DESIGN AND IMPLEMENTATION OF 16-BIT APPROXIMATE RADIX 16 BOOTH MULTIPLIER

Mrs. Ch.DhanaLakshmi[1], A. Princes Shara[2], T. Vanitha[3], M. Chinni Sravani[4], Y. Geethika Sree[5]
[1]Assistant Professor, Dept. of ECE, Vignan's Institute of Engineering for Women, Duvvada.
[2,3,4,5],UG Students, Dept. of ECE, Vignan's Institute of Engineering for Women, Duvvada.
[2]shara17221@gmail.com

ABSTRACT-
Multiplier is one of the hardware blocks which generally occupies a significant chip area and is needed to be minimized which will be fruitful to a number of operations. The performance of the multiplier is expressed in terms of high speed and low power parameters. still, the multiplication operation is present in numerous segments of a digital system and substantially used in signal processing operations. The booth algorithm is a multiplication algorithm that permits us to multiply the two signed binary integers in 2's complement, independently. This paper presents the design and implementation of a 16-bit radix 16 Booth multiplier. The multiplier is based on the radix-16 Booth algorithm, which reduces the number of partial products required for multiplication, resulting in a faster and more efficient multiplication process.The proposed design includes a modified partial product reduction stage that uses a carry save adder tree to reduce the number of adders required, and a Wallace tree multiplier to reduce the number of multipliers required. The design also includes a dynamic scaling technique that adjusts the scaling factor of the partial products based on the input operands, further optimizing the multiplication process. The proposed multiplier was implemented using Verilog HDL and synthesized using the Xilinx Vivado 2018.3 version. The simulation results show that the proposed design achieves a significant reduction in the number of adders and multipliers required, resulting in a faster and more efficient multiplication process compared to a conventional radix-16 Booth multiplier.
Index terms– Radix 8 booth multiplier, Radix 16 booth multiplier, XilinxVivado.

INTRODUCTION

Multipliers are more complex than adders and subtractors, so the speed of a multiplier generally determines the operating speed of a DSP system and the operation where it's used. In ultrahigh-speed DSP, multimedia operations, and GPU, the multiplier plays an important role because it dominates the chip's power consumption and operation speed. So far, exploration has concentrated on high-speed and low-power multipliers for computationally fast movable bias. Any kind of multiplier is divided into three stages similar to partial product generation, partial product addition, and the final addition stage.
The speed of multiplication can be increasedby decreasing the number of partial products. The booth algorithm is the most significantly used algorithm. The booth multiplication algorithm is quicker than other algorithms. The algorithm was designed by Andrew Donald in 1950, and the main importance of the booth multiplier is that it can multiply both signed and unsigned binary values. Some encoding methods are available depending on the number of bits in the group, such as radix-2, radix-4, radix-8, and radix-16.
Booth's algorithm for two complements multiplication:

- Multiplier value and multiplicand value are dumped in Y and X register correspondingly.
- Final results will be accumulated in the accumulator and Y registers.
- At the beginning of the process Accumulator and Y-1 register will be 0. The multiplication process is done in a pattern.
- A 1-bit register Y-1 is placed right of the LSB Y0 of the Y register.
- In each stage, Y0 and Y-1 bits are inspected.

i. If Y0 and Y-1 are 11 or 00 then the bits of accumulator, Y, and Y-1 are shifted right by 1 bit.
ii. If the value of Y0 and Y-1 is 10 then the multiplicand is subtractedfrom the accumulator, and the result will be stored in the accumulator. After this, the accumulator, Y, and Y-1 registers are shifted right by 1 bit.
iii. If the value of Y0 and Y-1 is 01 then the multiplicand is added to the accumulator, and the result will be stored in the accumulator. After this, the accumulator, Y, and Y-1 registers areshifted right by 1 bit.

