ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2350-0174
Vol.03, Issue.04
July-2023
Pages: 893-995

AREA AND LATENCY OPTIMISED EFFICIENT RADIX-8 BOOTH SIGNED MULTIPLIER

[1]SEERAM SATHISH KUMAR [2]E.ATCHA RAO
[1] M. Tech, Dept. of E.C.E, Kakinada Institute of Technology & Science, Divili, A.P
[2]Assistant Professor, H.O.D, Dept. of E.C.E, Kakinada Institute of Technology & Science, Divili, A.P

ABSTRACT: Binary multipliers are a widely used building block element in the design of microprocessors and embedded systems, and therefore, they are an important target for implementation optimization. The multiplication operation is present in many parts of a digital system or digital computer, most notably in signal processing, graphics and scientific computation. This project presents a design methodology for high-speed area efficient Booth encoded parallel multiplier. For partial product generation, we propose a new modified Booth encoding (MBE) scheme to improve the performance of traditional MBE schemes. Few FPGA-based designs that exist are largely limited to unsigned numbers, which require extra circuits to support signed operations. To overcome these limitations for the FPGA-based implementations of applications utilizing signed numbers, this paper presents an area-optimized, low-latency and energy-efficient architecture for accurate signed multiplier. As an extension of this concept, Radix8 modified booth encoding algorithm is proposed to implement signed multiplication to reduce both area and time constraints.

Index Terms: Booth encoding, Look up table, Latency, Register Transfer Level, Field Programmable Gate Array, Sign Extension.

INTRODUCTION: Energy minimization is major requirements in almost any electronic systems, especially the portable ones such as smart phones, tablets, and different gadgets. It is extremely desired to attain this minimization with minimal performance (speed) penalty [1]. Digital signal processing (DSP) blocks are most wanted in transportable components for realizing various multimedia applications. The computational core of these blocks is the ALU where the multiplications and additions are the major part [6]. The multiplications plays foremost operation in the processing elements which can leads to high consumption of energy and power. Many of the DSP cores implement image and video processing algorithms where final outputs are either images or videos prepared for human consumptions. It facilitates to go for approximations for improving the speed and energy in the arithmetic circuits. This originates from the limited perceptual abilities in observing an image or a video for human beings. In addition to the image and video processing applications, there are other areas where the exactness of the arithmetic operations is not critical to the functionality of the system (see [2],[3]). Approximate computing provides an accuracy, speed and power/energy consumption. The advantage of approximate multiplier reduces the error rate and gain high speed. For correcting the division error compare operation and a memory look up is required for the each operand is required which increases the time delay for entire multiplication process [4]. At various level of abstraction including circuit, logic and architecture levels the approximation is processed [5]. In the category for approximation methods in function, a number of approximating arithmetic building blocks, such as adders and multipliers, at different design levels have been suggested in various structures [6],[7]. Broken array multiplier was designed for efficient VLSI implementation[8].

The error of mean and variance of the imprecise model increase by only 0.63% and 0.86% with reverence to the precise WPA and the maximum error increases by 4%. Low-Power DSP uses approximate adders which are employed in different algorithms and design for signal processing. In contrast with standard multiplier, the dissipated power for the ETM dropped from 75% to 90%. While maintaining the lower average error from the conventional method, the proposed ETM achieves an impressive savings of more than 50% for a 12 x 12 fixed-width multiplication. The crucial part of the arithmetic units are basically built by the multiplier hardware, so multipliers play a prominent role in any design. [1] If we consider a Digital signal processing (DSP) the internal blocks of arithmetic logic designs, where multiplier plays a major role among other operations in the DSP systems [1].So, in the design of multiplier and accumulate unit (MAC) multipliers play an important role. Next, important design in the MAC unit is the Adder. Adders also share the equal important in this design. By the appropriate function methods different kinds of adders and multipliers designs are been suggested. By the approximate computing the designer can make tradeoffs, accuracy, speed, energy and power consumption. In this paper we proposed the modified form of rounding based approximate multiplier which is low power design, high speed and energy efficient.

LITERATURE SURVEY: A traditional method to reduce the aging effects is overdesign which includes techniques like guard-banding ad gate oversizing. This approach can be area and power inefficient [8]. To avoid this problem, an NBTI- aware technology mapping technique was proposed in [7] which guarantee the performance of the circuit during its lifetime. Another technique was an NBTI- aware sleep transistor in [3] which improve the lifetime stability of the power gated circuits under considerations. A joint logic restructuring and pin reordering method in [6] is based on detecting functional symmetries and transistor stacking effects. This approach is an NBTI optimization method that considered path sensitization. Dynamic voltage scaling and bogy-biasing techniques were proposed in [4] and [5] to reduce power or extend circuit life. These techniques require circuit modification or do not provide optimization of specific circuits. Every gate in any VLSI circuit has its own delay which reduces the performance of the chip. Traditional circuits use critical path delays the overall circuit clock cycle in order to perform correctly. However, in many worst-case designs, the probability that the critical path delay is activated is low. In such cases, the strategy of minimizing the worst-case conditions may lead to inefficient designs. For noncritical path, using the critical path delay as the overall cycle period will result in significant timing waste. Hence, the variable latency design was proposed to reduce the timing waste of traditional circuits. A short path activation function algorithm was proposed in [16] to improve the accuracy of the hold logic and to optimize the performance of the variable-latency circuit. An instruction scheduling algorithm was proposed in [17] to schedule the operations on non uniform latency functional units and improve the performance of Very Long Instruction Word processors. In [18], a variable-latency pipelined multiplier architecture with a Booth algorithm was proposed. In [19], process-variation tolerant architecture for arithmetic units was proposed, where the effect of process-variation is considered to increase the circuit yield. In addition, the critical paths are divided into two shorter paths that could be unequal and the clock cycle is set to the delay of the longer one. These research designs were able to reduce the timing waste of traditional circuits to improve performance, but they did not consider the aging effect and could not adjust themselves during the runtime. Chen et al (2003) presented low-power 2's complement multipliers by minimizing the switching activities of partial products using the radix-4 Booth algorithm. Before computation for two input data, the one with a smaller effective dynamic range is processed to generate Booth codes, thereby increasing the probability that the partial products become zero. By employing the dynamic-range determination unit to control input data paths, the multiplier with a column-based adder tree of compressors or counters is designed. To further reduce power consumption, the two multipliers based on row-based and hybrid-based adder trees are realized with operations on effective dynamic ranges of input data. Functional blocks of these two multipliers can preserve their previous input states for non effective dynamic data ranges and thus, reduce the number of their switching operations. It illustrates the proposed multipliers exhibiting low-power dissipation, the theoretical analyzes of switching activities of partial products 27 are derived.

