High-Performance and Resource-Efficient Squaring Architecture for FPGA Platforms

Burhan Khurshid

***Abstract*—Hardware-intensive signal processing has seen tremendous growth over the last few decades, owing to the advances in VLSI technology. This has resulted in a significant paradigm shift, wherein different computational functionalities are increasingly implemented using different hardware platforms. The squaring function is one such operation that finds its application in many signal-processing tasks. Since squaring is a specific case of multiplication, traditional multiplication algorithms can be adapted to create high-performance squaring architectures. In this paper, we present a squaring architecture that is based on the CORDIC algorithm. The hardware efficiency of the CORDIC algorithm enables it to compute different mathematical functions using only shift and add operations. By operating the algorithm in linear mode, the CORDIC computations can be modeled to emulate the squaring function. Our 8-bit CORDIC-based squaring architecture shows a 25% and 38% reduction in PDAP over the existing best design for ASIC and FPGA platforms, respectively.**

***Index Terms*— CORDIC algorithm, DSP, Fixed-point arithmetic, Pipelining, Vedic Mathematics.**

Original Research Paper
DOI: 10.53314/ELS2529053K

I. Introduction

Multiplication based functions like Multiply-Add (MAD), Multiply-Accumulate (MAC), Convolution, etc., form the core of many digital signal processing (DSP) applications [1], [2]. Image Processing, Neural Networks, Audio-Video Processing, and Machine Learning are a few of the many areas that have multiplication-inspired computations as an integral part of their functionality. The squaring function is one such operation that finds its application in many signal-processing tasks [3], [4]. The squaring operation can be seen as a specific instance of multiplication where the multiplicand and multiplier have the same value [5]. Evidently, most of the existing work related to squaring architectures is actually based on the optimization of the multiplication operation for the specific case of identical operands [6]. The performance bottleneck for a multiplier unit is typically limited by the partial product reduction (PPR) stage [7]. Traditionally, Wallace and Dadda trees have been used to reduce the partial product matrix (PPM) [8], [9]. Numerous modifications to these approaches have been proposed. Prominent among them are the approximation techniques that focus on designing compressors that reduce the complexity of the PPR stage. A 4:2 compressor-based multiplier based on the correlation of input data is presented in [10].

Approximate compressor trees with reduced accumulation layers are presented in [11], [12]. An approximate multiplier based on an inexact 4:2 compressor that shares the logic required for sum and carry is presented in [13]. Similarly, 4:2 compressor-based multipliers based on a constant approximation and probability-based error correction are presented in [14], [15]. Recursive approximate multipliers based on the divide and conquer strategy are presented in [16]. These show substantial improvements in error metrics when compared to traditional compressor-based multipliers. For field programmable gate array (FPGA) based designs, generalized parallel counters (GPC) are used to construct the compressor tress [17]-[19]. These map well on FPGAs and are often used to replace traditional Wallace and Dadda trees. However, given that squaring is a specific type of multiplication where both operands are identical, the number of partial products generated is generally fewer [5]. This reduces the computational complexity of the PPR stage. Evidently, a squaring architecture is often much faster and consumes fewer resources compared to an equivalent multiplier [20].

Recently, Vedic mathematics has been used to improve the performance of multiplier architectures in general [21], [22] and squaring architectures in specific [5], [6], [20], [23], [24]. In [23], a squaring architecture based on the duplex property of *Dwandwayoga* sutra is proposed and implemented using 14nm FinFET technology. A squaring architecture based on the *Urdhva Tiryagbhyam* sutra from Vedic mathematics is proposed in [24]. The concept is further modulated in [5], wherein the authors use a combination of *Urdhva Tiryagbhyam* sutra and *Karatsuba-Ofman* algorithm to design a recursive squaring architecture. In [20], the authors present a square architecture based on the *Anurupye* sutra of Vedic mathematics. The architecture is implemented on Kintex-7 FPGA and reports a subsequent improvement in performance over some earlier Vedic-based designs [24], [25]. However, it fails to match the performance of the design reported in [5]. The authors in [26] report a squaring architecture based on the *Ekadhikena Purvena* sutra of Vedic mathematics. The resulting structure is multiplier-less but fails to match the performance of designs in [5] and [20]. Similar squaring architectures based on the *Yavadunam* sutra of Vedic mathematics are reported in [6], [27].

Manuscript received on August 22nd, 2024. Received in revised form on January 6th and February 11th, 2025. Accepted for publication on March 10th, 2025.

Burhan Khurshid is with the Department of Electronics & Communication Engineering, National Institute of Technology Srinagar, J&K, India (e-mail: burhan@nitsri.ac.in).

This paper proposes a novel approximate squaring architecture based on the CORDIC algorithm. In the past, CORDIC-based computations have been employed to calculate various trigonometric and non-linear transcendental functions. We operate the CORDIC algorithm in linear mode and modify the computations to evaluate the squaring operation. We also conduct a Pareto analysis for an 8-bit CORDIC-based squaring architecture to determine the optimum number of stages that justify the accuracy-performance trade-offs for the proposed architecture.

The paper proceeds with the following structure: Section II briefly discusses the basics of the CORDIC algorithm. Squaring architecture based on CORDIC is presented in Section III. Error analysis is done in Section IV. Synthesis, implementation, and performance comparison are carried out in Section V. Section VI concludes the paper and points out any scope for future work. References are listed at the end.

II. Cordic Algorithm

CORDIC stands for Coordinate Rotation Digital Computer. The algorithm iteratively rotates a vector through arbitrary angles within linear or non-linear coordinate systems [28]-[30]. The basic rotation equations are given as [31]:

$$\begin{bmatrix}\mathbf{x}_m\\ \mathbf{y}_m\end{bmatrix}=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}\mathbf{x}_0\\ \mathbf{y}_0\end{bmatrix} \tag{1}$$