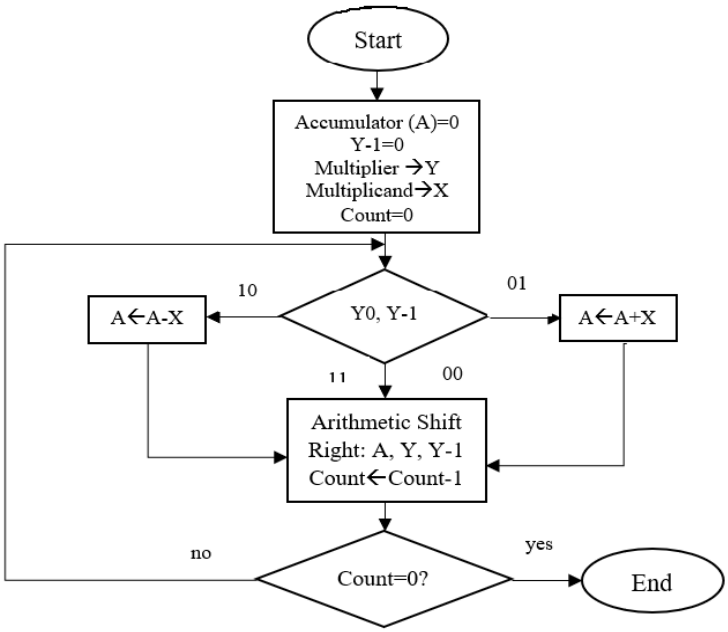

Fig.1 Flow chart of Booth Multiplier

Radix-2:

In the radix-2 algorithm, zero is added to the rightmost side of the multiplier bits, and they are grouped in such a way that each group consists of two bits. So that the first pair consists of an added zero and LSB of 6 multipliers, and the coming pair is the overlapping of the first brace in which the MSB of the first pair will be the LSB of the second brace. So, for n-bit, * n-bit multiplication, n partial products are produced. The radix-2 booth algorithm produces the same number of partial products as an array multiplier, so the number of cycles to evaluate the result is approximately similar.

Radix-4:

To further reduce the number of partial products, algorithms with a higher radix value are used. In the radix-4 algorithm, a grouping of multiplier bits is complete in such a way that each group consists of 3 bits. Then the next pair is the overlapping of the first pair, in which the MSB of the first pair will be the LSB of the second pair and the other two bits. The number of groups formed is based on the number of multiplier bits. By applying this algorithm, the number of partial product rows to be accumulated is decreased from n in the radix- 2 algorithms to n/ 2 in the radix- 4 algorithms.

Radix-8:

Radix-4 booth recording is a multiplier that picks the proper shift and adds operation based on product register bit groups. The product register holds a multiplier. The multiplicand and its two complements are added depending on the encoding value. Radix-8 booth uses the same process as radix-4, but with quartets of bits. Radix-8 lower the partial products to n/3, where n is multiplier bits. It is permissible to gain time when summing incomplete products. The radix-8 algorithm is superior to radix-4. Implementation of Y+2Y each time in a

radix-8 multiplier adds latency and power expense. So, a customized radix-8 booth multiplier is used to boost performance.

Radix-16:

The speed of multiplication can be increased by decreasing the number of partial products. To furtherreduce the number of partial products, algorithms with an advanced radix value are used. In the radix-16 algorithm, the classification of multiplier bits is done in such a way that each set consists of 4 bits. Also, the next pair is the overlap of the first pair, in which the MSB of the first pair will be the LSB of the second pair and the other three bits. The number of sets formed is based on the number of multiplier bits. By applying this algorithm, the number of partial product rows to be accumulated is decreased from n/ 3 in the radix- 8 algorithm to n/ 4 in the radix- 16 algorithms.

LITERATURE REVIEW

In [1] the use of approximate computing is a popular design methodology that reduces circuit complexity, increases performance (low delay), and minimizes power consumption. Two approximate radix-4 MBE algorithms are presented and analyzed in this paper. Booth multipliers are designed based on radix-4 MBEs, where a regular partial product array is obtained using an approximate Wallace tree structure. As a result, the error characteristics are analyzed with an approximation factor derived from the inexact bit width of the Booth multipliers. Additionally, simulation results are presented on delay, area, and power consumption for 45 nm CMOS technology. In this paper, we present case studies for image processing in order to demonstrate the validity of the proposed approximate radix-4 Booth multipliers.

