International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 11, November 2014 **ISSN 2319 - 4847**

Review Paper on High Speed Parallel Multiplier –Accumulator (MAC) Based On Radix-4 Modified Booth Algorithm

Gaurav Pohane [1], Sourabh Sharma[2]

[1] M.Tech Scholars TITR, Bhopal (EC DEPARTMENT)T.I.T.R, (R.G.P.V.)
Bhopal (M.P.), India Bhopal (M.P.), India

[2]Assistant Professor (EC DEPARTMENT)T.I.T.R, (R.G.P.V.)
Bhopal (M.P.), India Bhopal (M.P.), India

Abstract

In this paper, A high speed and area-efficient merged multiply Accumulate (MAC) Units is proposed in this work. To realize the area-efficient and high speed MAC unit proposed in this work, first we examine the critical delays and hardware complexities of conventional MAC architectures to derive at a unit with low critical delay and low hardware complexity. The* new architecture is based on *combination of multiplication with accumulation and devising a hybrid type of carry save adder (CSA), for improved the performance. This can be implementing by using Radix-4 booth encoder. Power dissipation is acknowledged as a critical parameter in modern VLSI field. Reducing the overall area is achieved by the full utilization of the compressors instead of putting zeros in free inputs. Increasing the speed of operation is achieved by avoid using the modified compressor in the critical path. Feeding the bits of the accumulated operand into the summation tree before the final adder helps to increase the speed too. The proposed MAC unit and the previous merged. MAC unit are mapped on a Field Programmable Gate Array (FPGA) chip, in order to compare between them. The simulation result shows that the proposed system for 8-bit, 16-bit, and 32-bit MAC unit reduces area by 6.25%, 3.2 %, and 2.5% and increases the speed by 14%, 16%, and 19% respectively.

Keywords:- Booth encoder, Radix-2 Booth Encoding multiplier, Radix-4 Booth Encoding multiplier, digital arithmetic, low power.

1. INTRODUCTION

Multipliers are used in many different places in VLSI design. It is the non-memory sub-block with the largest size and delay that has a big impact on the cycle time. Multipliers are also very important and frequently used in Digital Signal Processing applications to run complex high speed calculations. [1] Booth multiplication is used greatly to increase the speed of the multiplier by encoding the numbers that are multiplied. This is a standard technique used in chip design and provides significant improvements over the long multiplication technique. In order to improve the speed of the MAC unit, there are two major bottlenecks that need to be considered. The first One is the partial products reduction network that is used in the multiplication block and the second one is the accumulator. Both of these stages require addition of large operands that involve long paths for carry propagation. The classical straightforward approach to build multiply accumulate units implements each part separately using individual functional blocks [1] and then cascade them to realize the complete operation. Another approach to speed up the operation implements both the multiplication and the accumulation operations within the same functional block by merging the accumulator with the multiplication circuit. Using tree architectures for the partial products reduction network represent an attractive solution that is frequently applied to speed up the multiplication process. Booth multiplication is used greatly to increase the speed of the multiplier by encoding the numbers that are multiplied This is a standard technique used in chip design and provides significant improvements over the long multiplication technique. In the conventional multiplier, the number of partial products to be added are determined by the number of bits the multiplier or multiplicand being used. The bigger the number of bits the multiplicand or the multiplier contain, the longer time it takes to produce the product. The delay of multiplier is determined largely by the number of partial products to be added. One of the most popular algorithms used to reduce the number of partial products is Booth Encoding multiplier. Booth Encoding multiplication is able to reduce the number of partial products being encoded to increase the speed of the binary multiplications. Radix-4 Booth Encoding multiplier reduces the number of partial products by half, N/2. [2] This is able to increase the time of compression and contribute to an increase in speed. [3] It was then taken a step further in this analysis by designing and synthesizing Radix-8 Booth Encoding multiplier, Radix-16 Booth Encoding multiplier and Radix-32 Booth Encoding multiplier to determine if the speed reduces or the area increases as the higher the radix-based multiplier designs are. The number of partial products reduces as the Radix-based Booth Encoding

