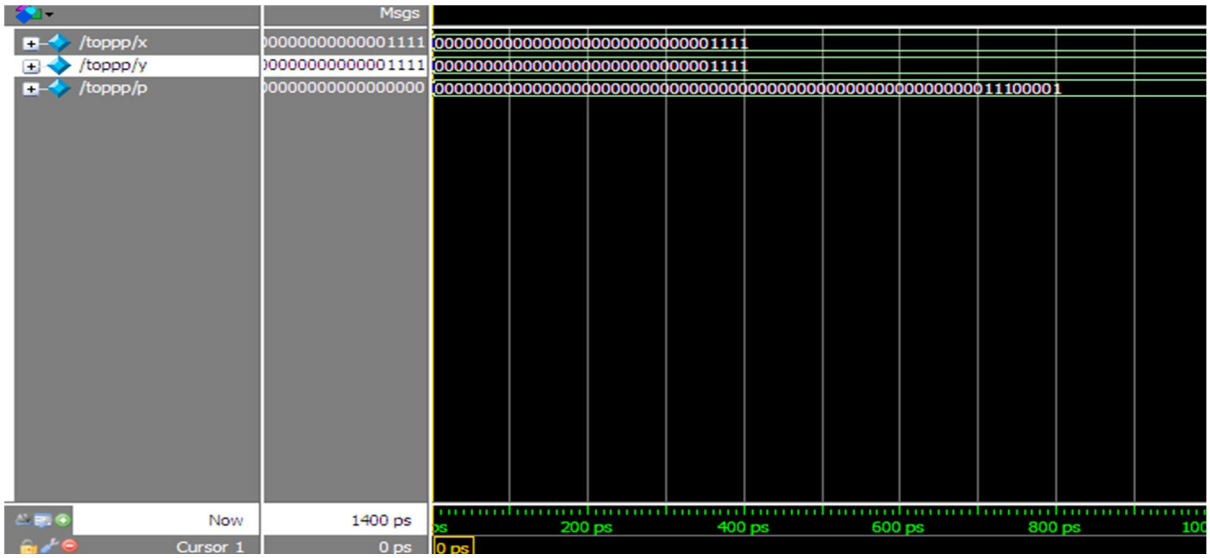


Fig.14. The input and output values of (x,y) and p respectively.

If the input value of x is 00000000000000000000000000001111 and the input value of y is 00000000000000000000000000001111 the product value that is our output is 00000000000000000000000000000000000000000000000000000011100001.
Suppose now we will change the value of x to be 10101000000000000000000000001111 and we will change the value of input y to be 10010000000000000000000000001111 so the value of product p will also change and the output will be 00100110011111111111111111101000100100000000000000000011100001. This shows that our proposed approximate multiplier works efficiently.