$(\mathbf{x}_0, \mathbf{y}_0)$ are the starting coordinates of the vector, and $(\mathbf{x}_m, \mathbf{y}_m)$ are the final coordinates of the vector. Since its induction, different algorithm modifications have been proposed to enable the algorithm to calculate a wider range of mathematical functions [32]. The basic operating modes of the algorithm are the rotation and the vectoring mode. While the former is characterized by the reduction of the value of the residual angle with every iteration [33], the latter performs a finite number of micro-rotations to compute the angle that aligns the final vector parallel to the x-axis. A single set of iterative unified CORDIC equations that encapsulate both modes is given as [34]:

$$\begin{bmatrix}\mathbf{x}_{i+1}\\ \mathbf{y}_{i+1}\end{bmatrix}=\mathbf{k}_i\begin{bmatrix}1 & -\mu\boldsymbol{\sigma}_i 2^{-i}\\ \boldsymbol{\sigma}_i 2^{-i} & 1\end{bmatrix}\begin{bmatrix}\mathbf{x}_i\\ \mathbf{y}_i\end{bmatrix} \tag{2}$$

$$\mathbf{z}_{i+1}=\begin{cases}\mathbf{z}_i-\boldsymbol{\sigma}_i\tanh^{-1}(2^{-i}), & \mu=-1\\ \mathbf{z}_i-\boldsymbol{\sigma}_i(2^{-i}), & \mu=0\\ \mathbf{z}_i-\boldsymbol{\sigma}_i\tan^{-1}(2^{-i}), & \mu=1\end{cases} \tag{3}$$

Where μ = -1, 0, and 1 for hyperbolic, linear, and circular coordinates, respectively.

$$\boldsymbol{\sigma}_i=\begin{matrix}\text{sign}(\mathbf{z}_i),\text{for rotation mode}\\ -\text{sign}(\mathbf{y}_i),\text{for vectoring mode}\end{matrix}$$

i and *n* represent the current and the total number of iterations, respectively. The rotations are pseudo rotations and tend to increase the length of the rotating vector with every iteration. The gain factor $\boldsymbol{k}_i$ is, therefore, used to maintain the original length of the rotating vector. k_i is calculated as:

$$\mathbf{k}_i=\prod_{i=0}^{n-1}\sqrt{1+\mu 2^{-2i}} \tag{4}$$

III. Proposed Squaring Architecture

CORDIC computations can be modeled to emulate the squaring operation by performing rotations in a linear coordinate system. From equations (2) and (3), we have:

$$\mathbf{x}_{i+1}=\mathbf{k}_i[\mathbf{x}_i-\mu\boldsymbol{\sigma}_i 2^{-i}\mathbf{y}_i]$$

$$\mathbf{y}_{i+1}=\mathbf{k}_i[\boldsymbol{\sigma}_i 2^{-i}\mathbf{x}_i+\mathbf{y}_i]$$

$$\mathbf{z}_{i+1}=\mathbf{z}_i-\boldsymbol{\sigma}_i 2^{-i}$$

For a linear coordinate system, μ = 0. The value of the gain factor $\boldsymbol{k}_i$ will, therefore, be unity. The above equations will be modified as:

$$\mathbf{x}_{i+1}=\mathbf{x}_i$$

$$\mathbf{y}_{i+1}=\mathbf{y}_i+\boldsymbol{\sigma}_i 2^{-i}\mathbf{x}_i$$

$$\mathbf{z}_{i+1}=\mathbf{z}_i-\boldsymbol{\sigma}_i 2^{-i}$$

Or

$$\begin{bmatrix}\mathbf{x}_{i+1}\\ \mathbf{y}_{i+1}\end{bmatrix}=\begin{bmatrix}1 & 0\\ \boldsymbol{\sigma}_i 2^{-i} & 1\end{bmatrix}\begin{bmatrix}\mathbf{x}_i\\ \mathbf{y}_i\end{bmatrix} \tag{5}$$

$$\mathbf{z}_{i+1}=\mathbf{z}_i-\boldsymbol{\sigma}_i 2^{-i} \tag{6}$$

The above equations are iterative. Note that depending upon the direction of rotation $\boldsymbol{\sigma}_i$ may be either +1 or -1. If the initial values of **x**, **y**, and **z** vectors are $\mathbf{x}_s$, $\mathbf{y}_s$, and $\mathbf{z}_s$, respectively, then the equations after the first iteration (i = 0) would be:

$$\mathbf{x}_1=\mathbf{x}_s$$

$$\mathbf{y}_1=\mathbf{y}_s\pm\mathbf{x}_s$$

$$\mathbf{z}_1=\mathbf{z}_s\mp 1$$

After second iteration (i = 1):

$$\mathbf{x}_2=\mathbf{x}_1=\mathbf{x}_s$$

$$\mathbf{y}_2=\mathbf{y}_1\pm 2^{-1}\mathbf{x}_1$$
$$=\mathbf{y}_s\pm\mathbf{x}_s\pm 2^{-1}\mathbf{x}_s$$

$$\mathbf{z}_2=\mathbf{z}_1\mp 2^{-1}$$
$$=\mathbf{z}_s\mp 1\mp 2^{-1}$$

After third iteration (i = 2):

$$\mathbf{x}_3=\mathbf{x}_2=\mathbf{x}_1=\mathbf{x}_s$$

$$\mathbf{y}_3=\mathbf{y}_2\pm 2^{-2}\mathbf{x}_2$$
$$=\mathbf{y}_s\pm\mathbf{x}_s\pm 2^{-1}\mathbf{x}_s\pm 2^{-2}\mathbf{x}_s$$

$$\mathbf{z}_3=\mathbf{z}_2\mp 2^{-2}$$
$$=\mathbf{z}_s\mp 1\mp 2^{-1}\mp 2^{-2}$$

After *m* iterations:

$$\mathbf{x}_m=\mathbf{x}_{m-1}\ldots=\mathbf{x}_2=\mathbf{x}_1=\mathbf{x}_s$$

$$\mathbf{y}_m=\mathbf{y}_{m-1}\pm 2^{-m+1}\mathbf{x}_{m-1}$$
$$=\mathbf{y}_s\pm\mathbf{x}_s\pm 2^{-1}\mathbf{x}_s\pm 2^{-2}\mathbf{x}_s\pm\cdots\pm 2^{-m+1}\mathbf{x}_s$$
$$=\mathbf{y}_s\pm\mathbf{x}_s[1\pm 2^{-1}\pm 2^{-2}\pm\cdots\pm 2^{-m+1}]$$

$$\mathbf{z}_m=\mathbf{z}_{m-1}\mp 2^{-m+1}$$

$$= \mathbf{z}_s \mp 1 \mp 2^{-1} \mp 2^{-2} \mp \cdots \mp 2^{-m+1}$$

Since the algorithm is being operated in rotation mode after m iterations, the value of the residual angle ($\mathbf{z}_m$) will be extremely small and may be equated to 0. Therefore, after *m* iterations:

$$\mathbf{z}_m = 0$$

$$\mathbf{z}_s \mp 1 \mp 2^{-1} \mp 2^{-2} \mp \cdots \mp 2^{-m+1} = 0$$

Or

$$\mathbf{z}_s = 1 \pm 2^{-1} \pm 2^{-2} \pm \cdots \pm 2^{-m+1}$$

Or

$$\mathbf{y}_m = \mathbf{y}_s \pm \mathbf{x}_s \mathbf{z}_s$$

The final equations after *m* iterations would, therefore, be:

$$\mathbf{x}_m = \mathbf{x}_s \quad (7)$$

$$\mathbf{y}_m = \mathbf{x}_s \mathbf{z}_s + \mathbf{y}_s \quad (8)$$

$$\mathbf{z}_m = 0 \quad (9)$$

Equivalently, the above set of equations may be written as:

$$\begin{bmatrix} \mathbf{x}_m \\ \mathbf{y}_m \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \mathbf{z}_s & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_s \\ \mathbf{y}_s \end{bmatrix} \quad (10)$$

$$\mathbf{z}_m = 0 \quad (11)$$

In equation (8), if $\mathbf{x}_s = \mathbf{z}_s$, and $\mathbf{y}_s = 0$, then $\mathbf{y}_m$ will compute the square of the input $\mathbf{x}_s$. A direct mapping of the set of equations in (5) and (6) will result in an iterative (folded) architecture, as shown in Fig. 1. The iterative architecture is sequential and incurs a lot of latency, which is unsuitable for high-speed DSP applications. The iterative architecture can be unfolded to map each iteration on a separate computing stage. This results in a parallel structure, as shown in Fig. 2. The parallel architecture can be easily pipelined by placing registers after every computation stage. Thus, a sequential iterative architecture is converted into a combinational (pipelined) feed-forward architecture.

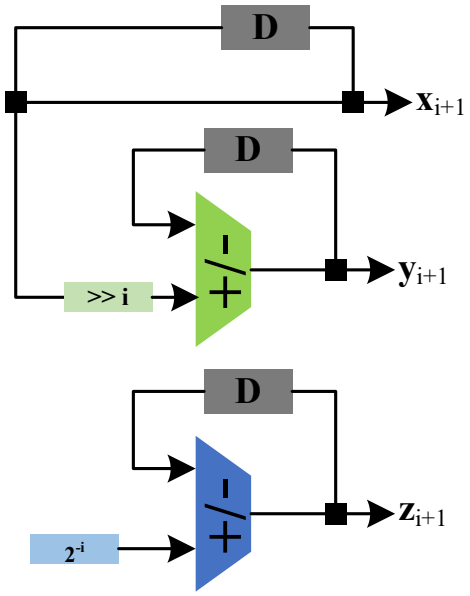

Fig. 1. Sequential CORDIC-based Square architecture

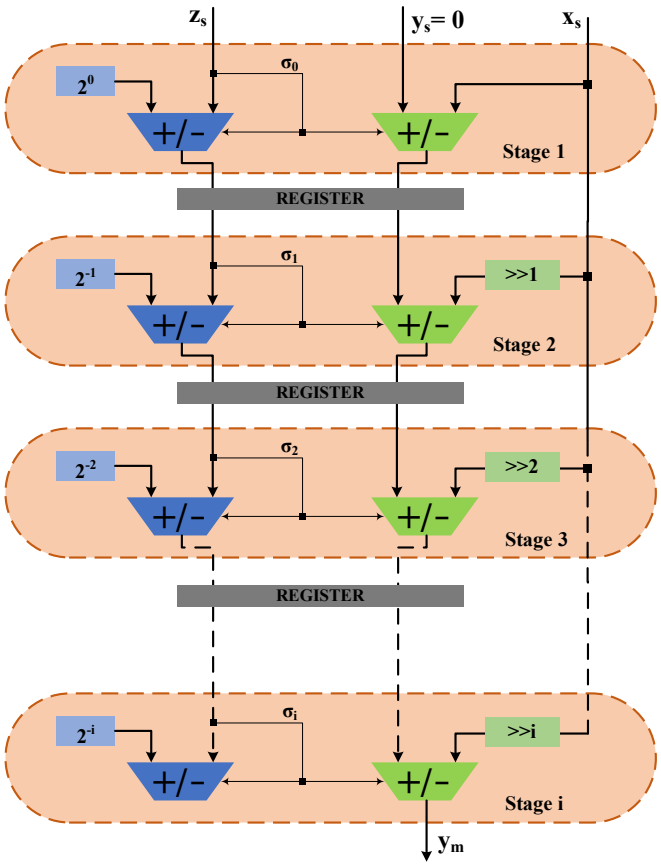

Fig. 2. Unfolded CORDIC-based Square architecture

The square architecture of Fig. 2 is based on simple shift and add operations. For larger magnitude operands, the algorithm will require a lot of stages to converge to an acceptable level of accuracy. This will incur a lot of hardware costs and a degradation in performance and accuracy. To avoid this, the input is scaled down such that its value lies in the range [0,1). This is done by scaling the **x** vector by 2^n. The new input is thus represented in fixed-point 2's complement format as:

$$\mathbf{x}'_s = \mathbf{x}'_n \,.\, \mathbf{x}'_{n-1}\, \mathbf{x}'_{n-2}\, \mathbf{x}'_{n-3} \ldots\ldots\, \mathbf{x}'_1\, \mathbf{x}'_0 \quad (12)$$

The value of the input is given as:

$$\mathbf{x}'_s = \sum_{i=1}^{n} \mathbf{x}'_{n-i} 2^{-i} \quad (13)$$

The result is then obtained by scaling the output from the last stage by 2^{2n}. Note that CORDIC-based computations are only approximate. The number of iterations will, therefore, determine the accuracy of the final result. We, therefore, conducted a detailed Pareto analysis to determine the number of optimal stages that will justify the accuracy-performance trade-offs. Our analysis considers the 8-bit CORDIC square architecture. The added advantage of similar inputs in the squaring operation lowers the permutations that can exist for the inputs. A total of 256 input combinations exist for an 8-bit square architecture, and all such combinations are considered Pareto points. The Pareto analysis focuses on the variation of error with the number of computation stages. Specifically, we have used error rate (ER) and mean error distance (MED) to quantify the error

in our computations. ER is defined as the percentage of outputs in error. MED is defined as the average of error distance (ED) values for a set of input-output samples. Mathematically,

$$\text{MED} = \frac{1}{\text{N}}\sum_{\text{i}=1}^{\text{N}}|\text{ED}_\text{i}| \quad (14)$$