multipliers increase higher. Radix-8 Booth Encoding multiplier will encounter a reduction of N/3 in the partial products while Radix-16 Booth Encoding multiplier reduces its number of partial product by N/4. Radix-32 Booth Encoding multiplier reduces the number of partial products even more by a reduction of N/5. The algorithm to produce the partial products gets a lot more complicated as the higher the Radix-based Booth Encoding multiplier. [4] Many researchers have attempted in designing MAC architecture with high speed and low power consumption. Elguibaly proposed a fast pipelined implementation to lower the MAC architecture's critical delay [5]. Murakami et al.adopted the half array implementation to design a high-speed and area-effective MAC architecture [6]. Raghuneth et al. made use of a carry-save multiplier that can simplify sign extension and saturation, and further applies it on MAC architecture to reduce the unit's area and power consumption [7]. While the abovementioned schemes may increase the computational performance or lower the power dissipation of MAC architectures, these schemes are applied on one type of MAC architecture instead of exploring the other types, which implies that there still is room to improve on a conventional MAC architecture. Li Hsun proposed a low-power Multiplication-Accumulation Computation (MAC) unit using the radix-4 Booth algorithm, by reducing its architectural complexity and minimizing the switching activities [8]. Kwon et al. developed a merged MAC unit based on fast 5:2 compressors instead of 3:2 and 4:2 compressors [9]. Fayed et al. proposed new data merging architecture for high speed multiply accumulate units [10, 11]. The architecture can be applied on binary trees constructed using 4:2 compressor circuits. Increasing the speed of operation is achieved by taking advantage of the available free input lines of the compressor circuits, which result from the natural parallelogram shape of the generated partial products and using the bits of the accumulated value to fill in these gaps. This results in merging the accumulation operation within the multiplication process. In this paper, we introduce a high speed and area-efficient merged Multiply Accumulate (MAC) Units.

2. GENERAL CONSTRUCTION OF MAC UNIT

The general construction of the MAC operation can be presented by this equation

Z + × = B A Z

Where the multiplier A and multiplicand B are assumed to have n bits each and the addend Z has (2n+1) bits. The basic MAC Unit is made up of a multiplier and an accumulator as shown in Fig. 1.The multiplier can also be divided into the partial products generator, summation tree, and final adder. This construct leads to four basic blocks to implement. The summation network represents the core of the MAC unit. This block occupies most of the area and consumes most of the circuit power and delay. Several algorithms and architectures are developed in attempt to optimize the implementation of this bock.

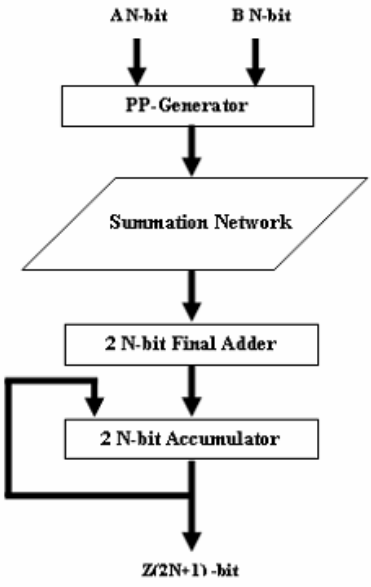

Figure 1. Block diagram of the basic MAC Unit.

The addition network reduces the number of partial products into two operands representing a sum and a carry.The final adder is then used to generate the multiplication result out of these two operands. The last block is the accumulator, which is required to perform a double precision addition operation between the multiplication result and the accumulated operand. This block required a very large adder due to the large operands size. This stage represents a bottleneck in the multiplication process in term of speed since it involves horizontal carry propagation. Tree architectures were proposed to improve the speed of the partial product addition [12] process by introducing

parallelism. The tree structure, which was first, introduced using 3-2 compressors suffer from irregular interconnections and is difficult to layout. It also results in high power consumption as a result of the capacitance introduced by large interconnections. A more regular structure is based upon binary trees construction using modified 4-2 compressors.

