ISSN:2348-2079

Volume-6 Issue-1

International Journal of Intellectual Advancements and Research in Engineering Computations

Design of an optimized multiplier based on approximation logic

Dhivya Bharathi Rajendiran[1], Parimaladevi Muthusamy[2], Dr.Brindha Palanisamy[3]
[1]II Year M. E. –VLSI Design, [2,3]Assistant Professor, Velalar College Of Engineering and Technology, Erode-638012, Tamilnadu, India.

***Abstract*— Approximate computation decreases the design complexity of error rebounding applications. This paper deals with the approximation process for optimizing power, area and delay. The partial products of multiplication are altered based on their probability and approximated. Due to the use of probability in computation, complexity of partial product is reduced. The proposed two approximate designs are applied to 4-bit multiplier. Multiplier uses different adders like Ripple Carry Adder (RCA), and Carry Look Ahead adder (CLA). They have better precision when compared to the existing method. For the two approximate design power, area and delay constraints are compared for the multiplier with and without approximation. Multiplier designed with two approximate designs consumes low power when compared to the existing method. The proposed Multiplier 1 saves 45% & 55% of power than the existing and exact multiplier. VHDL coding is computed and the synthesis results are obtained by the Model Sim software. Power consumption, Delay and Logic Utilization are estimated using the Spartan 3E Xilinx kit.**

***Index Terms*— Approximate computation, Ripple Carry Adder (RCA), Carry Look Ahead adder (CLA).**

I. INTRODUCTION

Multiplication is a mathematical operation where an integer is added for a specified number of times. Many multimedia and Digital Signal Processing (DSP) applications concentrate on the multiplication; hence the power consumption and the performance of these systems are dominated by multipliers. Digital Image Processing (DIP) applications rely on multipliers to improve the quality of the image. Exact computing is not always necessary and hence they are replaced with the approximate computation. In these applications, adders and multipliers are the main component.

In [1], at the transistor level approximate full adders are designed and utilized in digital signal processing applications. The approximate computation is used in the partial product accumulation of multiplier. Approximate computation is implemented in Broken array multiplier [2] and mainly focus on the partial product accumulation were power consumption is the crucial factor. Truncation is employed in fixed-width multiplier design to reduce hardware complexity of multipliers. Quantization error is introduced by the truncation and it is compensated by a constant correction or variable correction term [3], [4]. In [5], technique for faster data multiplier is designed and it is achieved by using two design techniques: partitioning the partial product into two individual parts and hybrid adder is used for the final addition. The approximate data compressor for the multiplication is proposed in [6] which relay on different features of compression. It is used in partial product reduction network and has a major drawback that they produce non-zero output for zero input value which affects Mean Relative Error (MRE), the proposed approximate multiplier overcomes the drawback and provide better precision.

In [7] approximate computing process is performed in the partial product generation and alteration stage in order to achieve significant low power. The proposed multiplier it further reduces the power consumption of the multiplier. Wallace (1964) introduced the first column compression multiplier [8]. The partial products are reduced by grouping 3 rows and 2 rows using (3,2) counter and (2,2) counter respectively. Later Dadda (1965) altered the Wallace approach by replacing counters (3,2) and (2,2) in the critical delay path

of the multiplier [9]. In the Static Segment Multiplier (SSM) proposed in [10], m*n multiplication is performed instead of n*n multiplication (m<n). Using n-bit operands, m-bit segments are derived. Based on the modification in the Karnaugh map 2*2 approximate multiplier is proposed in [11]. Better precision and logic complexity mainly focus on the approximate adders and data compressors. In this paper, the partial products are generated and accumulated. The accumulated partial products are altered by introducing two different signals like propagate and generate signals with different probabilities. Probability statistical data gives the probability of error (P_{err}) for the altered partial products. Partial product reduction stage uses some arithmetic units like a half-adder, full-adder and data compressor. Finally, vector merge adder is used to compute final product output and hence the proposed approximate multiplier achieves low power.