As a result, the R4ABE1 design has the advantage of only having four entries modified; however, the modifications are all to change a '1' to a '0', which always means the approximate output is smaller than the exact output. There has been a proposal to improve the performance of binary number multiplication using booth encoding for two-s complement numbers; this has been further improved by using MBE or radix-4 Booth encoding. In the Booth multiplier, the Booth encoder plays an important role in reducing by half the number of partial product rows. Let's consider the multiplication of two N-bit integers, i.e., multiplicand A and multiplier B.

In [2] the terms of approximate adders, simplifies adder logic so that approximate adders can be produced. The Guptas et al. To build approximate adders, approximate the logic functions of full adder cells to construct imprecise full adder cells. However, it remains unclear how these adders can be used in different tree architectures and how error scales with multi-operand accumulation. The reference proposes an approximate adder that consists of two parts: an accurate part and an inaccurate part. Reference designs a multiplier in which the LSBs of the additions are derived by applying bitwise OR to the respective input bits. A fast approximate adder is produced by limiting the carry propagation, based on proof that the longest carry chain in an n-bit adder is log n. However, despite the fact that all these techniques demonstrate the benefit of efficient computing, their fixed functionality and low-level design limit further improvements in efficiency.

In [3] Padmanabhan Balasubramanianetal., (2021) have proposed a new approximate adder that's designed-optimized and optimized error criteria. Optimization in design which translates into the optimization of the design parameters similar to detention, power, and area/ coffers, and optimization of the error criteria points to the utility of the proposed approximate adder for practical operations.

In[4] the concept of approximate computing is used in error-tolerant applications to reduce the accuracy of operations in order to increase other measurements of circuit performance. In approximate computing, applications are able to tolerate error due to their innate ability to tolerate it. It is usually acceptable in applications like digital signal processing, image processing, data mining, and pattern recognition to have relaxed accuracy requirements. It is a popular multiplication algorithm that reduces the size of the partial

product array by half using Radix-4 modified Booth encoding. In this paper, three Approximate Booth Multiplier Models (ABM-M1, ABM-M2, and ABM-M3) are proposed in which approximate computing is applied to the radix-4 modified Booth algorithm.

RELATED WORK

Radix 816-bit Booth Multiplier is an algorithm used in digital signal processing and arithmetic operations. It is a modified version of the Booth's algorithm, which is used to multiply two numbers in a single cycle. The algorithm is based on the concept of partial products, which are generated using the Booth's encoding technique. The algorithm works by encoding the two numbers in a particular way and then adding the partial products together to get the final result. This technique is more efficient than the traditional multiplication algorithm, as it requires fewer clock cycles to complete the multiplication process.The radix 8 booth multiplier can be enhanced by introducing parallelism which results in reducing the partial products by n/3 where n is the number of multiplier bits.

Radix-8 means: $8 = 2^3 = (1000)_2$

Radix-8 uses 4-bit

So, a group of 4-bit binary number is taken. For a group of 4-bits the Signed Multiplier Digit is specified in below Table.1 which defines Radix-8 Booth's recoding technique for all possible combinations in the binary input where Y is the Multiplier. There will be four partial products. Radix-8 Booth's Multiplier Sign Extension Trick defines the partial products as given below:

[Partial Product 1]

[Partial Product 2]0 0 0

[Partial Product 3] 0 0 0 0 0 0

[Partial Product 4] 0 0 0 0 0 0 0 0 0

Table 1: Radix-8 Booth's recoding technique

Multiplier bits				**Signed Multiplier digit**
Y_{j+2}	Y_{j+1}	Y_j	Y_{j-1}	
0	0	0	0	0
0	0	0	1	X
0	0	1	0	X
0	0	1	1	2X
0	1	0	0	2X
0	1	0	1	3X
0	1	1	0	3X
0	1	1	1	4X
1	0	0	0	-4X
1	0	0	1	-3X
1	0	1	0	-3X
1	0	1	1	-2X
1	1	0	0	-2X
1	1	0	1	-X
1	1	1	0	-X
1	1	1	1	0

Radix 1616-bit Booth Multiplier is a hardware multiplier that can be used to multiply two 16-bit numbers. It is a high-speed multiplier that can perform operations in a fraction of the time compared to other methods of multiplication.The Radix 16-16bit Booth Multiplier is made up of two components: the multiplier and the Booth

encoder. The multiplier is a 16-bit adder-subtractor that can add and subtract two 16-bit numbers. The Booth encoder is used to encode the two 16-bit numbers into a 16-bit representation that can be used by the multiplier.The radix 16 booth multiplier can be enhanced by introducing parallelism which results in reducing the partial products by n/4 where n is the number of multiplier bits.