3. HIGHER RADIX-BASED BOOTH ENCODING MULTIPLIER

Booth Encoding algorithm is one of the most well-known techniques used to reduce the number of partial products added while multiplying the multiplicand and the multiplier. [8] Reduction in the number of partial products relies upon the number of bits encoded. The algorithm of Radix-based Booth Encoding multipliers are explained below. Let A and B be two n-bit two's complement binary numbers where A represents the multiplicand and B represents the multiplier. A with 16 bits is represented as A15 A14 A13 ... A2 A1 A0 and B with 16 bits are represented as B15 B14 B13 … B2 B1 B0. Both of the multiplicand and the multiplier are signed binary numbers in two's complement form. The general equation for the Radix-based Booth Encoding multiplier is Product, P = A x B.

$$A \times B = \{(Sn-1 \times 2m(n-1)) + (Sn-2 \times 2m(n-2)) + (Sn-3 \times 2m(n-3)) + \ldots\ldots\ldots\ldots\ldots\ldots + (S2 \times 2m(2)) + (S1 \times 2m(1)) + (S0 \times 2m(0))\} \quad (1)$$

where S_k for $n-1<=k<=0$, is a value that depends on the value of B and can be determined as will be explained next while n represents the number of bits in the multiplicand or the multiplier and m = 2, 3, 4, .

Radix-based Booth Encoding multiplier is encoded using 2- bits which are represented as Ck and pairing with Sk to determine the partial product. The value of S_k is shown in Table 1 below for all the possible combinations values of a pair Ck.

TABLE I. COMBINATIONS OF Ck AND S_k FOR RADIX-2 BOOTH ENCODING MULTIPLIER

C_K	S_K
00, 11	0
01	+1 x MULTIPLICAND
10	-1 x MULTIPLICAND

The dot-notation of Radix-2 Booth Encoding multiplier can be represented in Figure 2 below.

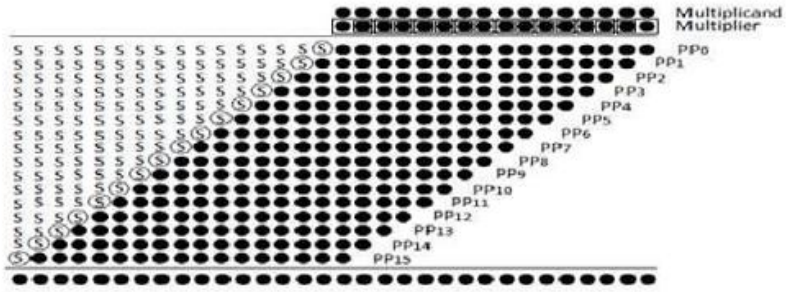

Figure 2: Dot notation of Radix-2 Booth Encoding multiplier.

A collection of 3-bits are taken in one Ck for Radix-4 Booth Encoding multiplier. The number of partial products is reduced to half. The value of S_k for all the possible combinations of a pair C_k is changed as shown in Table 2.

TABLE 2: COMBINATIONS OF Ck AND S_k FOR RADIX-4 BOOTH ENCODING MULTIPLIER

C_K	S_K
000, 111	0
001, 010	+1 x MULTIPLICAND
011	+2 x MULTIPLICAND
100	-2 x MULTIPLICAND
101, 110	-1 x MULTIPLICAND

The dot-notation representation of the Radix-4 Booth Encoding multiplier can be seen in Figure 3.

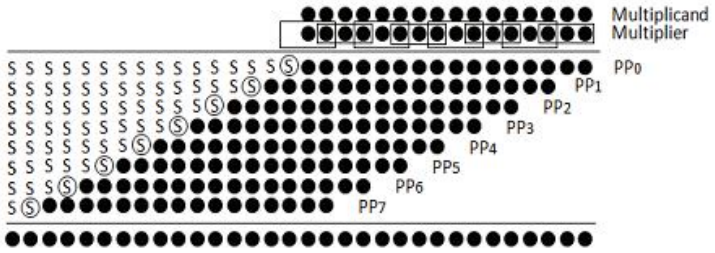

Figure 3: Dot diagram of Radix-4 Booth Encoding multiplier.