The proposed 16 /spl times/ 16-bit multiplier with the columnbased adder tree conserves more than 31.2%, 19.1%, and 33.0% of power consumed by the conventional multiplier. Furthermore, the proposed multipliers with row-based, hybrid-based adder trees reduce power consumption by over 35.3%, 25.3% and 39.6%, and 33.4%, 24.9% and 36.9%, respectively. Oscal et al (2003) presented low-power 2's complement multipliers by minimizing the switching activities of partial products using the radix-4 Booth algorithm. Before computation for two input data, the one with a smaller effective dynamic range is processed to generate Booth codes, thereby increasing the probability that the partial products become zero. By employing the dynamic-range determination unit to control input data paths, the multiplier with a column-based adder tree of compressors or counters is designed. To further reduce power consumption, the two multipliers based on row-based and hybrid-based adder trees are realized with operations on effective dynamic ranges of input data. Functional blocks of these two multipliers can preserve their previous input states for non effective dynamic data ranges and thus, reduce the number of their switching operations. To illustrate the proposed multipliers exhibiting low-power dissipation, the theoretical analyzes of switching activities of partial products are derived.

LUT BASED RADIX-4 BOOTH'S SIGNED MULTIPLICATION: Using the concepts of radix-4 Booth's multiplication algorithm, we present our area optimized, low-latency and energy efficient accurate signed multipliers. The correct sign of a partial product, in booth's encoding (BE)-based multiplier, is decided by the sign of the multiplicand (the MSB) and the corresponding value of BE. Tables I(a) and I(b) show the list of required sign extensions (SE) for all possible combinations of BE's values and MSB of the multiplicand.

We have used the Bewick's sign extension technique [16] to implement the correct sign of a partial product. Unlike state-ofthe-art implementations, our proposed architecture computes all partial products in parallel and then adds the generated partial products using multiple 4:2 compressors and a ripple carry adder (RCA). The parallel generation of partial products significantly reduces the critical path delay of the multiplier. The illustration of our contributions is shown in green in Fig. 1. Our implementations provide optimized configurations for the 6-input LUTs and the associated carry chains in a logic-slice of modern FPGAs, such as Xilinx Virtex-7 series.

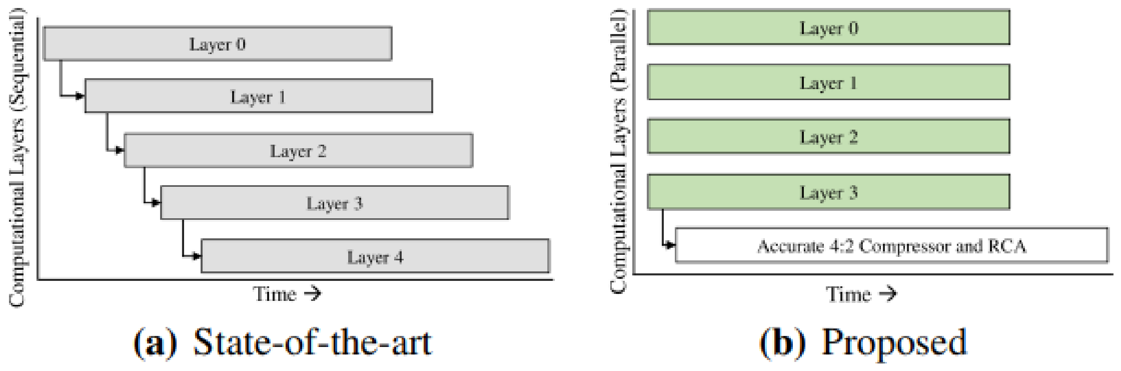


Fig. 1: Comparison of theoretical logic delays for an 8×8 multiplier using (a) technique used in [7], [8] and (b) proposed approach. The time scale is for illustration purpose only

Accurate Signed Partial Products Generation:

Fig. 2 shows the configurations of the 6-input LUTs used for the implementation of the proposed accurate multiplier.