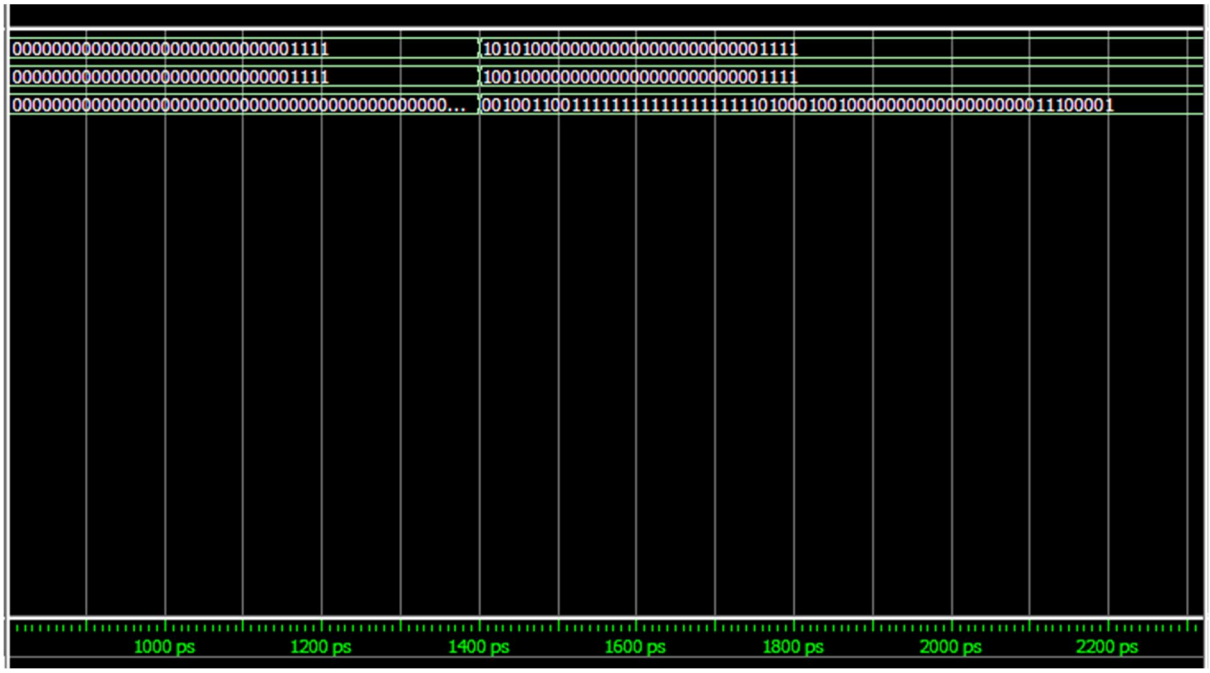


Fig.15. Changing the value of input x and y will give us the value of product p.

Area Calculation of the proposed approximate multiplier.

multiplier Project Status			
Project File:	multiplier.ise	Current State:	Placed and Routed
Module Name:	booth	• Errors:	No Errors
Target Device:	xc5vlx30-3ff324	• Warnings:	1 Warning
Product Version:	ISE 10.1 - Foundation Simulator	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

multiplier Partition Summary [-]
No partition information was found.

Device Utilization Summary [-]				
Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice LUTs	1,979	19,200	10%	
Number used as logic	1,978	19,200	10%	
Number using O6 output only	1,946			
Number using O5 output only	32			
Number used as exclusive route-thru	1			
Number of route-thrus	33	38,400	1%	
Number using O6 output only	33			
Slice Logic Distribution				
Number of occupied Slices	749	4,800	15%	
Number of LUT Flip Flop pairs used	1,979			
Number with an unused Flip Flop	1,979	1,979	100%	
Number with an unused LUT	0	1,979	0%	

Fig.16. Area of the proposed approximate multiplier

Several Slice LUT (Look Up Table)- Area is 19200 and utilization is only 10% the minimum is the utilization is the area consumption will be less.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The power calculation of the proposed approximate multiplier.

Name	Power (W)	Used	Total Available	Utilization (%)
Logic	0.000	1979	19200	10.3
Signals	0.000	1599	---	---
IOs	0.000	128	222	57.7
Total Quiescent Power	0.300			
Total Dynamic Power	0.000			
Total Power	0.300			

```
Power summary                                |      I(mA)     |      P(mW)     |
---------------------------------------------------------------------------
Total estimated power consumption            |                |         300    |
                                             ---
```

Fig.17. Power Consumption- 300mW (Power consumption is less).