II. PROPOSED ARCHITECTURE

The proposed multiplier architecture comprises of two different approximation designs.

Multiplier realization comprises of three steps:

1. Partial product generation
2. Reduction of partial product
3. Vector Merge addition

Final step gives the product output for the given input. The approximation is performed on the partial product reduction stage of the multiplier. Approximate computing decreases the design complexity for the error rebounding application. First approximate design deals with the multiplexer realization of half-adder, full-adder and uses different vector merge adders. In the second design, Sum, Carry expressions are approximated and the multiplier uses different vector merge adders like Ripple Carry Adder (RCA) and Carry Look Ahead adder (CLA) to get the product output. More power is consumed during the second step. Power, Delay and Logic Utilization constraints are compared with the two approximate multiplier designs.

A. Partial Product Generation

Partial Product generation is the first stage of the multiplier implementation. A 4-bit unsigned multiplier is considered for the approximation. Further, the approximation is extended for 8-bit unsigned operands.

Consider two 4-bit unsigned input operands a (multiplicand) and b (multiplier) where, $a= \sum^{3}_{m=0} a_m 2^m$ and $b= \sum^{3}_{n=0} b_n 2^n$. AND operation is performed to generate the partial product for the two operands a and b we get,

$$a_{m,n}=a_m \cdot b_n \quad (1)$$

Using (1), partial products are generated and accumulated. The Fig. 1 shows the accumulation of generated partial product. Table I. shows the AND operation of Dadda multiplier.

a3,3 a3,2 a3,1 a3,0 a2,0 a1,0 a0,0
a2,3 a1,3 a0,3 a0,2 a0,1
a2,2 a2,1 a1,1
a1,2

Fig. 1 Accumulation of generated partial product

TABLE I. AND OPERATION FOR DADDA MULTIPLIER

	$\alpha3$	$\alpha2$	$\alpha1$	$\alpha0$	
	$\alpha3\beta3$	$\alpha2\beta3$	$\alpha1\beta3$	$\alpha0\beta3$	$\beta3$
	$\alpha3\beta2$	$\alpha2\beta2$	$\alpha1\beta2$	$\alpha0\beta2$	$\beta2$
	$\alpha3\beta1$	$\alpha2\beta1$	$\alpha1\beta1$	$\alpha0\beta1$	$\beta1$
	$\alpha3\beta0$	$\alpha2\beta0$	$\alpha1\beta0$	$\alpha0\beta0$	$\beta0$

B. Alteration of Partial Product

From the statistical analysis, it is clear that the partial product probability is ¼ of being 1. The partial products are altered for the column containing more than three partial products. The partial products $a_{m,n}$ and $a_{n,m}$ are combined together to produce propagate and generate signals. Propagate and Generate signals are generated using (2) and (3), thus forming the altered partial product.

Propagate signal equation:

$$p_{m,n}=a_{m,n} + a_{n,m} \quad (2)$$

Generate signal equation:

$$g_{m,n}=a_{m,n} \cdot a_{n,m} \quad (3)$$

The generate signal ($g_{m,n}$) has the probability 1/16 of being 1, which is consequently lower than ¼ of $a_{m,n}$. The propagate signal ($p_{m,n}$) has the probability 1/16 + 3/16 + 3/16 = 7/16 of being 1, it is higher than ($g_{m,n}$). These probabilities are considered for the approximation process. Fig. 2 shows the transformation of partial product into the altered partial product.

a3,3 a3,2 a3,1 a3,0 a2,0 a1,0 a0,0
a2,3 a2,2 p2,1 a1,1 a0,1
g3,1 g3,0 g2,0
g2,1

Fig. 2 Transformation of partial product into altered partial product

C. Approximation of altered partial product g(m,n)

Generate signals are accumulated in column wise manner and has a probability 1/16 of being 1, in the same column two elements being one are decreased. The following conditions reveal the probability statistics of 4 generate signals in a column they are,

1) All the elements being 1 is $(1\text{-pr})^4$
2) Only one element being 1 is $4pr(1\text{-pr})^3$
3) Two elements being 1 is $6pr^2(1\text{-pr})^2$
4) Three elements being 1 is $4pr^3(1\text{-pr})$
5) All elements being 1 is pr^4 , where pr is 1/16.