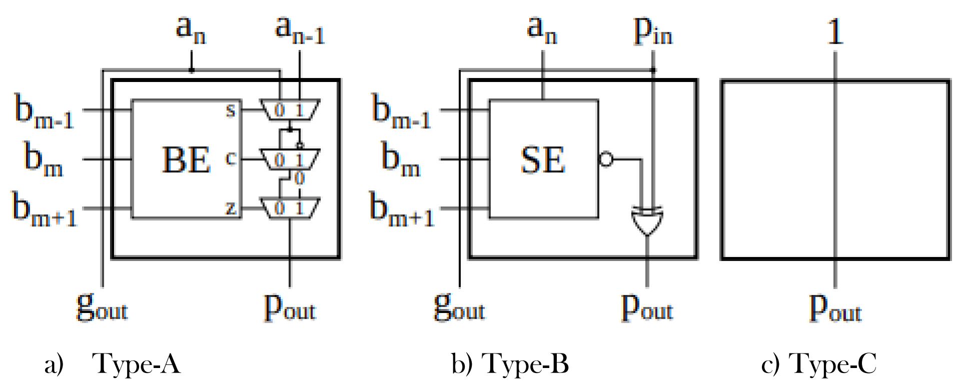


Fig. 2: Configuration of LUTs used in proposed design

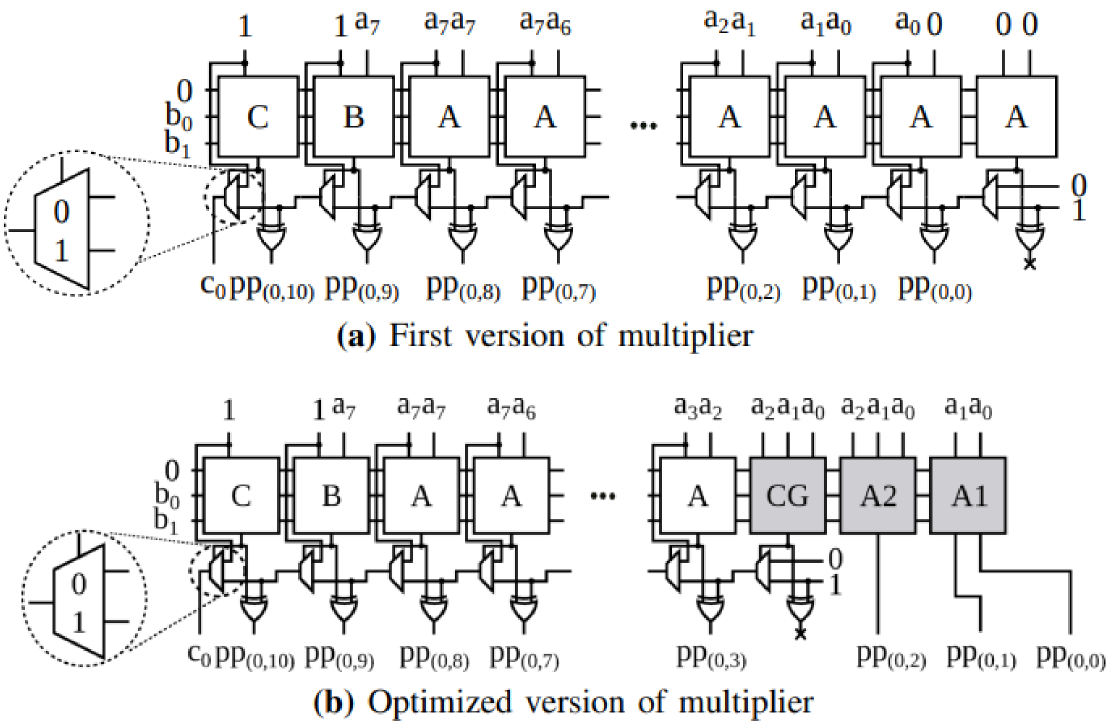


Fig. 3: First partial product row for an 8×8 multiplier

The Booth's encoding is implemented by LUT Type-A configuration, as shown in Fig. 2(a). It receives five inputs, i.e., an and an-1 (from multiplicand) and bm+1, bm and bm-1 (from multiplier). The LUT internally implements three MUXes. Based on the value of BE, the first MUX (controlled by s signal) decides whether an or an-1 should be forwarded for partial product generation. The second MUX, controlled by c signal, manages the inversion of the output of the first MUX. Finally, the third MUX can make the partial product zero depending upon the value of the z signal. This information is forwarded to the associated carry chain as carry propagate signal 'pout'. The input an is used as the carry generate signal 'gout' for the carry chain. The Bewick's sign extension technique for each partial product row is implemented by LUT Type-B and LUT TypeC configurations, as shown in Fig. 2(b) and 2(c) respectively. The LUT Type-B receives five inputs, i.e., bm+1, bm and bm-1 (from multiplier), an (the MSB of the multiplicand) and pin. The pin signal is constant '1' for the first row of partial products and for all other rows it is constant '0'. The LUT computes the SE signal, performs the XOR operation on it and provides the result to the associated carry chain as the carry propagate signal 'pout'. The carry generate signal 'gout' is directly provided by the pin signal. LUT Type-C is used to transfer the correct sign information of its respective partial product row to the following partial product row. Utilizing LUTs of types A, B and C, Fig. 3(a) shows the first row of partial products for an 8×8 multiplier. The rightmost LUT of Type-A in each partial product row is used for computing the required input carry. This input carry is applied for representing a partial product in 2's complement format. For an 8×8 multiplier, total four partial product rows will be generated. The last partial product row does not require a LUT of Type-C. For an N×M multiplier, the length of the carry chain in each partial product row is N+4 bits. To improve the critical path delay of the multiplier, the length of the carry chain can be reduced to N+1 bits. A critical path delayoptimized implementation of our novel multiplier is shown in Fig. 3(b). The partial product terms pp(x, 0) and pp(x, 1), in each partial product row, require one and two bits of multiplicand respectively. These two partial product terms can be implemented by one single 6-input LUT 'A1'. Similarly, pp(x, 2), in each partial product row, can be independently implemented using another 6-input LUT 'A2'. A separate 6- input LUT, 'CG', can be used to compute the correct input carry for each partial product row. Fig. 4 shows the internal configurations of LUT types A1, A2 and CG respectively.

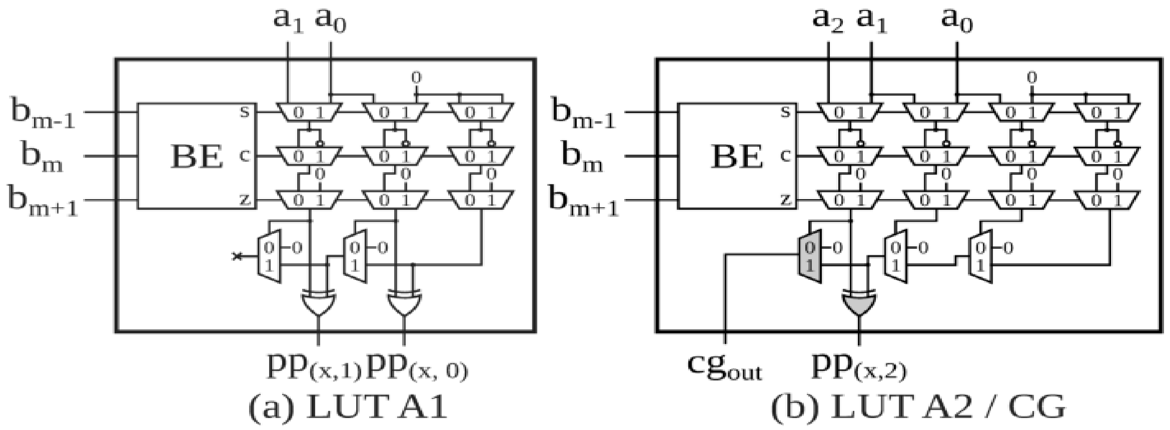


Fig. 4: Configuration of LUTs types A1, A2 and CG

For the reduction of generated partial products to compute the final product, binary adders, ternary adders and 4:2 compressors can be utilized. A 4:2 compressor is capable of reducing four partial product rows to two output rows. During our experiments, we observe that the deployment of ternary adders might reduce the overall resource utilization. However, they have higher critical path delays than the binary adders. Therefore, in our work, the 4:2 compressors and binary adders are used for the reduction of the generated partial products. We have used the 6-input LUTs and the associated carry chains to implement them.