1) Time Details of the proposed Multiplier

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 399181947670842620 / 64
-------------------------------------------------------------------------
Delay:               34.965ns (Levels of Logic = 54)
  Source:            x<0> (PAD)
  Destination:       p<63> (PAD)

  Data Path: x<0> to p<63>
                                 Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             43   0.694   0.452  x_0_IBUF (x_0_IBUF)
     LUT1:I0->O             1   0.086   0.000  Madd_inv_x_cy<0>_rt (Madd_inv_x_cy<0>_rt)
     MUXCY:S->O             1   0.305   0.000  Madd_inv_x_cy<0> (Madd_inv_x_cy<0>)
     MUXCY:CI->O            1   0.023   0.000  Madd_inv_x_cy<1> (Madd_inv_x_cy<1>)
     XORCY:CI->O           34   0.300   0.526  Madd_inv_x_xor<2> (inv_x<2>)
     LUT4:I2->O             1   0.086   0.000  spp_1_mux0000<2>_G (N4192)
     MUXF7:I1->O            3   0.214   0.776  spp_1_mux0000<2> (spp<1><4>)
     LUT5:I0->O             4   0.086   0.447  abc[1].UUT/aa[4].b1/sk1p1 (abc[1].UUT/C<5>)
     LUT5:I3->O             4   0.086   0.447  abc[1].UUT/aa[6].b1/sk1p1 (abc[1].UUT/C<7>)
     LUT5:I3->O             2   0.086   0.772  abc[1].UUT/aa[8].b1/Mxor_sk1m_xo<0>1 (prod1<1><8>)
     LUT5:I0->O             3   0.086   0.369  abc[2].UUT/aa[8].b1/Mxor_sk1m_xo<0>1 (prod1<2><8>)
     LUT5:I4->O             4   0.086   0.861  abc[3].UUT/aa[8].b1/sk1p1 (abc[3].UUT/C<9>)
     LUT6:I0->O             1   0.086   0.000  abc[3].UUT/aa[9].b1/sk1p12 (abc[3].UUT/aa[9].b1/sk1p11)
     MUXF7:I0->O            3   0.213   0.444  abc[3].UUT/aa[9].b1/sk1p1_f7 (abc[3].UUT/C<10>)
```

```
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[43].b1/sk1p1 (abc[15].UUT/C<44>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[45].b1/sk1p1 (abc[15].UUT/C<46>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[47].b1/sk1p1 (abc[15].UUT/C<48>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[49].b1/sk1p1 (abc[15].UUT/C<50>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[51].b1/sk1p1 (abc[15].UUT/C<52>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[53].b1/sk1p1 (abc[15].UUT/C<54>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[55].b1/sk1p1 (abc[15].UUT/C<56>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[57].b1/sk1p1 (abc[15].UUT/C<58>)
     LUT5:I3->O             3   0.086   0.444  abc[15].UUT/aa[59].b1/sk1p1 (abc[15].UUT/C<60>)
     LUT5:I3->O             2   0.086   0.365  abc[15].UUT/aa[61].b1/sk1p1 (abc[15].UUT/C<62>)
     LUT6:I5->O             1   0.086   0.235  abc[15].UUT/aa[63].b1/Mxor_sk1m_xo<0>1 (p_63_OBUF)
     OBUF:I->O                  2.144          p_63_OBUF (p<63>)
    ----------------------------------------
    Total                      34.965ns (7.935ns logic, 27.031ns route)
                                        (22.7% logic, 77.3% route)