Radix-16 means: $16 = 2^4 = (10000)_2$

Radix-16 uses 5-bit

So, a group of 5-bitsis taken in the input binary number. Signed multiplier digit for the group is defined in Table 2 as per the Booth's recoding technique for every binary combination where Y is the multiplier.There will be five partial products as per the Booth's multiplier sign extension trick as explained further:

Radix-16 Booth's Multiplier Sign Extension Tricks.

[Partial Product 1]

[Partial Product 2]0000

[Partial Product 3]00000000

[Partial Product 4] 000000000000

[Partial Product 5] 0000000000000000

Table 2: Radix-16 Booth's recoding technique

Multiplier bits					**Signed Multiplier digit**
Y_{j+2}	Y_{j+1}	Y_j	Y_{j-1}	Y_{j-2}	
0	0	0	0	0	0
0	0	0	0	1	X
0	0	0	1	0	X
0	0	0	1	1	2X
0	0	1	0	0	2X
0	0	1	0	1	3X
0	0	1	1	0	3X
0	0	1	1	1	4X
0	1	0	0	0	4X
0	1	0	0	1	5X
0	1	0	1	0	5X
0	1	0	1	1	6X
0	1	1	0	0	6X
0	1	1	0	1	7X
0	1	1	1	0	7X
0	1	1	1	1	8X
1	0	0	0	0	-8X
1	0	0	0	1	-7X
1	0	0	1	0	-7X
1	0	0	1	1	-6X
1	0	1	0	0	-6X
1	0	1	0	1	-5X
1	0	1	1	0	-5X
1	0	1	1	1	-4X
1	1	0	0	0	-4X
1	1	0	0	1	-3X

1	1	0	1	0	-3X
1	1	0	1	1	-2X
1	1	1	0	0	-2X
1	1	1	0	1	-X
1	1	1	1	0	-X
1	1	1	1	1	0

From the above tables signed multiplier digit terms ±2X, ±4X, ±8Xcan be obtained by left shifting X by one bit, two bits and three bits position respectively. Remaining terms like ±3X, ±5X, ±6X,±7X are obtained by adding 2X+X,4X+X,4X+2X,4X+3X using recoding adder.

RESULTS AND SIMULATION

Fig.2 and fig.3 shows the simulation and RTL analysis elaborated design results of radix-8 16-bit booth multiplier.Fig.4 and fig.5 shows the simulation and RTL analysis elaborated design results of radix-16 16-bit booth multiplier which indicates radix 16 booth multiplier speed is more than the radix 8 booth multiplier.