RADIX-8 MODIFIED BOOTH ALGORITHM:

The Booth algorithm consists of repeatedly adding one of two predetermined values to a product P and then performing an arithmetic shift to the right on P.

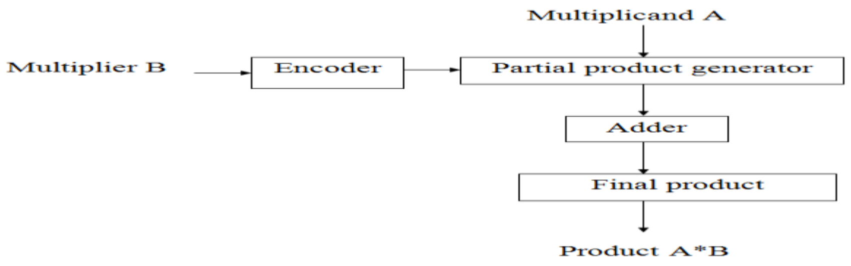


Fig5. Booth algorithm

The multiplier architecture consists of two architectures, i.e., Modified Booth. By the study of different multiplier architectures, we find that Modified Booth increases the speed because it reduces the partial products by half. Also, the delay in the multiplier can be reduced by using Wallace tree. The energy consumption of the Wallace Tree multiplier is also lower than the Booth and the array. The characteristics of the two multipliers can be combined to produce a high-speed and low-power multiplier. The modified stand-alone multiplier consists of a modified recorder (MBR). MBR has two parts, i.e., Booth Encoder (BE) and Booth Selector (BS). The operation of BE is to decode the multiplier signal, and the output is used by BS to produce the partial product. Then, the partial products are added to the Wallace tree adders, similar to the carry-save-adder approach. The last transfer and sum output line are added by a carry look-ahead adder, the carry being stretched to the left by positioning.

Table1 . Quartet coded signed-digit table

Quartet value	Signed-digit value
0000	0
0001	+1
0010	+1
0011	+2
0100	+2
0101	+3
0110	+3
0111	+4
1000	-4
1001	-3
1010	-3
1011	-2
1100	-2
1101	-1
1110	-1
1111	0

Here we have a multiplication multiplier, 3Y, which is not immediately available. To Generate it, we must run the previous addition operation: 2Y + Y = 3Y. But we are designing a multiplier for specific purposes and then the multiplier belongs to a set of previously known numbers stored in a memory chip. We have tried to take advantage of this fact, to relieve the radix-8 bottleneck, that is, 3Y generation. In this way, we try to obtain a better overall multiplication time or at least comparable to the time, we can obtain using a radix-4 architecture (with the added benefit of using fewer transistors). To generate 3Y with 21-bit words you just have to add 2Y + Y,ie add the number with the same number moved to a left position. A product formed by multiplying it with a multiplier digit when the multiplier has many digits. Partial products are calculated as intermediate steps in the calculation of larger products. The partial product generator is designed to produce the product multiplying by multiplying A by 0, 1, -1, 2, -2, -3, -4, 3, 4. Multiply by zero implies that the product is "0 ". Multiply by" 1 "means that the product remains the same as the multiplier. Multiply by "-1" means that the product is the complementary form of the number of two. Multiplying with "-2" is to move left one as this rest as per table.

SIGN EXTENSION CORRECTOR:

The Sign Extension Corrector is designed to increase the Booth multiplier capacity by multiplying not only the unsigned number but also the signed number. The principle of the sign extension that converts the signed multiplier not signed as follows. When unsign is signalled s_u = 0, it indicates the multiplication of the unsigned number and when s_u = 1, it shows the multiplication of the signed number. When a bit signal is called unsigned bit (s_u), it is indicated whether the multiplication operation is an unsigned number or number.

Table.5.Sign extension corrector

Sign-unsign	Type of operation
0 1	Unsigned multiplication Signed multiplication

Example:

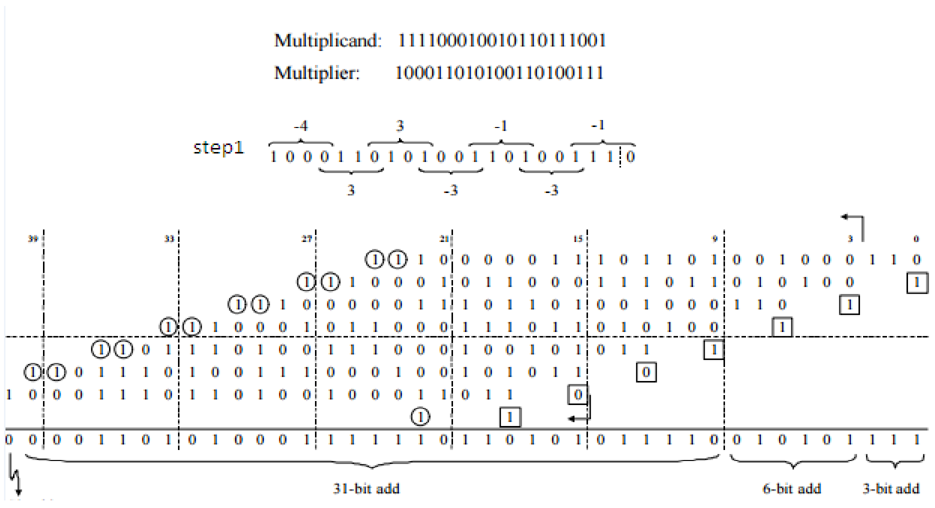
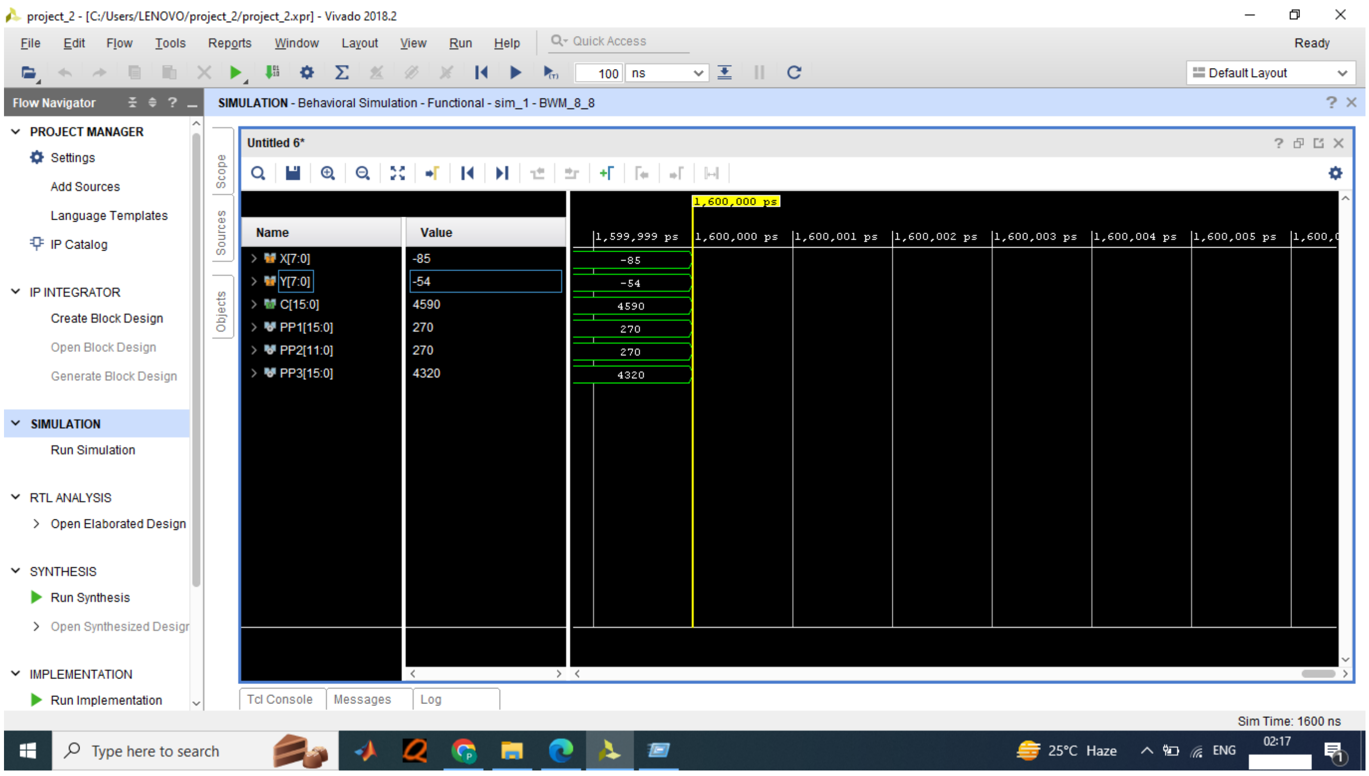