=========================================================================
```

Fig.18. Total Time taken so Time-34.96ns.

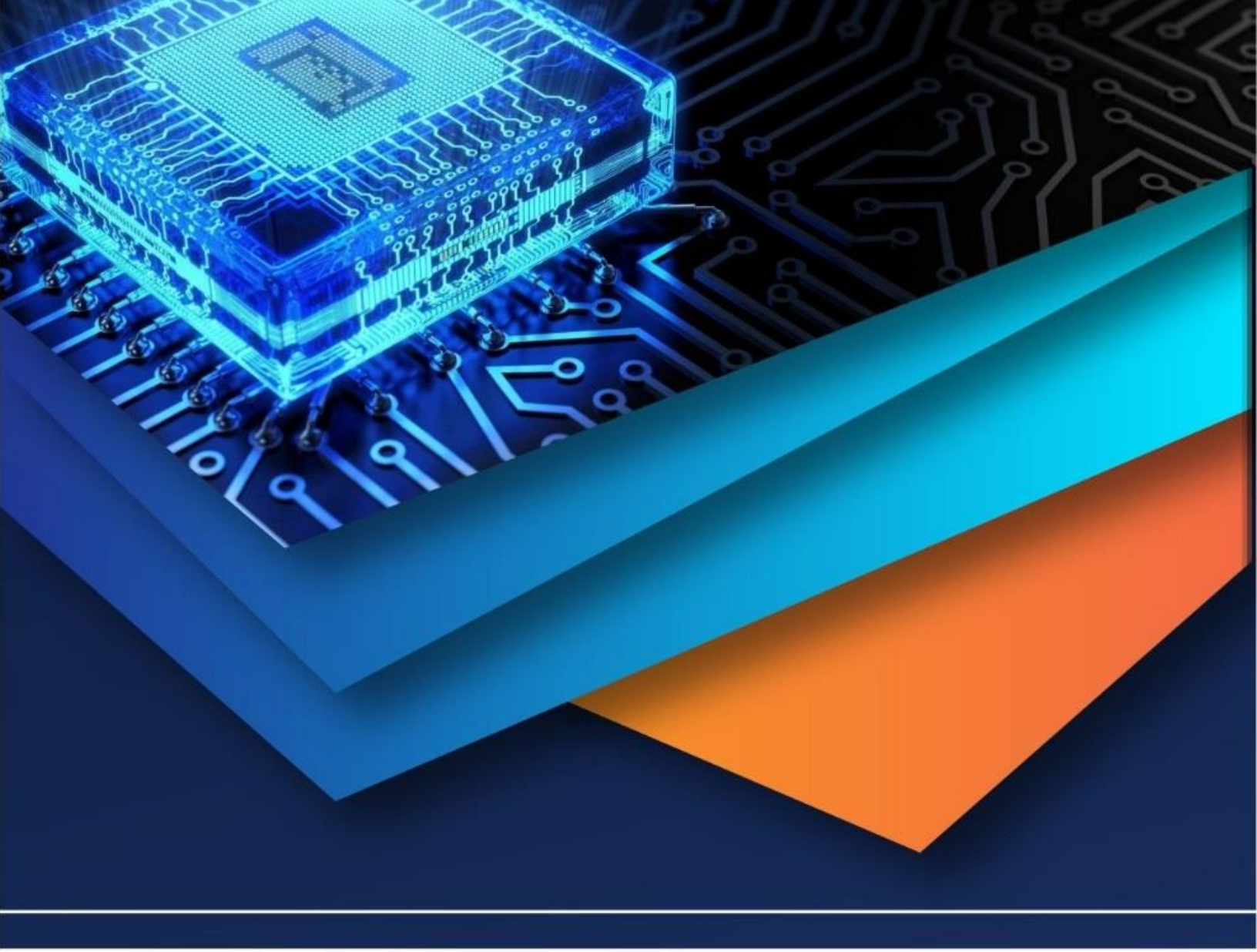
V. CONCLUSION

In this investigation, several approximation multipliers based on approximation in partial product summation were constructed, assessed, and compared. The architecture may multiply 32-bit input and produce 64-bit output; the design space of approximate multipliers is discovered to be mostly reliant on the type of approximation FA utilized. Compared with the different existing multipliers our proposed multiplier has less area that is several Slice LUT (Look Up Table)- The area is 19200 and utilization is only 10% the minimum is the utilization is the area consumption will be less, time delay is less and power consumption is less which is 34.96 ns, and 300 mW in contrast to the Wallace tree multiplier, Booth multiplier and other existing multiplier which shows that our suggested approximate multiplier is more efficient and flexible.

REFERENCES

[1] F. Y. Gu, I. C. Lin, and J. W. Lin, "A Low-Power and High-Accuracy Approximate Multiplier With Reconfigurable Truncation," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 10, pp. 2022.
[2] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in VLSI Design, 2011, pp. 346–351.
[3] C. K. Koc, "Parallel Canonical Recording," Electronics Letters, Vol. 32, No. 22, 1996, pp. 2063-2065. doi:10.1049/el:19961402.
[4] A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly Journal of Mechanics and Applied Mathematics, Vol. 4, No. 2, 1951, pp. 236-240. doi:10.1093/qjmam/4.2.236.
[5] C. S. Wallace, "A Suggestion for a Fast Transactions on Electronic Multiplier," IEEE Computers, Vol. 13, No. 2, 1964, pp. 14-17. doi:10.1109/PGEC.1964.263830.
[6] M. D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufmann Publishers, Burlington, 2003.
[7] D. Villeger and V. G. Oklobdzija, "Evaluation of Booth Encoding Techniques for Parallel Multiplier Implementation," Electronics Letters, Vol. 29, No. 23, 1993, pp. 2016-2017. doi:10.1049/el:19931345
[8] V. G. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree tion and Genand Optimized Final Adder in CMOS Technology," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 3, No. 2, 1995, pp. 292-301.
[9] V. G. Oklobdzija, D. Villeger and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Transactions on Computers, Vol. 45, No. 3, 1996, pp. 294-306. doi:10.1109/12.485568