Table II. shows the probability statistics of the generate signal. Based on the statistical table, OR gate is used instead of generate elements which are accumulated in each column. Probability of error (P_{err}) is also computed for the OR gate, which reduces the generate elements in each column are given in Table II. Error probability and error value increases as the number of generate signal increases, leads to the misprediction in probability. This is overcomed by grouping maximum number of generate elements using an OR gate. [m/4] OR gates are required for column having 'm' generate elements.

TABLE II. PROBABILITY STATISTICS OF GENERATE SIGNAL

M	Probability of the generate elements being				P_{err}
	All zero	One 1	Two 1's	Three 1's and more	
2	0.8789	0.1172	0.0039	-	0.00390
3	0.8240	0.1648	0.0110	0.00024	0.01124
4	0.7725	0.2060	0.0206	0.00093	0.02153

D. Approximation of other partial products using two designs

The accumulated other partial products $a_{m,n}$ and $p_{m,n}$ has probability ¼ and 7/16. For the approximation of other partial products two designs are proposed, which are carried out during the reduction stage. In the first design, half-adder and full-adder are approximated using multiplexer implementation. Finally, in vector addition step different adders are realized for the approximated multiplier. Second design, deals with the approximation of Sum and Carry expression for half-adder and full-adder. Different adders are realized in the final step.

1) Multiplexer Implementation

To estimate the critical path of the system adder plays an important role, which determines the overall performance of the system. Fundamental elements of the computer arithmetic circuits are half-adder and full-adder. NAND and NOR gates are called as the universal gates, because they can create any kind of logic gate and digital circuits. "Universal Logic" concept is used to create any kind of logic gates and digital circuits. Multiplexers and Decoders are considered as a "Universal Logic". In the proposed work, half-adder and full-adder are realized using multiplexer. Multiplexer is a circuit that accepts many inputs and produces one single output. It can handle both analog and digital data. For analog application, it uses relay and transistor switches and for digital application, it is built using standard logic gates. It is also called as "Data Selector". Multiplexer uses 2^n input signal and 1 output signal and hence it is called as (2^n:1) MUX. Where 'n' is considered to be control signal or selection input for the multiplier. Depending on the control signal and the given input multiplexer produces the desired output.

2) Realization of the half-adder using multiplexer

A Conventional half-adder is realized using two 2:1 multiplexer. One 2:1 multiplexer is used to generate Carry output and another 2:1 multiplexer is used to generate Sum output. 2:1 MUX requires 2 input signals, 1 output and 1 selection line (i.e) n=1.Fig. 3 shows the realization of the half-adder using multiplexer. Consider a and b are the two inputs of half adder. Multiplexer 1 that produces Carry output has one input a and other input always remains to '0'. While multiplexer 2 produce Sum output has one input a and other input is complement of a. Both the multiplexers have same selection line b. The following point shows the working of half-adder realized using multiplexers.

1) When a b = 00, MUX 1 produce 0 and MUX 2 produce 0.
2) When a b = 01, MUX 1 produce 0 and MUX 2 produce 1.
3) When a b = 10, MUX 1 produce 0 and MUX 2 produce 1.
4) When a b = 01, MUX 1 produce 1 and MUX 2 produce 0.

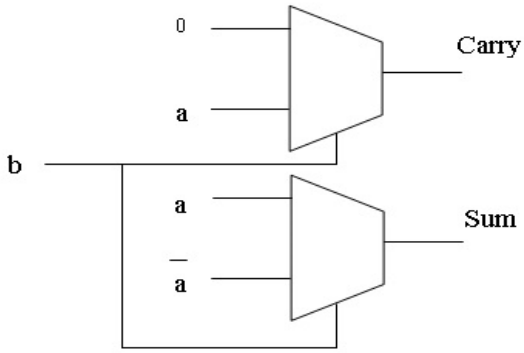

Fig. 3 Circuit diagram of multiplexer realized half-adder.