Multiplicand: 111100010010110111001
Multiplier: 100011010100110100111
step1
-4
3
-1
-1
3
-3
-3
31-bit add
6-bit add
3-bit add

Fig6. Example of modified booh algorithem

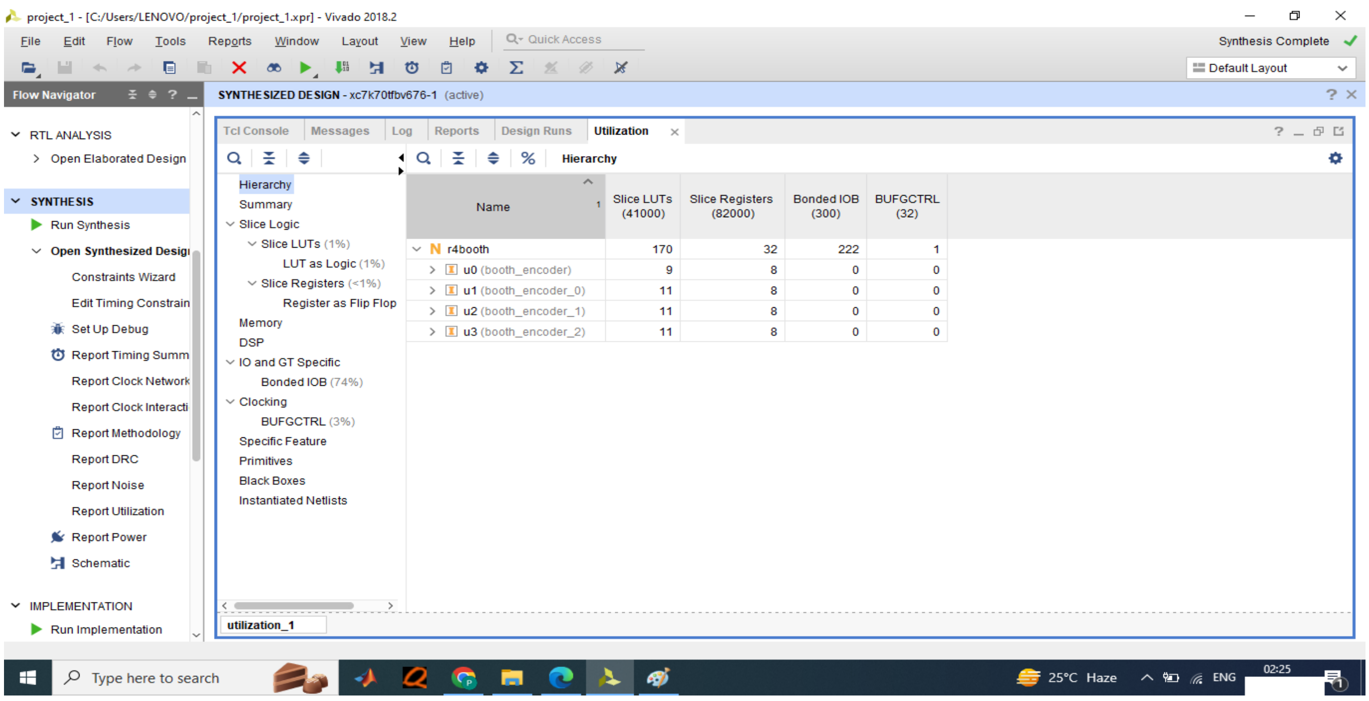
RESULTS:


project_2 - [C:/Users/LENOVO/project_2/project_2.xpr] - Vivado 2018.2
SIMULATION - Behavioral Simulation - Functional - sim_1 - BWM_8_8
Untitled 6*
Sim Time: 1600 ns

Name	Value
X[7:0]	-85
Y[7:0]	-54
C[15:0]	4590
PP1[15:0]	270
PP2[11:0]	270
PP3[15:0]	4320