(1) $\alpha + \beta = \chi$. (1) (1)

Radix-8 Booth Encoding multiplier uses 4-bit encoding scheme [9] to produce one third the number of partial products. All the possible combinations of a pair of C_k and the value of S_k to encode the partial products re as follow in Table 3.

TABLE 3: COMBINATIONS OF Ck AND S_k FOR RADIX-8 BOOTH ENCODING MULTIPLIER

C_K	S_K
0000, 1111	0
0001, 0010	+1 x MULTIPLICAND
0011, 0100	+2 x MULTIPLICAND
0101, 0110	+3 x MULTIPLICAND
0111	+4 x MULTIPLICAND
1000	-4 x MULTIPLICAND
1001, 1010	-3 x MULTIPLICAND
1011, 1100	-2 x MULTIPLICAND
1101, 1110	-1 x MULTIPLICAND

Figure 4 below displays the dot-notation of Radix-8 Booth Encoding multiplier.

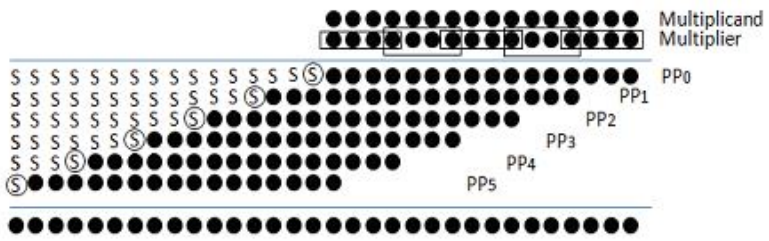

Figure 4: Dot-notation of Radix-8 Booth Encoding multiplier.

This Radix-16 Booth Encoding multiplier uses 5-bit encoding scheme to produce one fourth number of partial products. The combinations of one pair of C_k which consist of 5-bits and S_k to encode Radix-16 Booth Encoding multiplier is shown in Table 4.

TABLE 4: COMBINATIONS OF C_k AND S_k FOR RADIX-16 BOOTH ENCODING MULTIPLIER

C_K	S_K
00000, 11111	0
00001, 00010	+1 x MULTIPLICAND
00011, 00100	+2 x MULTIPLICAND
00101, 00110	+3 x MULTIPLICAND
00111, 01000	+4 x MULTIPLICAND
01001, 01010	+5 x MULTIPLICAND
01011, 01100	+6 x MULTIPLICAND
01101, 01110	+7 x MULTIPLICAND
01111	+8 x MULTIPLICAND
10000	-8 x MULTIPLICAND
10001, 10010	-7 x MULTIPLICAND
10011, 10100	-6 x MULTIPLICAND
10101, 10110	-5 x MULTIPLICAND
10111, 11000	-4 x MULTIPLICAND
11001, 11010	-3 x MULTIPLICAND
11011, 11100	-2 x MULTIPLICAND
11101, 11110	-1 x MULTIPLICAND

The dot-notation of Radix-16 Booth Encoding multiplier can be best represented as follow in Figure 5.

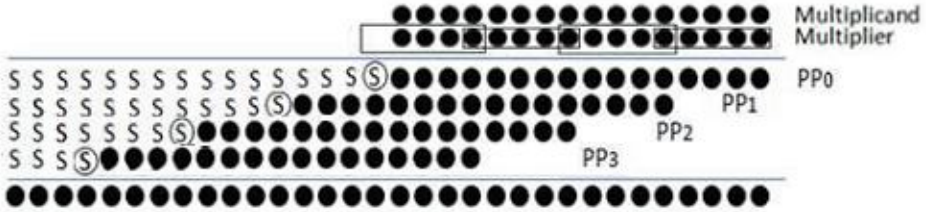

Figure 5: Dot diagram of Radix-16 Booth Encoding multiplier

Radix-32 Booth Encoding multiplier [10] reduces the number of partial products to one fifth. Table 5 shows the combinations of one pair of C_k with the value of S_k.