3) Realization of the full-adder using multiplexer

A Conventional full-adder is realized using two 2:1 multiplexers and an XOR gate. One 2:1 multiplexer is used to generate Carry output and another 2:1 multiplexer is used to generate Sum output. 2:1 MUX requires 2 input signals, 1 output and 1 selection line (i.e) n=1. XOR gate provides the input to the selection line. Fig. 4 shows the circuit diagram of multiplexer realized full-adder

Consider a, b and cin are the three inputs of full-adder. Multiplexer 1 that produces Sum output has one input to be a and other input is complement of a. While multiplexer 2 produce Carry output and has one input b and other input is a. b and cin inputs are given as input to the XOR gate and the output is fed as the selection line for both the multiplexers. The following are some example shows the working of multiplexer based full adder.

1) When a b cin = 000, xor output is 0, MUX 1 produce 0 and MUX 2 produce 0.
2) When a b cin = 011, xor output is 0, MUX 1 produce 0 and MUX 2 produce 1.
3) When a b cin = 111, xor output is 0, MUX 1 produce 1 and MUX 2 produce 1.

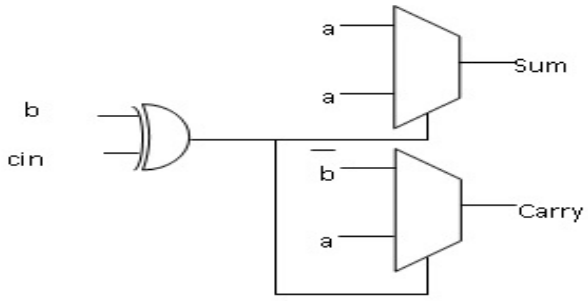

Fig. 4 Circuit diagram of multiplexer realized full-adder

E. Approximation of adders

In the second design, Sum and Carry expression half-adder and full-adder are approximated. Weight of carry is high and error in carry bit will produce error difference of about two in output. The Approximation is computed in such a way that the approximated output produces error difference as one.

In both full-adder and half-adder XOR gates are used where it contributes high delay and area. For half-adder approximation XOR gate in the sum expression is replaced by OR gate given in (4). While comparing the exact output and approximated output, one bit in sum differs from the exact output. Carry expression remains the same given in (5). Comparison table for approximated half-adder and full-adder is given in the Table III & IV. The tick mark denotes the matched outputs and cross mark denotes the mismatched output.

Sum=a or b (4)
Carry=a and b (5)

For full-adder approximation one XOR gate is replaced with another variable that uses or gate in the sum calculation. The variable is used for the carry calculation in order to number of OR & AND gates used. This results in error in two bits of the sum and one bit in the carry out of eight cases. Approximated Sum and Carry expressions are given in (7) and (8). Table IV. shows the approximated full adder truth table.

W= a or b (6)
Sum= W ⊕ cin (7)
Carry= W and cin (8)

TABLE III. TRUTH TABLE OF APPROXIMATED HALF-ADDER

Inputs		Exact outputs		Approximated outputs		Difference
A	B	Sum	Carry	Sum	Carry	
0	0	0	0	0✓	0✓	0
0	1	1	0	1✓	0✓	0
1	0	1	0	1✓	0✓	0
1	1	0	1	1✓	1×	1

TABLE IV. TRUTH TABLE OF APPROXIMATED FULL-ADDER

Inputs			Exact outputs		Approximate outputs		Difference
A	B	cin	Sum	Carry	Sum	Carry	
0	0	0	0	0	0✓	0✓	0
0	0	1	1	0	1✓	0✓	0
0	1	0	1	0	1✓	0✓	0
0	1	1	0	1	0✓	1✓	0
1	0	0	1	0	1✓	0✓	0
1	0	1	0	1	0✓	1✓	0
1	1	0	0	1	1×	0×	1
1	1	1	1	1	0✓	1×	1