Fig7: Proposed simulation result taken in XILINX VIVADO

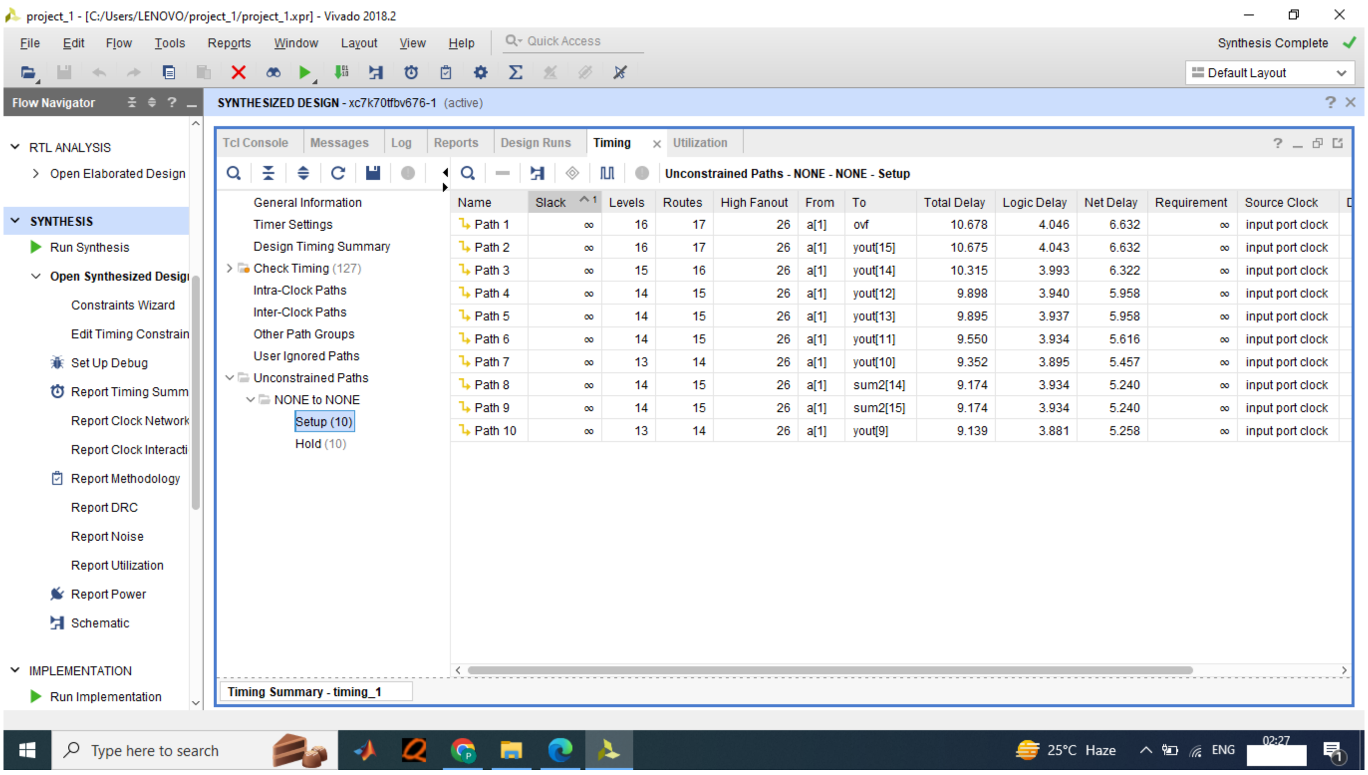
Above simulated screenshot taken in XILINX-VIVADO2018.2 for proposed method. Input 'X' is given as -85 and 'Y' is given as -54, final output 'C' is yielded as '4590' with only 3 partial products.


project_1 - [C:/Users/LENOVO/project_1/project_1.xpr] - Vivado 2018.2
SYNTHESIZED DESIGN - xc7k70tfbv676-1 (active)
Utilization
Hierarchy

Name	Slice LUTs (41000)	Slice Registers (82000)	Bonded IOB (300)	BUFGCTRL (32)
r4booth	170	32	222	1
u0 (booth_encoder)	9	8	0	0
u1 (booth_encoder_0)	11	8	0	0
u2 (booth_encoder_1)	11	8	0	0
u3 (booth_encoder_2)	11	8	0	0

Fig8: Existing design utilisation summary

Above synthesis results shown in screenshots represents area report of existing method