Where N is the number of patterns. The results of the Pareto analysis are presented in Figs. 3 and 4, where the MED and ER for all the input combinations are plotted as a function of the number of CORDIC stages, respectively. A MED of 7.8 is achieved with a CORDIC square architecture designed in 8 stages; thereafter, any increase in the number of stages does not result in any significant reduction in MED. In fact, as the number of stages is further increased, the correctly obtained outputs are subjected to further computations, thereby introducing errors in the results. This is evident from Fig. 3, where the MED slightly increases after 10 stages due to over-computation of the results. Similarly, an ER of 5% is achieved with a CORDIC square architecture designed in 8 stages, which signifies that 95% of results are accurate. Based on the fixed-point representation of equation (12) and the results from the Pareto analysis, an illustration of the stage-wise computations for the CORDIC squarer is presented in Table I. The modified square structure is shown in Fig. 5.

TABLE I
STAGE-WISE COMPUTATIONS FOR AN 8-BIT CORDIC SQUARE ARCHITECTURE.

Initial Inputs	$= \mathbf{01101110_2}$ $= \mathbf{110_{10}}$	$= \mathbf{00000000_2}$ $= \mathbf{0_{10}}$	$= \mathbf{01101110_2}$ $= \mathbf{110_{10}}$
Inputs and scaled down by 2^8	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.00000000_2$ $= 0.0_{10}$	$= 0.01101110_2$ $= 0.4296875_{10}$
1st Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.00000000_2 + 0.01101110_2$ $= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.01110001_2 - 1.00000000_2$ $= 1.01101110_2$ $= -0.570313_{10}$
2nd Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.011011100_2 - 0.001101110_2$ $= 0.001101110_2$ $= 0.21484375_{10}$	$= 1.01101110_2 + 0.10000000_2$ $= 1.11101110_2$ $= -0.070313_{10}$
3rd Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.0011011100_2 - 0.0001101110_2$ $= 0.0001101110_2$ $= 0.107421875_{10}$	$= 1.11101110_2 + 0.01000000_2$ $= 0.00101101_2$ $= 0.179687_{10}$
4th Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.00011011100_2 + 0.00001101110_2$ $= 0.00101001010_2$ $= 0.1611328125_{10}$	$= 0.00101101_2 - 0.00100000_2$ $= 0.00001101_2$ $= 0.054687_{10}$
5th Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.001010010100_2 + 0.000001101110_2$ $= 0.001100000010_2$ $= 0.18798828125_{10}$	$= 0.00001101_2 - 0.00010000_2$ $= 0.00000010_2$ $= -0.007813_{10}$
6th Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.0011000000100_2 - 0.0000001101110_2$ $= 0.0010110010110_2$ $= 0.174560546875_{10}$	$= 0.00000010_2 + 0.00001000_2$ $= 0.00000101_2$ $= 0.023437_{10}$
7th Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.00101100101100_2 + 0.00000001101110_2$ $= 0.00101110011010_2$ $= 0.1812744140625_{10}$	$= 0.00000101_2 - 0.00000100_2$ $= 0.00000001_2$ $= 0.007812_{10}$
8th Stage	$= 0.01101110_2$ $= 0.4296875_{10}$	$= 0.001011100110100_2 + 0.000000001101110_2$ $= 0.001011110100010_2$ $= 0.18463134765625_{10}$	$= 0.00000001_2 - 0.00000010_2$ $= 1.11111111_2$ $= -0.0000005_{10}$
Multiplier Output after 8 stages (Bit adjusted and scaled up by 2^{16})		$= \mathbf{00010111101000100_2}$ $= \mathbf{12100_{10}}$	

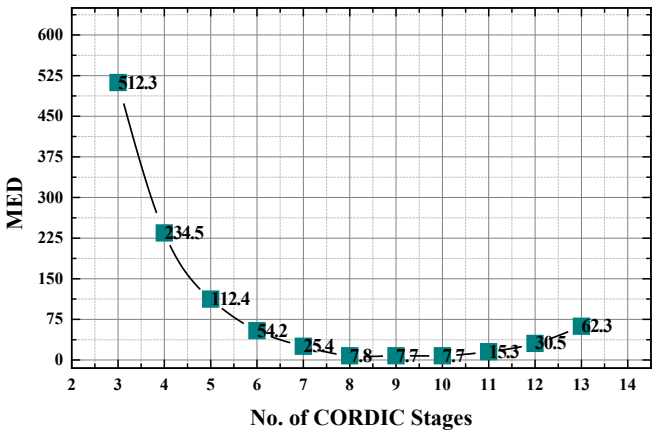

Fig. 3. Variation of MED with the number of computation stages for CORDIC-based 8-bit Square architecture.

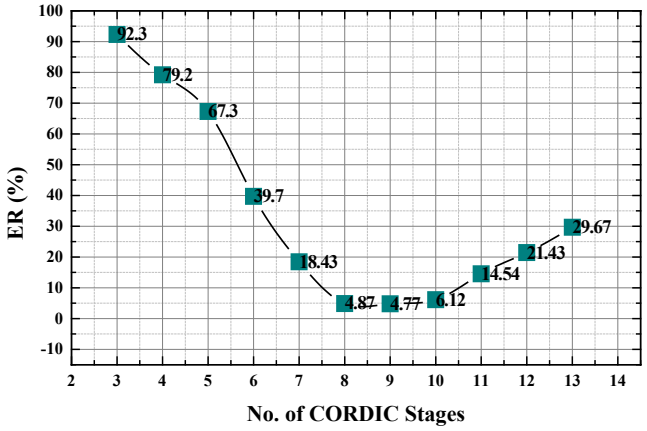

Fig. 4. Variation of ER with the number of computation stages for CORDIC-based 8-bit Square architecture.

IV. Error Analysis

The accuracy of the proposed architecture is compared against some approximate multiplier-based square architectures in terms of MED and normalized mean error distance (NMED). MED is given by equation (14). NMED is computed from MED as:

$$\text{NMED} = \frac{\text{MED}}{R_{max}} \tag{15}$$

Where R_{max} is the maximum exact result of the computation. The error metrics are computed using the Vivado 23.1 simulator. The input patterns are provided as an input text file, and the calculated output of different approximate designs is also written to a separate text file. The computed outputs are then compared to the exact outputs for each pattern, and ED for each pattern is calculated. MED is then computed as the average of EDs. Table II provides a comparison of different 8-bit square architectures. It is observed that the CORDIC-based square architecture has the least MED and NMED among the various designs.