F. Vector merge addition

During the vector merge addition stage the following vector merge adders are used,

1) Ripple Carry Adder (RCA)
2) Carry Look Ahead adder (CLA)

By realizing the accumulated Si and Ci into the various adders simulation outputs are verified. Power, delay and logic utilization are computed and compared for the multipliers that are approximated using two techniques and realized using various adders. Fig. 6 shows the architecture for 4x4 Dadda multiplier.

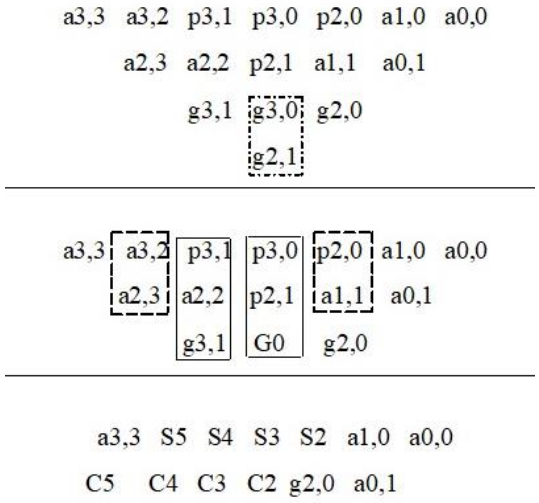

Fig. 5 Reduction tree of altered partial product

III. RESULTS AND DISCUSSION

All the multiplier designs (Multiplier1, Multiplier2, Multiplier3) that have been discussed above are designed in VHDL Language and simulation results are obtained using Model Sim software. The Fig. 7 shows the simulation of the proposed designed (Approximated Dadda multiplier implemented using Carry Look Ahead adder (CLA)). The Fig. 8 & 9 shows the simulation of multiplexer approximated Dadda multiplier using Ripple Carry Adder (RCA) and Carry Look Ahead adder (CLA).