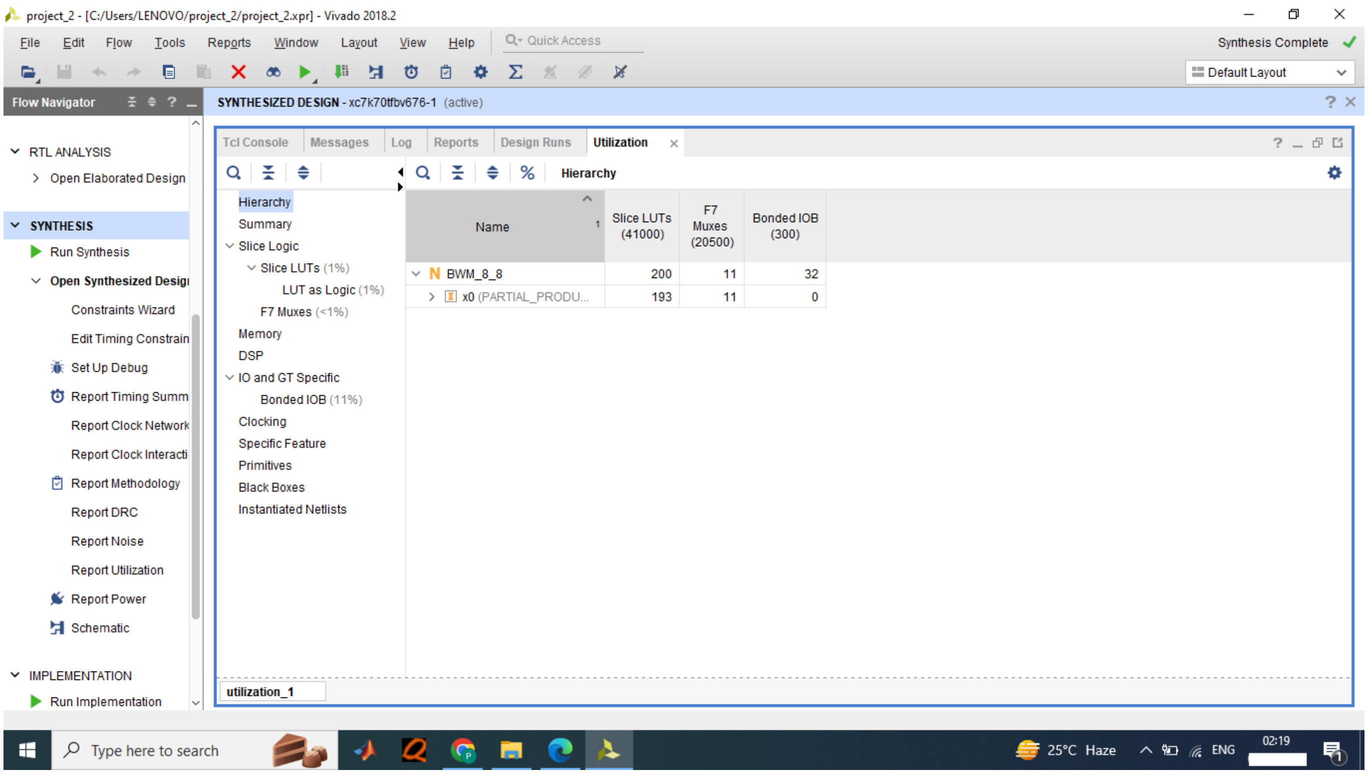


Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	16	17	26	a[1]	ovf	10.678	4.046	6.632	∞	input port clock
Path 2	∞	16	17	26	a[1]	yout[15]	10.675	4.043	6.632	∞	input port clock
Path 3	∞	15	16	26	a[1]	yout[14]	10.315	3.993	6.322	∞	input port clock
Path 4	∞	14	15	26	a[1]	yout[12]	9.898	3.940	5.958	∞	input port clock
Path 5	∞	14	15	26	a[1]	yout[13]	9.895	3.937	5.958	∞	input port clock
Path 6	∞	14	15	26	a[1]	yout[11]	9.550	3.934	5.616	∞	input port clock
Path 7	∞	13	14	26	a[1]	yout[10]	9.352	3.895	5.457	∞	input port clock
Path 8	∞	14	15	26	a[1]	sum2[14]	9.174	3.934	5.240	∞	input port clock
Path 9	∞	14	15	26	a[1]	sum2[15]	9.174	3.934	5.240	∞	input port clock
Path 10	∞	13	14	26	a[1]	yout[9]	9.139	3.881	5.258	∞	input port clock

Fig9: Existing timing summary

Above synthesis results shown in screenshots represents latency report of existing method

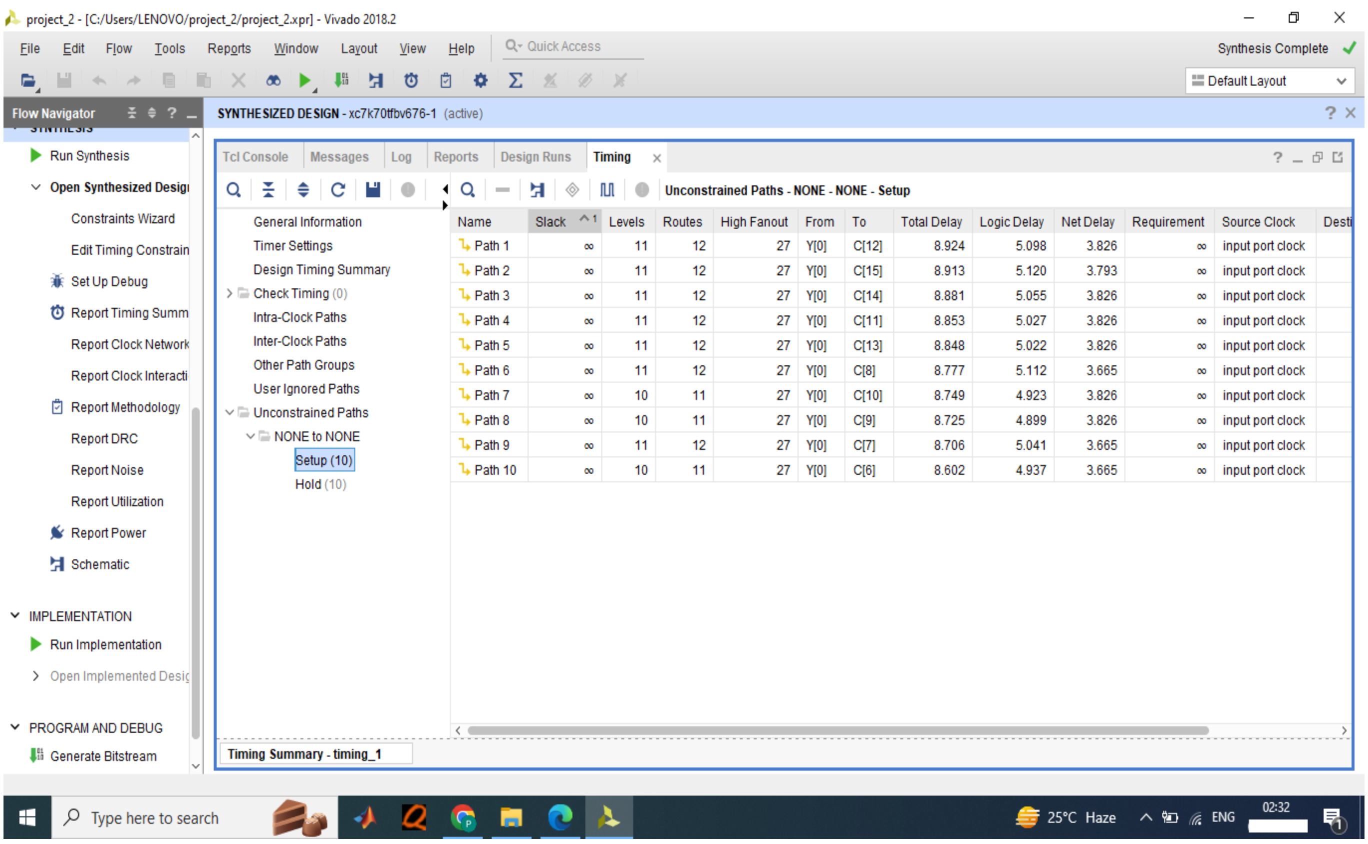


Hierarchy

Name	Slice LUTs (41000)	F7 Muxes (20500)	Bonded IOB (300)
BWM_8_8	200	11	32
x0 (PARTIAL_PRODU...	193	11	0