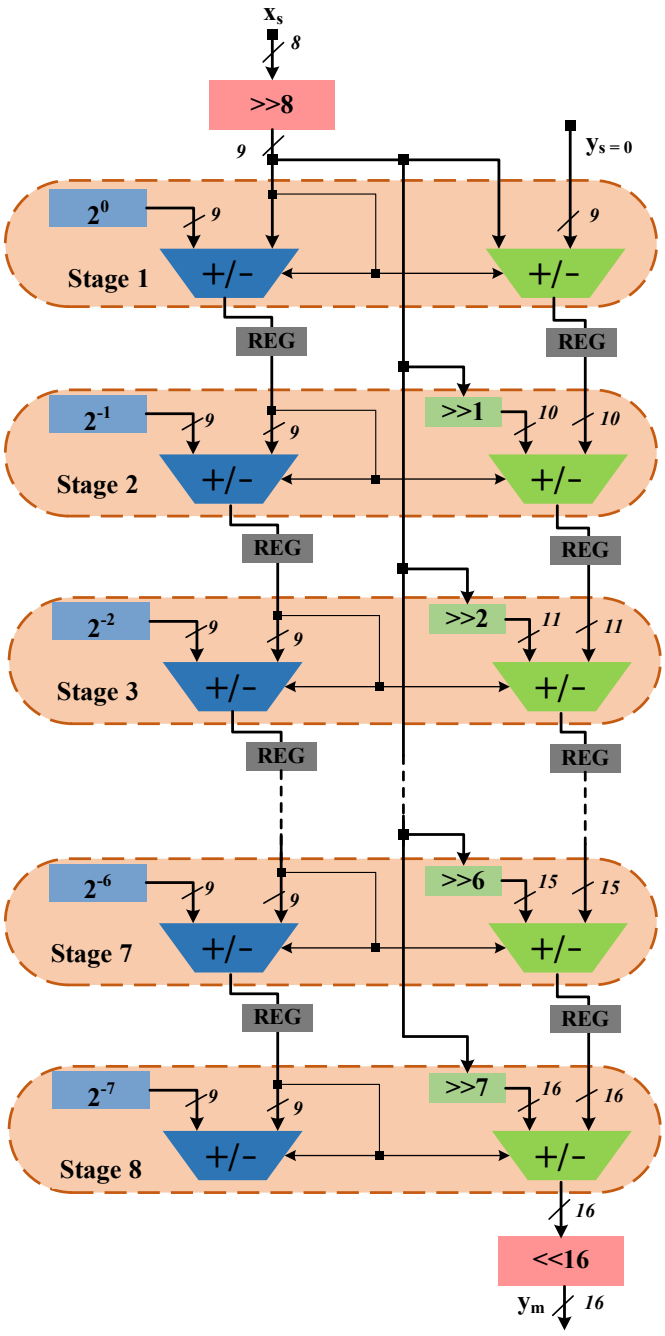

Fig. 5. CORDIC-based Square architecture with scaled inputs

TABLE II
Accuracy Analysis of Different Square Architectures for 8-bit Operands

Design	MED	NMED
[11]	21.8	0.00034
MCom 4:2 [12]	204.1	0.00314
MFA1 [12]	67.45	0.00104
MFA2 [12]	45.12	0.0007
[13]	137.3	0.0021
Normal [14]	16.4	0.00025
Hybrid [14]	11.3	0.00017
MUL1 [15]	121.32	0.00186
MUL2 [15]	115.7	0.00178
RRAM-I [16]	72.8	0.00112
RRAM-II [16]	9.32	0.00014
Proposed	7.8	0.00012

V. Synthesis & Implementation

The proposed square architecture is implemented both on ASIC and FPGA platforms. The ASIC implementation is based on the TSMC 180 nm CMOS technology using the Synopsys design tool. The simulations are conducted at an operating voltage of 1.8 V with an operating temperature of 25º C and a target clock frequency of 100 MHz. The area, critical path delay, and dynamic power dissipation are reported for each design. For FPGA implementation, the architecture in Fig. 5 is coded using VHDL and implemented using Xilinx Vivado 23.1, targeting the 7th generation xc7k325tffg900c FPGA device from the Kintex-7 family. The analysis involves resources, timing, and dynamic power dissipation as parameters of interest. Implementation is done under a constrained environment for both ASIC and FPGA platforms, with both physical and timing constraints being duly provided to emulate the actual operating conditions. The metrics are reported after the complete placement and routing of the design. Table III compares the performance of the proposed square architecture against different exact and approximate square architectures reported in the literature. The analysis is done for an operand word length of eight bits. The analysis reveals that the CORDIC-based square architecture shows an improved performance when compared to the existing exact and approximate designs. For ASIC implementation, a few designs report slightly better metrics than our proposed architecture. However, our design reports the best metrics for FPGA platforms, with only MUL1-based square architecture reported in [15] showing reduced LUT count and dynamic power dissipation. However, the square architecture based on the MUL1 design is a serial architecture and has an extremely large critical path delay. To get a better perspective of the performance improvement, Table IV shows the power-delay-area product (PDAP) for different designs on ASIC and FPGA platforms. Our design reports a 25% and 38% improvement over the next best design for ASIC and FPGA platforms, respectively. Due to the rigorous pipelining, our design implementation requires more registers than the other reported. However, each logic slice in Kintex-7 FPGAs supports eight flip-flop registers; thus, using these registers does not incur any extra hardware cost. Including these registers in the implementation process considerably reduces the critical paths. This has two advantages; first, the structure can be clocked at higher frequencies, enhancing speed and throughput. Second, the capacitances to be charged and discharged in a single clock cycle are reduced, reducing dynamic power dissipation. Further analysis focuses on the achievable accuracy versus performance trade-offs using the proposed square architecture. This is shown in Figs. 6 and 7, where power-delay-product (PDP) and resource usage are plotted against the NMED for different exact and approximate square architectures for FPGA and ASIC platforms, respectively. For a given NMED, our proposed square architecture has the least PDP and utilizes the least resources. Similar results are reflected in Figs. 8 and 9, plotting PDAP against the NMED for

TABLE III
Performance Analysis of Different 8-bit Square Architectures on ASIC and FPGA Platforms