TABLE 5: COMBINATIONS OF C_k AND S_k FOR RADIX-16 BOOTH ENCODING MULTIPLIER

C_K	S_K
000000, 111111	0
000001, 000010	1 X MULTIPLICAND
000011, 000100	2 X MULTIPLICAND
000101, 000110	3 X MULTIPLICAND
000111, 001000	4 X MULTIPLICAND
001001, 001010	5 X MULTIPLICAND
001011, 001100	6 X MULTIPLICAND
001101, 001110	7 X MULTIPLICAND
001111, 010000	8 X MULTIPLICAND
010001, 010010	9 X MULTIPLICAND
010011, 010100	10 X MULTIPLICAND
010101, 010110	11 X MULTIPLICAND
010111, 011000	12 X MULTIPLICAND
011001, 011010	13 X MULTIPLICAND
011011, 011100	14 X MULTIPLICAND
011101, 011110	15 X MULTIPLICAND
011111	16 X MULTIPLICAND
100000	-16 X MULTIPLICAND
100001, 100010	-15 X MULTIPLICAND
100011, 100100	-14 X MULTIPLICAND
100101, 100110	-13 X MULTIPLICAND
100111, 010111	-12 X MULTIPLICAND
101001, 101010	-11 X MULTIPLICAND
101011, 101100	-10 X MULTIPLICAND
101101, 101110	-9 X MULTIPLICAND
101111, 110000	-8 X MULTIPLICAND
110001, 110010	-7 X MULTIPLICAND

110001, 110010	-7 X MULTIPLICAND
110011, 110100	-6 X MULTIPLICAND
110101, 110110	-5 X MULTIPLICAND
110111, 111000	-4 X MULTIPLICAND
111001, 111010	-3 X MULTIPLICAND
111011, 111100	-2 X MULTIPLICAND
111101, 111110	-1 X MULTIPLICAND

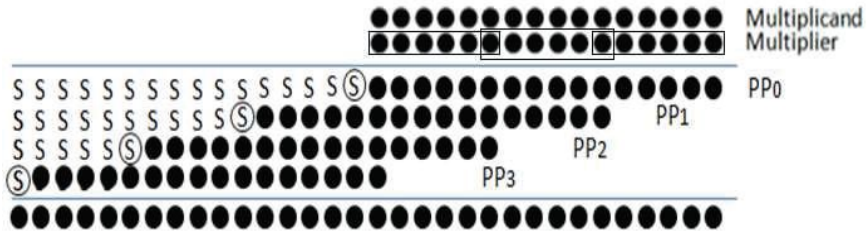

Figure 6 is a dot diagram representation of Radix-32 Booth Encoding multiplier

4. METHODOLOGY

The general steps of creating the Radix-based Booth Encoding multiplier are as shown in Figure 7.

