Lowcomplexity Generic VLSI Architecture Design Methodology for N^{th} Root and N^{th} Power Computations

Suresh Mopuri and Amit Acharyya, *Member, IEEE*

***Abstract*— In this paper, we propose a low complexity architecture design methodology for fixed point root and power computations. The state of the art approaches perform the root and power computations based on the natural logarithm-exponential relation using Hyperbolic COordinate Rotation DIgital Computer (CORDIC). In this paper, any root and power computations have been performed using binary logarithm-binary inverse logarithm relation. The designs are modeled using VHDL for fixed point numbers and synthesized under the $TSMC$40-nm CMOS technology @ 1 GHz frequency. The synthesis results shows that the proposed N^{th} root computation saves 19.38% on chip area and 15.86% power consumption when compared with the state of the art architecture for root computation without compromising the computational accuracy. Similarly, the proposed N^{th} power computation saves 38% on chip area, 35.67% power consumption when compared with the state of the art power computation with out loss in accuracy. The proposed root and power computation designs save 8 clock cycle latency when compared with the state of the art implementations.**

***Index Terms*— CORDIC, logarithm, exponential, VLSI architecture, root computation, power computation, hyperbolic CORDIC.**

I. Introduction

ROOT and power computations have been used in different areas such as atmospheric models, digital image synthesis, 3-D graphics and many VLSI signal processing applications [1]–[3]. However, the design and implementation of low complexity as well as highly accurate VLSI architecture of such N^{th} root and N^{th} power computation is a challenging task for real time resource constrained platform.

There are various approaches available for root computation. The well known method is Newton-Raphson (NR) method requiring an initial guess which may result different precision in the outputs [4]–[6]. The hardware complexity of NR method increases with increasing value of N. A General digit-recurrence algorithm is presented in [7] whose hardware complexity like NR approach, also depends on N. In [8], a top-level approach has been presented based on the binary logarithm-binary inverse logarithm relation i.e, $R^{\frac{1}{N}} = 2^{\frac{log_2(R)}{N}}$. But this approach [8] did not present the implementation details of the binary logarithm, division and binary inverse logarithm. Another approach was presented in [9] based on the natural logarithm-exponential relation i.e, $R^{\frac{1}{N}} = exp(\frac{ln(R)}{N})$ where the natural logarithm, division and exponential computations are performed using CORDIC. On the other hand, the powers are computed using multipliers [10]–[13], in which the square and cube operations were computed using reduced partial product arrays and ancient Indian Vedic mathematics. However, these approaches [10]–[13] are not generic for the N^{th} power computation. Such a generic approach for N^{th} power computation is proposed in [14] based on the natural logarithm-exponential relation i.e, $R^N = exp(ln(R) \times N)$ where the natural logarithm and exponential computations are performed using CORDIC.

It is well known that the CORDIC performs several tasks such as trigonometric, hyperbolic and logarithmic functions, real and complex multiplications, division and square-root using shift add operations [15]–[21]. However, the convergence and precision of the CORDIC depend on its negative index (m) and positive (n) boundaries respectively [15], [21] (elaborated in section II, please see equation (11), (12) and (13a)). The CORDIC convergence boundary (m) poses the following limitations on the state of the art N^{th} root and N^{th} power computations [9], [14].

- The CORDIC negative index boundary (m) limits the input range of R and N. As m value increases the input ranges of R and N will increase.
- As m value increases, number of CORDIC iterative stages will increase in turn the hardware complexity, area, power consumption and latency will increase.

Addressing the fore mentioned limitations, in this paper,

- We propose a low complexity architecture design methodology for the N^{th} root and N^{th} power computation based on the binary logarithm-exponential relation using CORDIC.
- We propose Binary Hyperbolic CORDIC algorithm to perform the binary logarithm and inverse binary logarithm computations.

Manuscript received March 2, 2019; revised July 13, 2019; accepted August 29, 2019. This work was supported in part by the Science and Engineering Research Board (SERB), Government of India, for the project entitled "Intelligent IoT enabled Autonomous Structural Health Monitoring System for Ships, Aeroplanes, Trains and Automobiles" through the Impacting Research Innovation and Technology (IMPRINT) Program under Grant IMP/2018/000375. This article was recommended by Associate Editor A. Cilardo. *(Corresponding author: Amit Acharyya.)*

The authors are with the Department of Electrical Engineering, Indian Institute of Technology Hyderabad (IIT Hyderabad), Hyderabad 502285, India (e-mail: ee13p0004@iith.ac.in; amit_ acharyya@iith.ac.in).

Digital Object Identifier 10.1109/TCSI.2019.2939720

The highlights of this paper are given below.

- The proposed approach unlike the state of the art approach will not depend on the CORDIC negative index boundary (m) for logarithm and exponential computations and thereby reduces the hardware complexity, power consumption and latency.
- The proposed architecture and the state of the art architecture are coded in VHDL for fixed point numbers and the ASIC implementation has been done at TSMC 45nm CMOS technology @ $VDD = 1.08V$ and clock frequency @ $1GHz$ with the help of Synopsis Design Compiler (DC) and IC compiler. The synthesis results show that the proposed N^{th} root design saves 19.38% on chip area, 15.86% power consumption when compared with the state of the art architecture [9] without compromising the computational accuracy. Similarly, the proposed N^{th} power computation design saves 38% on chip area, 35.67% power consumption when compared with the state of the art power computation [14].
- The proposed approach is one order superior in accuracy for the N^{th} root computation and two order superior for N^{th} power computation.
- The proposed approach will fix the shortcomings of [8] including the implementation of binary logarithm and inverse binary logarithm computations.