Design	ASIC (180 nm CMOS)			FPGA (xc7k325tffg900c)		
	Area (μm^2)	Delay (ns)	Power (mW)	LUT	Delay (ns)	Power (mW)
Accurate						
[05]	1910.32	2.34	0.062	56	4.879	28.2
[20]	2115.44	2.65	0.067	62	8.6	29
[23]	1978.96	2.44	0.062	58	5.6	29
[24]	4503.84	2.98	0.075	132	9.8	28.2
[25]	3616.72	2.71	0.071	106	9.4	28.2
[26]	3343.76	2.66	0.071	98	8.9	20
[27]	2653.21	2.93	0.069	68	9.11	21
Approximate						
[11]	3421.12	1.612	0.066	59	4.1	21.33
MCom 4:2 [12]	2114.32	1.495	0.052	48	3.4	23.43
MFA1 [12]	3461.65	1.567	0.063	64	4.02	21.88
MFA2 [12]	4094.22	1.588	0.071	66	4.44	22.11
[13]	2312.16	1.441	0.0591	52	3.33	20.67
Normal [14]	2117.65	1.431	0.0583	50	3.76	21.56
Hybrid [14]	2216.71	1.442	0.0586	52	3.81	21.6
MUL1 [15]	1412.32	3.112	0.0411	32	7.8	18.75
MUL2 [15]	1721.11	3.211	0.0431	39	8.01	18.89
RRAM-I [16]	2410.61	1.621	0.056	53	3.01	20.87
RRAM-II [16]	2441.11	1.562	0.0552	54	3.334	20.81
Proposed	1522.01	1.44	0.0561	36	2.929	19.43

different exact and approximate square architectures for FPGA and ASIC platforms, respectively.

The proposed square architecture is also tested by using it to increase the intensity of the output pixels in an image edge-detection application, thereby leading to better contrast enhancement. Edge detection is performed by applying Gaussian noise filtering followed by horizontal and vertical filters to detect the edges. The proposed square architecture is then used to improve the contrast of the final filtered image by multiplying it with itself. The quality of the output image is reported in terms of the peak signal-to-noise ratio (PSNR) calculated with respect to the image obtained using the exact square architecture. Fig. 10 shows the output images obtained using different square architectures. Our proposed square architectures report an improvement of 20% in PSNR over the next best architecture.

TABLE IV
PDAP Analysis of Different 8-bit Square Architectures on ASIC and FPGA Platforms

Design	ASIC (180 nm CMOS)	FPGA (xc7k325tffg900c)
	PDAP (fJm^2)	PDAP (nJA)
[05]	0.277149	7.704917
[20]	0.375596	15.4628
[23]	0.299377	9.4192
[24]	1.006608	36.47952
[25]	0.695893	28.09848
[26]	0.631503	17.444
[27]	0.5363	13.009
[11]	0.36398	5.159727
MCom 4:2 [12]	0.164367	3.823776
MFA1 [12]	0.341738	5.629286
MFA2 [12]	0.461615	6.479114
[13]	0.196911	3.579217
Normal [14]	0.17667	4.05328
Hybrid [14]	0.187315	4.279392
MUL1 [15]	0.18064	4.68
MUL2 [15]	0.238191	5.901047
RRAM-I [16]	0.218826	3.329391
RRAM-II [16]	0.210478	3.746549
Proposed	0.122954	2.048777

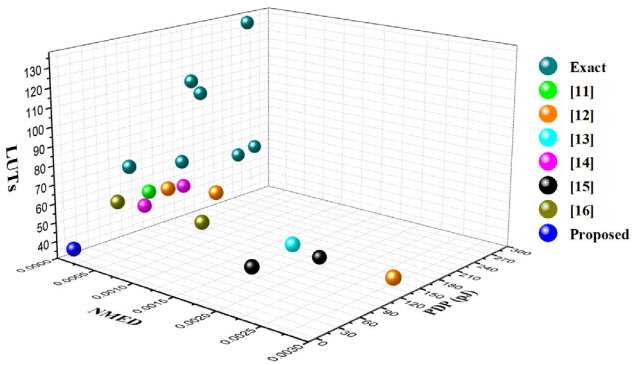

Fig. 6. PDP-LUT-NMED plot for different Square architectures implemented on the FPGA platform.

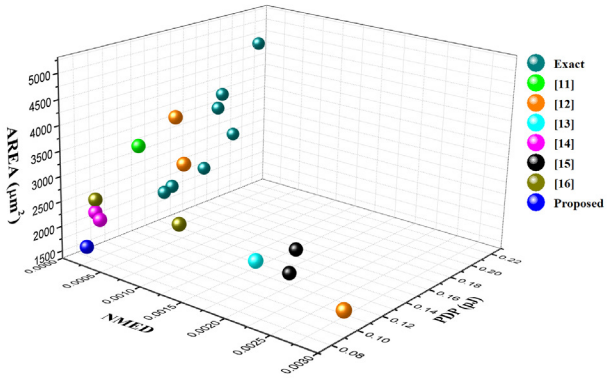

Fig. 7. PDP-AREA-NMED plot for different Square architectures implemented on the ASIC platform.

V. CONCLUSION

In this paper, we propose a square architecture based on the operation of the CORDIC algorithm in a linear coordinate system. While CORDIC has frequently been used in literature to evaluate different trigonometric and transcendental functions, it is very rarely used to evaluate linear functions. Being hardware-efficient, the computations are simple, enabling the inherent algorithm to be translated into diverse architectures to suit the application demands. In this paper, we focussed on unfolded (pipelined) architectures. Our analysis with ASIC and 7th-generation FPGAs reported a substantial performance improvement evaluated in terms of PDAP. Further, the accuracy-performance trade-offs achievable with our proposed square architecture outperform all the existing approximate multiplier-based square architectures. Heuristically, the convergence of the architecture shares a linear relationship with the operand word length. For large operand word-lengths, therefore, there will be an exponential increase in the resource utilized. Our future endeavors will focus on speeding up the convergence of the computations through the use of radix-4 arithmetic. This will reduce the number of iterative stages, resulting in lesser LUT utilization.