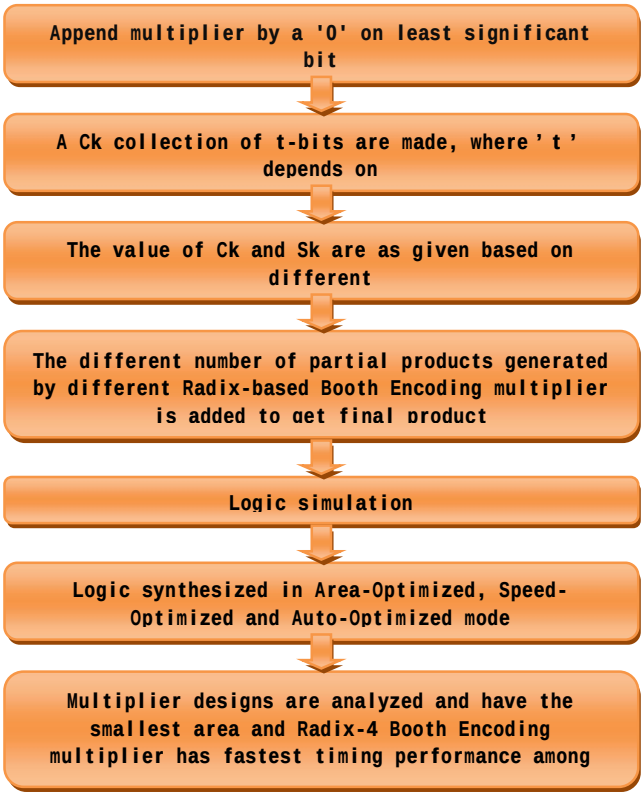

Figure 7: Methodology of Radix-based algorithm

5. RESULTS AND DISCUSSIONS

A. Area Comparison

The best-case given by Area Optimized results showed that Radix-4 Booth Encoding multiplier outperformed the other Radix-based Booth Encoding multiplier and has the most area saving compared to the others. However, the area saving becomes less significant as the higher the Radix-based Booth Encoding multipliers reached. The area reduced to about 44.95% from Radix-2 to Radix-4 Booth Encoding multiplier. The area then showed an increase of 49.10% from Radix-4 to the Radix-8 multiplier design, a hike of 99.15% from Radix-8 to Radix-16 and an increment of 78.06% from Radix-16 to Radix-32. Figure 8 shows the area changes among Radix-based Booth Encoding multiplier in the Area-Optimized mode.

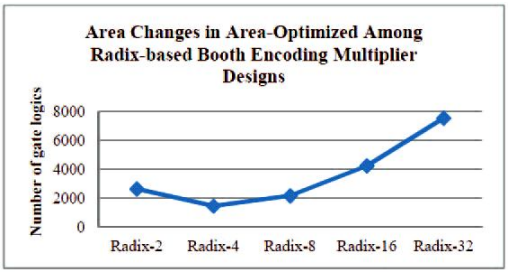

Figure 8: Area changes among Radix-based Booth Encoding multiplier in Area- Optimized mode.

For the Speed-Optimized mode, the logic gate usage increased significantly compared to the Area Optimized mode settings. From Radix-2 to Radix-4 Booth Encoding multiplier, it resulted in a decrement of 50.22%, a continual increment up to 9.13% from Radix-4 to Radix-8 multiplier, 101.33% of increase shown in Radix-8 to Radix-16 multiplier and finally an increase of 87.64% for Radix-16 to Radix-32 multiplier. The area gets bigger as higher the Radix-based Booth Encoding multipliers go because the partial products being computed gets more complicated along the way. Figure 9 below displays the changes in area among the Radix-based Booth Encoding multipliers in Speed-Optimized mode.

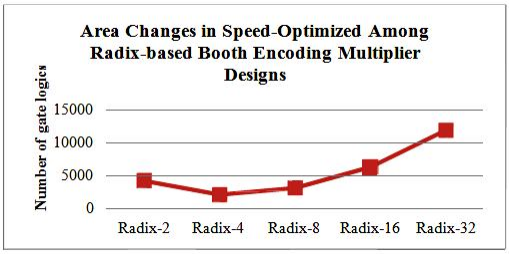

Figure 9: Area changes among Radix-based Booth Encoding multiplier in Speed- Optimized mode.

In the Auto-Optimized mode, the time spent on synthesis effort is expected to be the least and it will produce a result identical in structure to the original HDL design. All the multiplier designs were comparatively very much the same to one another except for Radix-16 and Radix-32 Booth Encoding multiplier. From these findings, we were able to conclude that when a non-optimizing synthesis mode is used, it favors more towards an area-efficient output design.

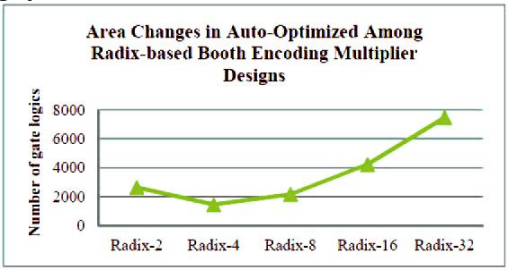

Figure 10: Area changes among Radix-based Booth Encoding multiplier in Auto-Optimized mode during synthesis.

In all three modes, Radix-4 Booth Encoding multiplier has the best area performance. This reason is because more logic gates are needed as the number of Radix-based Booth Encoding increases as the encoding of the increasing number of bits to determine the partial products encoded gets more complicated. That is the main reason we can see an increase in Radix-8, Radix-16 and Radix-32 in terms of area performance.