[10] P. F. Stelling and V. G. Oklobdzija, "Optimal Circuits for Parallel Multipliers," IEEE Transactions on Computers, Vol. 47, No. 3, 1998, pp. 273-285.
[11] A. Weinberger and J. L. Smith, "An L Addition," National Bure logic for High-Speed au of Standards Circulation, Vol.591,1985, pp. 3-12.
[12] A. Weinberger, "4:2 Carry-Save Adder Module," IBM Technical Disclosure 3811-3814. [21] D. Villeger and V. G. Oklobdzija, "Analysis of Booth Encoding Ef pressors for Reduction of Partial Products," 1993 Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems, and Computers, Vol. 1, 1993, pp. 781-784.
[13] K. N. Singh and H. Tarunkumar, "A review on various multipliers designs in VLSI," Annual IEEE India Conference, pp 1-4, 2015 [2] B. Lamba, and A. Sharma, "A review paper on different multipliers based on their different performance parameter", 2nd International Conference on Inventive Systems and Control, pp 324-327, 2018.
[14] Savita Nair, "A review paper on comparison of multiplier based on performance parameter", International journal of computer application, vol-2, pp 6-9, 2014.
[15] Bhawna Singroul, Pallavee Jaiswal, "A review on performance Evaluation different digital multiplier in VLSI using VHDL", International Journal of Engineering Research & Technology, vol-7, issue-5,2018.
[16] Sumit Vaidya, Deepak Dandekar, "A review on delay performance comparison of multiplier in VLSI circuit design", International journal of computer network & communication, Vol 2, issue 4, pp 47-55, 2010.
[17] Soniya, Suresh Kumar, "A review of different types multiplier and multiplier accumulator unit", International journal Emerging Trends and technology in computer science, vol-2, issue-4, pp 364-368, 2013.
[18] Bhavya Lahari Gundapaneni, JRK Kumar Dabbakutti, " A review on Booth Algorithm for the design of multiplier", International Journal of Innovative Technology and Exploring Engineering, vol-8, issue-7, pp 1506-1509, 2019.
[19] Kiran Kumar, S. Anusha, G. Y. Rekha, "A design of low power modified Booth multiplier", International journal of current engineering and scientific research, vol-5, issue-4, pp 287-292, 2018.
[20] A. D. BOOTH, "A signed Binary multiplication technique", in the journal of Mech. APPL. Math, Oxford University Press, vol-4, pp 236-240,1951.
[21] Shweta S. Khobragade, Swapnil P. Kormore, "A review on low power VLSI design of Modified Booth multiplier", International Journal of Engineering and Advanced Technology, vol-2, issue-5, pp 463-466, 2015.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)