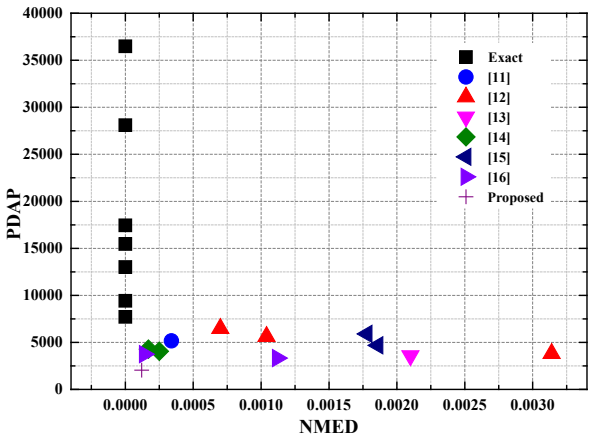

Fig. 8. PDAP-NMED plot for different Square architectures implemented on the FPGA platform.

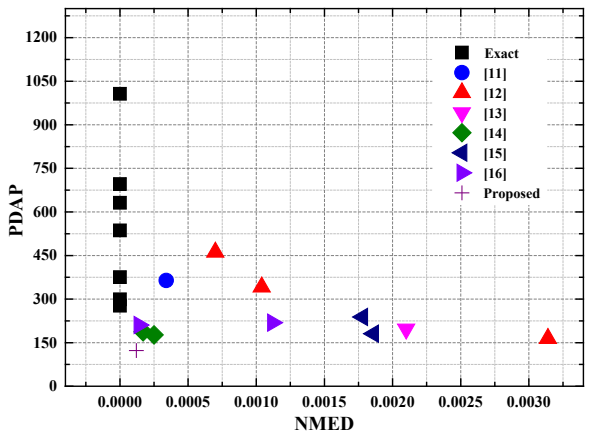

Fig. 9. PDAP-NMED plot for different Square architectures implemented on the ASIC platform.

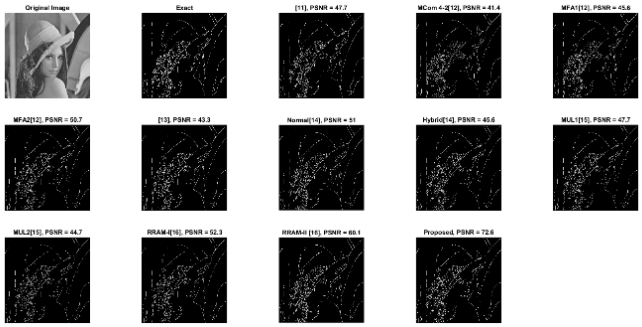

Fig. 10. Contrast enhancement using different square architectures for an edge-detection application.

References

[1] P. Jebashini, R. Uma, P. Dhavachelvan and H. K. Wye, "A Survey and Comparative Analysis of Multiply-Accumulate (MAC) block for Digital Signal Processing Application on ASIC and FPGA," *Journal of Applied Sciences,* July 2015; 15(7): 934-946, doi:10.3923/jas.2015.934.946.

[2] A. Deepa and C. N. Marimuthu, "A High Speed VLSI Architecture of a Pipelined Reed Solomon Encoder for Data Storage in Communication Systems," *Asian Journal of Research in Social Sciences and Humanities,* Feb. 2017; 7(2): 228–238, doi:10.5958/2249-7315.2017.00085.5.

[3] J. Pihl and E. J. Aas, "A multiplier and squarer generator for high-performance DSP applications," in *Proceedings of the 39th Midwest Symposium on Circuits and Systems*, Ames, IA, USA, 1996, pp. 109-112, doi:10.1109/MWSCAS.1996.594049.

[4] M. Kaveh, M. Khishe and M. R. Mosavi, "Design and implementation of a neighborhood search biogeography-based optimization trainer for classifying sonar dataset using multi-layer perceptron neural network," *Analog Integrated Circuits and Signal Processing* Nov. 2018; 100: 405-428, doi: 10.1007/s10470-018-1366-3.

[5] J. Bajaj and B. Jajodia, "Efficient Hardware Implementation of High-Speed Recursive Vedic Squaring Architecture on FPGA," in *Proceedings of the International Conference on Electrical, Computer and Energy Technologies (ICECET)*, Cape Town, South Africa, Dec. 2021, pp. 1-6, doi:10.1109/ICECET52533.2021.9698774.

[6] A. Deepa and C. N. Marimuthu, "Design of a high speed Vedic multiplier and square architecture based on *Yavadunam Sutra*." *Sadhana*, Indian Academy of Sciences Aug. 2019; 44:197: 1–10, doi:0.1007/s12046-019-1180-3.

[7] P. J. Edavoor, S. Raveendran and A. D. Rahulkar, "Approximate Multiplier Design using Novel Dual-Stage 4:2 Compressors," *IEEE Access*, Mar. 2020; 8: 48337-51, doi: 10.1109/ACCESS.2020.2978773.

[8] S. Asif and Y. Kong, "Design of an Algorithmic Wallace Multiplier using High-Speed Counters," in *Proc. 10th Int. Conf. Comput. Eng. Syst. (ICCES)*, Cairo, Egypt; Dec. 2015, pp. 133-138, doi: 10.1109/ICCES.2015.7393033.

[9] W. J. Townsend, E. E. Swartzlander and J. A. Abraham, "A Comparison of Dadda and Wallace Multiplier Delays," in *Proc. SPIE Annual meeting Opt. Sci. Technol.,* San Diego, CA, USA, Dec. 2003, pp. 552-560, doi: 10.1117/12.507012.

[10] H. Xiao, H. Xu, X. Chen, Y. Wang, Y. Han, "Fast and high-accuracy approximate MAC unit design for CNN computing," *IEEE Embedded Systems Letters* 14 (3) September 2022 155–158, doi: 10.1109/LES.2021.3137335.

[11] T. Yang, T. Ukezono, T. Sato, "Low-power and high-speed approximate multiplier design with a tree compressor," in: *Proceedings of the 35th International Conference on Computer Design (ICCD),* Boston, MA, USA, November 2017, pp. 89–96, doi:10.1109/ICCD.2017.22.

[12] B. Rashidi, "Efficient and low-cost approximate multipliers for image processing," *Integration, the VLSI Journal* Jan. 24; 94(102084), pp. 1-13, doi: 10.1016/j.vlsi.2023.102084.

[13] S. K. Beura, B. P. Devi, P. K. Saha and P. K. Meher, "Design of a Novel Inexact 4:2 Compressor and Its Placement in the Partial Product Array for Area, Delay, and Power-Efficient Approximate Multipliers," *Circuits, Systems, and Signal Processing* March 2024 43: 3748-3774, doi: 10.1007/s00034-024-02630-4.