B. Delay Comparison

In the Speed-Optimized mode, a delay decreased of 18.35ns occurred during the transaction between Radix-2 and Radix-4 Booth Encoding design when the multipliers were synthesized. The delay of the synthesized designs then slightly increased by 1.09ns from Radix-4 to Radix-8, experienced a hike up of 1.86ns from Radix-8 to Radix-16 and a

raised of 1.11ns from Radix-16 to Radix-32 Booth Encoding multiplier. Figure 11 shows the delay changes in the Speed-Optimized mode.

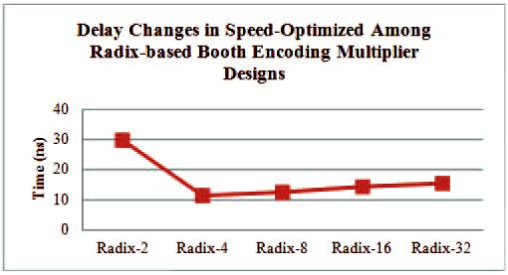

Figure 11: Delay changes summarized in the Speed-Optimized mode.

The results and findings were further compared between the area and delay parameters by computing AD and AD^2 using the values obtained from the previous section. The AD and AD^2 parameters computed will be a great help in assisting the selection of the right applications of the multiplier designs in terms of a particular high speed or a limited area. With the computation of area and delay, the power consumption can be known. Table 8 below shows the parameter of AD and AD^2 tabulated for the Radix-based Booth Encoding multipliers.

TABLE 6: AD AND AD2 COMPUTED FOR RADIX-BASED BOOTH ENCODING DESIGNS IN ALL THREE MODES

AREA-OPTIMIZED					
	RADIX-2 BOOTH ENCODING MULTIPLIER	RADIX-4 BOOTH ENCODING MULTIPLIER	RADIX-8 BOOTH ENCODING MULTIPLIER	RADIX-16 BOOTH ENCODING MULTIPLIER	RADIX-32 BOOTH ENCODING MULTIPLIER
A	2,632	1,451	2,125	4,233	7538
D	29.05	12.29	14.07	15.67	16.57
AD	76,623.09	17,791.90	29,806.56	66,304.47	125072.05
AD^2	2,228,964.82	218,301.71	417,887.41	1,038,327.53	2,074944.65
SPEED-OPTIMIZED					
	RADIX-2 BOOTH ENCODING MULTIPLIER	RADIX-4 BOOTH ENCODING MULTIPLIER	RADIX-8 BOOTH ENCODING MULTIPLIER	RADIX-16 BOOTH ENCODING MULTIPLIER	RADIX-32 BOOTH ENCODING MULTIPLIER
A	4,262	2,123	3,161	6,367	11,948
D	29.79	11.45	12.80	14.39	15.48
AD	126,807.38	24,200.67	39,537.70	91,444.49	184,851.07
AD^2	3,773,787.03	276,128.96	493,750	1,313,14.73	2,859,645.43
	RADIX-2 BOOTH ENCODING MULTIPLIER	RADIX-4 BOOTH ENCODING MULTIPLIER	RADIX-8 BOOTH ENCODING MULTIPLIER	RADIX-16 BOOTH ENCODING MULTIPLIER	RADIX-32 BOOTH ENCODING MULTIPLIER
A	2,633	1,451	2,164	4,233	7,469
D	29.07	12.29	14.05	15.68	16.18
AD	76,623.09	17,791.80	30,311.54	66,273.17	120,309.48
AD^2	2,228,964.83	218,301.75	424,963.58	1,037,837.05	