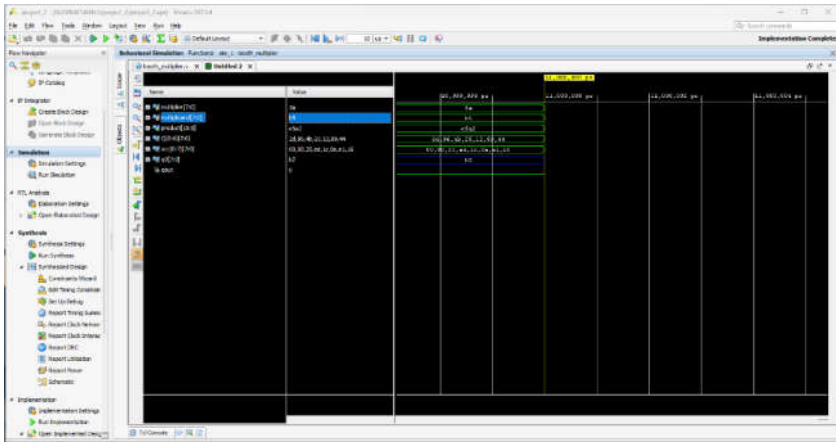

Fig.2 Simulation of Radix-8 16-bit booth multiplier

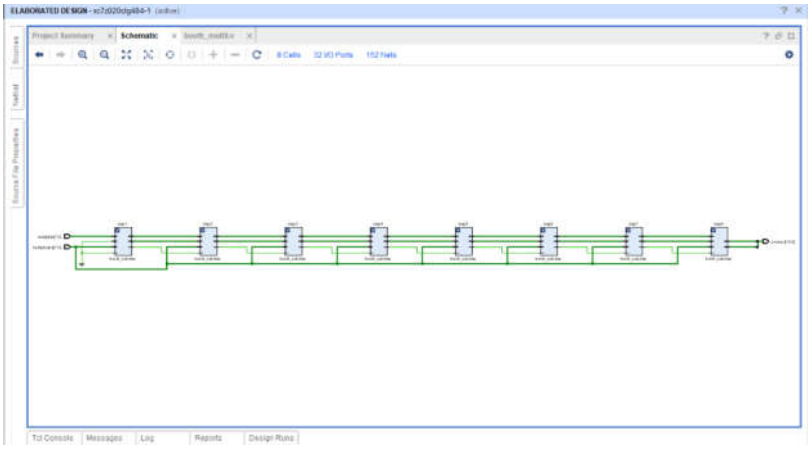

Fig.3RTL Analysis of Radix-8 16-bit booth multiplier

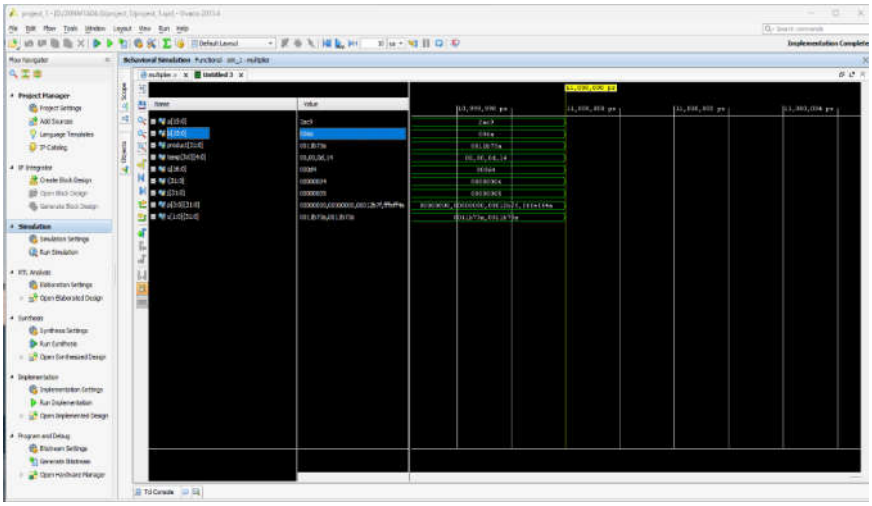

Fig.4 Simulation of Radix-16 16-bit booth multiplier

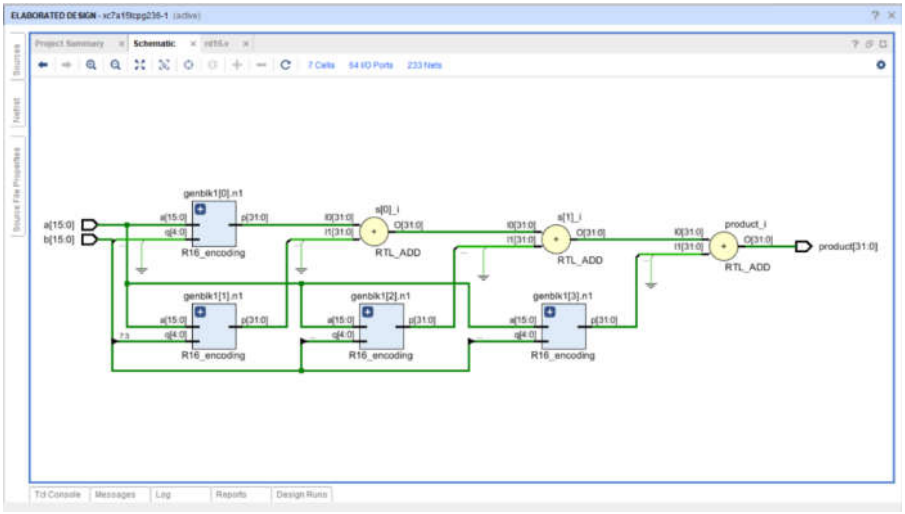

Fig.5 RTL Analysis of Radix-16 16-bit booth multiplier

COMPARISON OF RADIX-8 & RADIX-16:

S.No.	Parameters	Radix-8	Radix-16
1.	Number of bits	16	16
2.	Grouping of bits	4	5
3.	Number of partial products	5	4
4.	LUT	90	787
5.	Logic	1.197W	5.636W
6.	Device Static Power	0.279W	0.485W

CONCLUSION:

This paper has proposed approximate radix-16 booth multiplier for 16 bits.Here we compared different radix by their different criteria. They worked well in either partial product and speed. So, the performance of each radix is different from the others. The radix avoids unwanted glitches and thus minimize the switching power dissipation. As a result of the radix-16 booth multiplier, the number of partial products is reduced by half, since the bits from the multiplier term are grouped in the multiplication operation, which speeds up the process. In conclusion, we have presented a design and implementation of 16-bit approximate radix-16 booth multiplier our results demonstrate the effectiveness of our approach in reducing partial products and increasing speed while maintaining acceptable levels of accuracy.

REFERENCES:

[1] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, **"Design of Approximate Radix-4 Booth Multipliers for Error Tolerant Computing,"** in IEEE Trans. on Computers, vol. 66, no. 8, pp. 1435-1441, Aug. 2017.

[2]V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, **"Approximate Hybrid High Radix Encoding for Energy-Efficient Inexact Multipliers,"** in IEEE Trans. on Very Large-Scale Integration (VLSI) Systems, vol. 26, no. 3, pp. 421-430, Mar. 2018.

[3]V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, **"Low-Power Digital Signal Processing Using Approximate Adders,"** inIEEE Trans. on Computer-Aided Design of Integrated Circuitsand Systems, vol. 32, no. 1, pp. 124-137, Jan. 2013.

[4] S. Venkatachalam, E. Adams, H. J. Lee and S. Ko**, "Design and Analysis of Area and Power Efficient Approximate Booth Multipliers,"** in IEEE Trans. on Computers, vol. 68, no. 11, pp. 1697-1703, Nov. 2019.

[5] K. Y. Kyaw, W. L. Goh and K. S. Yeo, "**Low-power high-speed multiplier for error-tolerant application**," in IEEE Int. Conf. of Electron Devices and Solid-State Circuits (EDSSC), Hong Kong, 2010, pp. 1-4.

[6] H. Jiang, J. Han, F. Qiao and F. Lombardi, "**Approximate Radix-8 Booth Multipliers for Low-Power and High-Performance Operation**," in IEEE Trans. on Computers, vol. 65, no. 8, pp. 2638-2644, Aug. 2016.

[7] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "**Design and Analysis of Approximate Compressors for Multiplication**," in IEEE Trans. on Computers, vol. 64, no. 4, pp. 984-994, Apr. 2015.

[8] S. Venkatachalam and S. Ko, "**Design of Power and Area Efficient Approximate Multipliers**," in IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 5, pp. 1782-1786, May 2017.

[9] C. Liu, J. Han, and F. Lombardi, "**A low-power, high-performance approximate multiplier with configurable partial error recovery**," in DATE, 2014, p. 95.

[10] J. Hidalgo, V. Moreno-Vergara, O. Oballe, A. Daza, M. Mart´ın-Vazquez, and A. Gago, "**A radix-8 multiplier unit design for specific purpose**," in XIII Conference of Design of Circuits and Integrated Systems, vol. 10, 1998, pp. 1535–1546.

[11] N. Zhu, W. L. Goh, and K. S. Yeo, "**An enhanced low-power high-speed adder for error-tolerant application**," in ISIC. IEEE, 2009,pp. 69–72.

[12] W. Liu et al., "**Design and analysis of approximate redundant binary multipliers**," IEEE Trans. Comput., vol. 68, no. 6, pp. 804–819,Jun. 2019.