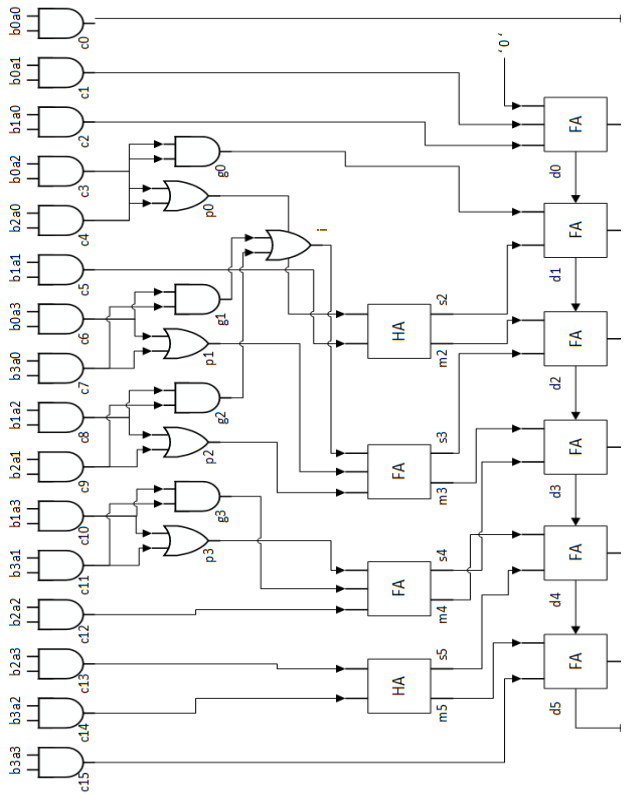

Fig. 6 Architecture of 4x4 Dadda multiplier

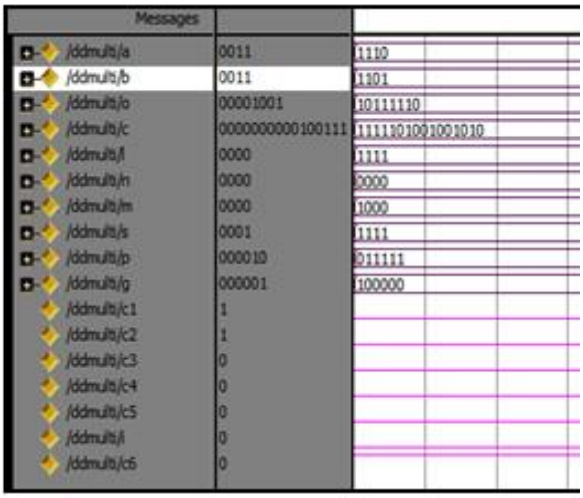

Fig. 7 Simulation waveform for 4x4 approximated dadda multiplier using CLA

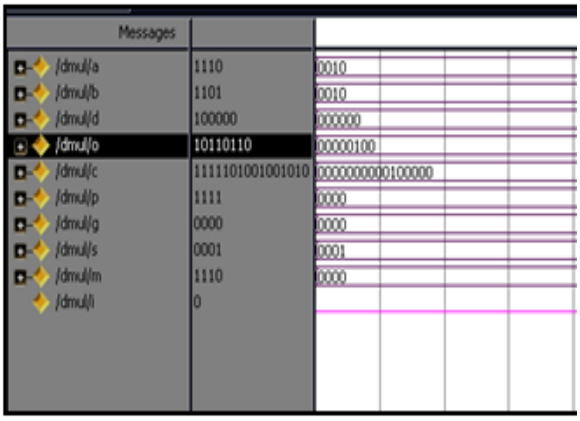

Fig. 8 Simulation waveform for 4x4 multiplexer approximated dadda multiplier using RCA

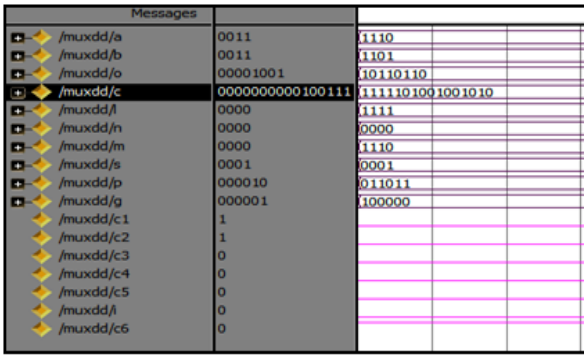

Fig. 9 Simulation waveform for 4x4 multiplexer approximated dadda multiplier using CLA

A. FPGA implementation using Spartan 3E

All the three multipliers are implemented on Spartan 3E FPGA using the Xilinx ISE design tool. Carry Look Ahead adder (CLA) consumes low power than the Ripple Carry Adder (RCA) but comparatively logic utilization (area) is higher than RCA.

TABLE V. PARAMETER COMPARISON OF DADDA MULTIPLIER

MULTIPLIER	POWER (mw)	LOGIC UTILIZATION	LOGIC DELAY (ns)	ROUTE DELAY
Exact Dadda multiplier	6822.28	28	9.418	4.947
Existing Dadda multiplier	5578.00	17	6.602	2.397
Multiplier1	3063.90	30	8.714	4.449
Multiplier2	4957.17	28	9.418	4.953
Multiplier3	5074.48	32	11.147	6.420

Table V. shows the parameter comparison for proposed multiplier with exact and existing multiplier. Exact (4x4) Dadda multiplier attains the power of about (6822.28mw), LUT (28), Logic delay (9.418ns) and Route delay (4.947ns). Existing (4x4) Dadda multiplier attains power of about (5578.00mw), LUT (17), Logic delay (6.602ns) and Route delay (2.397ns). Proposed multiplier1 attains the power of about (3063.90mw), LUT (30), Logic delay (8.714ns) and Route delay (4.449ns). Multiplier2 attains power of about (4957.17mw), LUT (28), Logic delay (9.418ns) and Route

delay (4.953ns). Multiplier3 attains power of about (5074.48mw), LUT (32), Logic delay (11.47ns) and Route delay (6.420ns).