6. CONCULSION

In this paper, five types of Radix-based Booth Encoding multiplier inclusive of Radix-2, Radix-4, Radix-8, Radix-16 and Radix-32 Booth Encoding multipliers were designed using Verilog HDL, simulated in Modelsim and synthesized to compare the performance in terms of gate level logic and delay using 0.35-micron ASIC Design Kit standard cell library in Leonardo Spectrum. From the results computed, it is found that Radix-4 Booth Encoding multiplier has the

best speed advantage among the 32- bits Radix-based Booth Encoding designs studied here and it is the best-suited for high speed applications. The other higher Radix-based Booth Encoding designs such as Radix-8, Radix-16 and Radix-32 shows a delay smaller than the Radix-2 but the delay increased when compared with the Radix-4 design. The gate level logic performance according to the synthesis results of the Radix-4 multiplier again has the lowest area constraints compared to Radix-2 Booth Encoding multiplier. However the gate level logic of Radix-8, Radix-16 and Radix-32 designs increased compared to Radix-4 Booth Encoding designs. Radix-4 and Radix-8 multiplier are the designs that show a lower area performance compared to the Radix-2 Booth Encoding multiplier.

REFERENCES

[1] Bryan C. Catanzaro, Department of Electrical and Computer Engineering Brigham Young University, "Higher Radix Floating-Point Representations for FPGA-Based Arithmetic", 2005.
[2] Behrooz Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford New York, Oxford University Press, New York, 2000.
[3] C. N.Marimuthu, P. Thangaraj, Anna University, India, "Low Power High Performance Multiplier", ICGST-PDCS International Conference on Computer Science and Engineering, 2008.
[4] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1951-1957, 1999. (conference style)
[5] H.H. Crokell, "Specialization and International Competitiveness," in Managing the Multinational Subsidiary, H. Etemad and L. S, Sulude (eds.), Croom-Helm, London, 1986. (book chapter style)
[6] K. Deb, S. Agrawal, A. Pratab, T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithms for Multiobjective Optimization: NSGA II," KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000. (technical report style).
[7] F. Elguibaly, "A fast parallel multiplier-accumulator using the modified Booth algorithm," IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing, vol. 47, pp. 902-098, Sept. 2000.
[8] H. Murakami, et al." A multiplier-accumulator macro for a 45 MIPS embedded RISC processor," IEEE J. Solid-State Circuits, vol. 31, pp. 1067-1071, July 1996.
[9] R. K. J. Raghunath, et al. "A compact carry-save multiplier architecture and its applications," Proc. IEEE 40th Midwest Symp.Circuits and Systems, vol. 2, pp. 794-797, Aug. 1997.
[10] Li-Hsun Chen; Chen, O.T.C., "A multiplication-accumulation computation unit with optimized compressors and minimized switching activities" Circuits and Systems, 48th Midwest Symposium, MWSCAS.2005, Page(s):1223 - 1226, 2005
[11] Ohsang Kwon, K. Nowka and E. E. Swartzlander, "A 16-bitx16-bit MAC design using fast 5:2 compressors," Proc. Of IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 235 – 243, 2000.
[12] Ayman Fayed, Walid Elgharbawy, and Magdy Bayoumi, "A merged multiply accumulate for hight-speed signal processing application," ICASSP IEEE 2004.
[13] Ayman Fayed , Walid Elgharbawy, and Magdy Bayoumi, "A data merging technique for hight-speed low-power multiply accumulate units," IEEE 2002
[14] Vojin G. Oklobdzija, and David Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," IEEE Trans. on VLSI, Vol. 3, No. 2, pp. 292-301, June 1995.
[15] J. Geralds, "Sega Ends Production of Dreamcast," vnunet.com, para. 2, Jan. 31, 2001. [Online]. Available: http://nl1.vnunet.com/news/1116995. [Accessed: Sept. 12, 2004]. (General Internet site).
[16] Kelly Liew Suet Swee, Universiti Teknologi PETRONAS,Bandar Seri Iskandar," Performance Comparison Review of Radix-Based Multiplier Designs", 2012**.**
[17] A. Abdelgawad "High Speed and Area-Efficient Multiply Accumulate (MAC) Unit for Digital Signal Prossing Applications" The Center for Advanced Computer Studies, University of Louisiana at Lafayette,Lafayette, LA 70504, USA

AUTHOR

Gaurav Pohane received the B.E. degrees in Electrical and Tele Communication Engineering From Rashtrasant Tukadoji Maharaj Nagpur University in 2006, respectively. Pursuing M.Tech in VLSI.