Fig10: Proposed design utilisation summary

Above synthesis results shown in screenshots represents area report of proposed method



project_2 - [C:/Users/LENOVO/project_2/project_2.xpr] - Vivado 2018.2

SYNTHESIZED DESIGN - xc7k70tfbv676-1 (active)

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Desti
Path 1	∞	11	12	27	Y[0]	C[12]	8.924	5.098	3.826	∞	input port clock	
Path 2	∞	11	12	27	Y[0]	C[15]	8.913	5.120	3.793	∞	input port clock	
Path 3	∞	11	12	27	Y[0]	C[14]	8.881	5.055	3.826	∞	input port clock	
Path 4	∞	11	12	27	Y[0]	C[11]	8.853	5.027	3.826	∞	input port clock	
Path 5	∞	11	12	27	Y[0]	C[13]	8.848	5.022	3.826	∞	input port clock	
Path 6	∞	11	12	27	Y[0]	C[8]	8.777	5.112	3.665	∞	input port clock	
Path 7	∞	10	11	27	Y[0]	C[10]	8.749	4.923	3.826	∞	input port clock	
Path 8	∞	10	11	27	Y[0]	C[9]	8.725	4.899	3.826	∞	input port clock	
Path 9	∞	11	12	27	Y[0]	C[7]	8.706	5.041	3.665	∞	input port clock	
Path 10	∞	10	11	27	Y[0]	C[6]	8.602	4.937	3.665	∞	input port clock	

Timing Summary - timing_1

Fig11: Proposed timing summary

Above synthesis results shown in screenshots represents latency report of proposed method

CONCLUSION and FUTURE SCOPE: Finally, this project presented a novel area-optimized, low-latency and energy-efficient accurate signed multiplier architecture for FPGA-based systems. This also evaluated the benefits of our multipliers in neural network applications. As an extension Radix-8 modified booth encoding with carry skip addition algorithm is proposed for efficient implementation of multiplier. The proposed approximation reduces the hardware for the Booth-encoded signals and lead to a design of a simpler partial product generator which is efficiently mapped using a single hardware architecture.

A clue of symmetric computation was developed around 2500 years ago which is brought out by Vedic mathematics. trigonometry, calculus, geometry and basic arithmetic are dealt by this symmetric computation. All these methods are very powerful as far as manual calculations are considered. The computational speed drastically reduces if all those methods are effectively used for the hardware implementation. Hence there is a chance for implementing a complete ALU using Vedic mathematics methods. Vedic mathematics is long been known but has not been implemented in the DSP and ADSP processors employing large number of multiplications in calculating the various transforms like FFTs and the IFFTs. By using these ancient Indian Vedic mathematics methods world can achieve new heights of performance and quality for the cutting edge technology devices.

REFRENCES:

[1] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [point of view]," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020.

[2] H. Pettenghi, F. Pratas, and L. Sousa, "Method for designing efficient mixed radix multipliers," *Circuits Syst. Signal Process.*, vol. 33, no. 10, pp. 3165–3193, 2014.

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

[4] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based approximate multipliers," in *Proc. 55th Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2018, pp. 1–6.

[5] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[6] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *Proc. 22nd Symp. Comput. Arithmetic (ARITH)*, Lyon, France, 2015, pp. 18–25.

[7] M. Langhammer and G. Baeckler, "High density and performance multiplication for FPGA," in *Proc. 25th Symp. Comput. Arithmetic (ARITH)*, Amherst, MA, USA, 2018, pp. 5–12.

[8] Y. Guo, H. Sun, and S. Kimura, "Small-area and low-power FPGAbased multipliers using approximate elementary modules," in *Proc. 25th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Beijing, China, 2020, pp. 599–604.

[9] Z. Ebrahimi, S. Ullah, and A. Kumar, "LeAp: Leading-one detectionbased softcore approximate multipliers with tunable accuracy," in *Proc. 25th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, Beijing, China, 2020, pp. 605–610.

[10] N. Van Toan and J. Lee, "FPGA-based multi-level approximate multipliers for high-performance error-resilient applications," *IEEE Access*, vol. 8, pp. 25481–25497, 2020.

[11] S. Ullah, H. Schmidl, S. S. Sahoo, S. Rehman, and A. Kumar, "Areaoptimized accurate and approximate softcore signed multiplier architectures," *IEEE Trans. Comput.*, vol. 70, no. 3, pp. 384–392, Mar. 2021, doi: 10.1109/TC.2020.2988404.

[12] B. S. Prabakaran, V. Mrazek, Z. Vasicek, L. Sekanina, and M. Shafique, "ApproxFPGAs: Embracing ASIC-based approximate arithmetic components for FPGA-based systems," 2020. [Online]. Available: arXiv:2004.10502.

[13] *7 Series FPGAs Configurable Logic Block, User Guide*, Xilinx, San Jose, CA, USA, 2016. [Online]. Available: https://www.xilinx.com/support/ documentation/userguides/ug4747SeriesCLB.pdf

[14] H. Jiang, J. Han, F. Qiao, and F. Lombard, "Approximate radix-8 booth multipliers for low-power and high-performance operation," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2638–2644, Aug. 2016.

[15] W. Liu, L. Qian, C. Wang, H. Jiang, J. Han, and F. Lombardi, "Design of approximate radix-4 Booth multipliers for error-toleran computing," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1435–1441, Aug. 2017.

[16] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 421–430, Mar. 2018.

[17] S. Venkatachalam, E. Adams, H. J. Lee, and S.-B. Ko, "Design and analysis of area and power efficient approximate booth multipliers," *IEEE Trans. Comput.*, vol. 68, no. 11, pp. 1697–1703, Nov. 2019.

[18] H. Waris, C. Wang, and W. Liu, “Hybrid low radix encoding-based approximate booth multipliers,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 12, pp. 3367–3371, Dec. 2020.

[19] B. S. Prabakaran *et al.*, “DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Dresden, Germany, Mar. 2018, pp. 917–920.

[20] H. Waris, C. Wang, W. Liu, J. Han, and F. Lombardi, “Hybrid partial product-based high-performance approximate recursive multipliers,” *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 4, 2020, doi: 10.1109/TETC.2020.3013977.