Multiplier1 consume low power when compared to the existing multiplier and exact multiplier. Trade off occur between LUT count and delay. Multiplier1 attains lower delay than the exact multiplier, but consumes slightly more area. Multiplier2 and Multiplier3 also consume low power when compared to the existing multiplier and exact multiplier. LUT count and delay significantly varies when compared to existing and exact multiplier. Henceforth, multiplier1 consume low power than multiplier2 and multiplier 3. Multiplier1 is considered to be power efficient approximate Dadda multiplier and saves the power of about 45% & 55% than the existing and exact multiplier.

IV. CONCLUSION

In this brief an efficient approximate multiplier is proposed by modifying the partial products of the multiplier by generating propagate and generate signals. Generate signals are approximated using OR gates. Two different approximation techniques are applied to the multiplier. In the first design partial products are approximated during the partial product reduction stage. Then further after approximating the partial products the reduced values from the reduction stage are finally realized using Carry Look Ahead adder (CLA). While in the second design half-adder, full-adder are implemented using multiplexer and the partial products are reduced at the reduction stage. Finally outputs are obtained using vector merge adders like Ripple Carry Adder (RCA) and Carry Look Ahead adder (CLA). Power, delay and logic utilization are compared for the two approximated multipliers. Multiplier1 significantly consume low power than multplier2 and 3. The proposed multiplier1 is considered to be power efficient approximate multiplier.

REFERENCES

[1] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," IEEE Trans. Computer Aided Design Integr. Circuits Syst., vol. 32, no. 1, pp. 124-137, Jan. 2013.

[2] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 4, pp. 850-862, Apr. 2010.

[3] E. J. King and E. E. Swartzlander, Jr., "Data-dependent truncation scheme for parallel multipliers," in Proc. 31st Asilomar Conf. Signals, Circuits Syst., Nov. 1998, pp. 1178-1182.

[4] K. J. Cho, K. C. Lee, J. G. Chung, and K. K. Parhi, "Design of low-error fixed width modified booth multiplier," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 12, no. 5, pp. 522-531, May 2004.

[5] B. Ramkumar, V. Sreedeep, and Harish M Kittur, "A Design technique for faster dadda multiplier," Member, IEEE

[6] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," IEEE Trans. Comput., vol. 64, no. 4, pp. 984-994, Apr. 2015.

[7] Suganthi Venkatachalam and Seo-Bum Ko, "Design of power and area efficient approximate multipliers," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 25, no. 5, May 2017.

[8] C. S. Wallace, "A Suggestion for a fast multiplier," IEEE Trans. on Electronic Computers, vol. EC-13, pp. 14-17, 1964.

[9] Lugi Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349-356, August 1965.

[10] S. Narayanamoorthy, H.A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 6, pp. 1180-1184, June 2015.

[11] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," J. Low Power Electron., vol. 7, no. 4, pp. 490-501, 2011.

[12] V. G. Oklobdzija, D. Villeger, and S. S. Liu (1996), "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel multiplier using an Algorithmic Approach," IEEE Trans. on Computers, vol. 45, pp. 294-305, June 1996.

[13] Z. Wang, G. A. Jullien, and W. C. Miller, "A Design Technique for Column Compression Multipliers", IEEE Transactions on Computers, vol. 44, no. 8, pp. 962-970, August 1995.

[14] G. Zervaskis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," IEEE Trans. on Very Large Scale Integr. (VLSI) Syst., vol. 24, no. 10, pp. 3105-3117, Oct. 2016.