[14] M. Zhang, S. Nishizawa and S. Kimura, "Area Efficient Approximate 4-2 Compressor and Probability-Based Error Adjustment for Approximate Multiplier," *IEEE Transactions on Circuits and Systems-II: Express Briefs*, Vol. 70, No. 5, May 2023, doi: 10.1109/TCSII.2023.3257852.

[15] L. Sayadi, S. Timarachi, A.S. Akbari, Two efficient approximate unsigned multipliers by developing new configuration for approximate 4:2 compressors, IEEE Transactions on Circuits and Systems-I Feb 70 (4) (2023) 1649–1659, doi: 10.1109/TCSI.2023.3242558.

[16] V. Tammineni, S. K. Beura, M. V. H. B. Murthy, S. Majumdar and P. Saha, "Optimized recursive approximate multipliers for edge detection and image smoothing applications," *Microsystem Technologies*, Nov. 2024, doi: 10.1007/s00542-024-05810-z.

[17] T. Matsunaga, S. Kimura and Y. Matsunaga, "Multi-Operand Adder Synthesis on FPGAs using Generallized Parallel Counters," in *Proc. of 15th Asia and South Pacific Design Automation Conference (ASP-DAC),* Taipei, Taiwan, Jan 2010, pp. 337-342, doi: 10.1109/ASPDAC.2010.5419871.

[18] H. P. Afshar, P. Brisk and P. Ienne, "Efficient Synthesis of Compressor Trees on FPGAs," in *Proc. of 2008 Asia and South Pacific Design Automation Conference,* Seoul, South Korea, Mar. 2008, pp. 138-143, doi: 10.1109/ASPDAC.2008.4483927.

[19] H. P. Afshar, A. Neogy, P. Brisk and P. Ienne, "Compressor Tree Synthesis on Commercial High-Performance FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, Dec. 2011; 4(4): 1-19, doi: 10.1145/2068716.2068725.

[20] B. N. K. Reddy, "Design and implementation of high performance and area efficient square architecture using Vedic Mathematics," *Analog Integrated Circuits and Signal Processing,* July 2019; 102: 501-506, doi: 10.1007/s10470-019-01496-w.

[21] P. Saritha, J. Vinitha, S. Sravya, V. Vijay and E. Mahesh, "4-Bit Vedic Multiplier with 18 nm FinFET Technology," in *Proceedings of the International Conference on Electronics and Sustainable Communication System (ICESC)*, Coimbatore, India, 2020, pp. 1079-84, doi: 10.1109/ICESC48915.2020.9155707.

[22] V. Bianchi V and I. D. Munari, "A modular Vedic multiplier architecture for model-based design and deployment on FPGA platforms," *Microprocessors and Microsystems,* July 2020; 76: 103106: 1-9, doi: 10.1016/j.micpro.2020.103106.

[23] S. Shetkar and S. Kohli, "Squaring Circuit Using 14nmFinFET Technology with Vedic Mathematics Approach." *IETE Journal of Research,* May 2024, 1-9, doi: 10.1080/03772063.2024.2355662.

[24] P. S. Kasliwal, B. P. Patil and D. K. Gautam, "Performance evaluation of squaring operation by Vedic Mathematics," *IETE Journal of Research*, 2011; 57(1), 39-41, doi: 10.4103/0377-2063.78327.

[25] H. Thapliyal, S. Kotiyal and M. B. Srinivas, "Design and analysis of a novel parallel square and cube architecture based on ancient Indian Vedic Mathematics" in *Proceedings of 48th Midwest Symposium on Circuits and Systems,* Covington, KY, USA, Aug. 2005, pp. 1462–1465, doi: 10.1109/MWSCAS.2005.1594388.

[26] D. K. Yadav and R. R. Lal, "Analysis of Vedic Mathematics Ekadhikena Purvena Sutra in Squaring and Multiplication," in *8th International Conference on Communication and Electronics Systems (ICCES 2023)*, Coimbatore, India, June 2023, pp. 12–19, doi: 10.1109/ICCES57224.2023.10192851.

[27] J. Sravana, K. S. Indrani, S. Mahurkar, M. Pranathi, D. R. Reddy and V. Vallabhuni, "Optimized VLSI Design of Squaring Multiplier Using Yavadunam Sutra Through Deficiency Bits Reduction," *Advances in Signal Processing and Communication Engineering. Lecture Notes in Electrical Engineering*, vol 929: 387-399. Springer, Singapore, Dec. 2022, doi: 10.1007/978-981-19-5550-1_36.

[28] B. Khurshid and J. J. Khan, "An Efficient Fixed-Point Multiplier based on CORDIC Algorithm," *Journal of Circuits, Systems and Computers,* 2021; 30(05): 1-19, doi: 10.1142/S0218126621500808.

[29] J. S. Walther, "A Unified Algorithm for Elementary Functions," in *Proceedings of the AFIPS Spring Joint Computer Conference*, New York, USA, May 1971, pp. 379-385, doi: 10.1145/1478786.1478840.

[30] J. S. Walther, "The story of Unified CORDIC," *Journal of VLSI Signal Processing systems for signal, image and video technology,* June 2000; 25(2): 107-112, doi: 10.1023/A:1008162721424.

[31] P. K. Meher, J. Valls, T. B. Juang and K. Sridhavan, "50 years of CORDIC: Algorithms, Architectures and Applications," *IEEE Transactions on Circuits and Systems-I*, 2009; 56(9): 1893-1907, doi: 10.1109/TCSI.2009.2025803.

[32] B. Lakshmi and A. Dhar, "CORDIC Architectures: A Survey," *VLSI Design*, 2010; 794891: 1-19, doi: 10.1155/2010/794891.

[33] S. M. Mohamed, W. S Sayed, A. G. Radwan and L. A. Said, "FPGA Implementation of Reconfigurable CORDIC Algorithm and a Memristive Chaotic System with Transcendental Non-Linearities," *IEEE Transactions on Circuits and Systems-I,* Regular paper, July 2022; 69(7): 2885-92, doi: 10.1109/TCSI.2022.3165469.

[34] G. Raut, S. Rai, S. K. Vishvakarma and A. Kumar, "RECON: Resource-Efficient CORDIC-based Neuron Architecture," *IEEE Open Journal of Circuits and Systems,* Jan 2021; 2: 170-181, doi: 10.1109/OJCAS.2020.3042743.