The rest of this paper is organized as follows: Section II provides the necessary theoretical background. Section III introduces the proposed methodology. Section IV details the experimental results and section V concludes the discussion.

II. Theoretical Background

The state of the art approaches perform the N^{th} root and N^{th} power computations based on the the natural logarithm-exponential relation [9], [14]

$$R^{\frac{1}{N}} = exp(\frac{ln(R)}{N}) \tag{1a}$$

$$R^{N} = exp(ln(R) \times N) \tag{1b}$$

The computations shown in (1) have been divided into three steps. The first step is the computation of the natural logarithm i.e, $ln(.)$ in both approaches [9], [14]. The next step is division operation for root computation and multiplication operation for power computation. The final step is exponential computation i.e, $exp(.)$. The natural logarithm and exponential computations are performed in [9], [14] using Hyperbolic CORDIC. The division operation is performed using linear CORDIC [9]. The basic working principle of Hyperbolic CORDIC can be expressed as:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = R_H * \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}; \ R_H = \begin{bmatrix} cosh(z) & sinh(z) \\ sinh(z) & cosh(z) \end{bmatrix} \tag{2}$$

where $[x_0, y_0]$ and $[x_f, y_f]$ are the initial and final position vectors, R_H is hyperbolic rotation matrix and z is the angle of rotation [15]. By factoring out the $cosh(z)$ term, the above equation can be rewritten as follows

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = cosh(z) * \begin{bmatrix} 1 & tanh(z) \\ tanh(z) & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{3}$$

TABLE I
Convergence Range of CORDIC

HR CORDIC	HV CORDIC	LV CORDIC	Scale factor(K_H)
$\lvert z_0\rvert \leq 1.1182$	$\lvert tanh^{-1}(\frac{y_0}{x_0})\rvert \leq 1.1182$	$\frac{y_0}{x_0} \leq 2$	1.2075

The CORDIC performs the rotation iteratively through an angle α_i instead of performing the rotation directly through the angle z, where $\alpha_i = tanh^{-1}(2^{-i})$. So that z can be decomposed as follows

$$z = \sum_{i=1}^{n} \sigma_i \alpha_i; \ \sigma_i = \pm 1 \tag{4}$$

where σ_i is decomposition factor. The iteration formula for conventional Hyperbolic CORDIC can be expressed as follows

$$x_{i+1} = x_i + \sigma_i(2^{-i} y_i) \tag{5a}$$

$$y_{i+1} = y_i + \sigma_i(2^{-i} x_i) \tag{5b}$$

$$z_{i+1} = z_i - \sigma_i(tanh^{-1}(2^{-i})) \tag{5c}$$

where i is an integer starts with 1. The σ_i can be determined by mode of operation [15]. Based on the mode of operation, the CORDIC has been divided into two classes. One is rotation mode CORDIC and other is vectoring mode CORDIC. The σ_i for Hyperbolic Rotation (HR) mode CORDIC and Hyperbolic Vectoring (HV) mode CORDIC is given by

$$\sigma_i = sign(z_i) \tag{6a}$$

$$\sigma_i = -sign(y_i) \tag{6b}$$

The scale factor for the Hyperbolic CORDIC K_H is expressed as:

$$K_H = \prod_{1}^{n} cosh(\sigma_i \alpha_i) = \prod_{1}^{n} cosh(\alpha_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{1 - 2^{-2i}}} \tag{7}$$

From the CORDIC convergence theorem in [21], to guarantee the convergence, the iterations $i = 4, 13, 40, \cdots, k, (3k+1)$ must be repeated. The maximum z value of conventional Hyperbolic CORDIC is expressed as

$$|z|_{max} = \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} tanh^{-1}(2^{-i}) \tag{8}$$

The iterative formula for Linear Vectoring (LV) mode CORDIC [15] is expressed as follows

$$x_{i+1} = x_i \tag{9a}$$

$$y_{i+1} = y_i + \sigma_i(2^{-i} x_i) \tag{9b}$$

$$z_{i+1} = z_i - \sigma_i(2^{-i}) \tag{9c}$$

where i is an integer starts with 0, and $\sigma_i = -sign(y_i)$. Table I summarizes the convergence range of the HR, HV and LV CORDIC as $n \to \infty$. The convergence limits of CORDIC shown in the Table I are not enough for the implementation logarithm, exponential and division computations in practical applications [9], [15], [21]. Hence, the convergence range for

TABLE II
IMPROVEMENT IN CONVERGENCE RANGE OF CORDIC

m	HR CORDIC $\lvert z_0 \rvert \le$	HV CORDIC $\lvert tanh^{-1}(\frac{y_0}{x_0}) \rvert \le$	LV CORDIC $(\frac{y_0}{x_0} \le 2^{m+1})$	Scale factor(K_H)
0	2.0999335	2.0999335	2	1.8256
1	3.81692712	3.81692712	4	5.2461
2	6.935112	6.935112	8	59.4108
3	12.82685914	12.82685914	16	1.0755×10^4
4	24.2555	24.2555	32	4.9838×10^8

the Hyperbolic CORDIC was improved in [21] by considering the negative index numbers as $i = -m, -m+1, \cdots, 1, \cdots, n$. However, the iteration formula can not be same as (5). As mentioned in [21], when $i \le 0$, the term 2^{-i} in (5) will be replaced with the term $(1-2^{-2^{-i+1}})$. The iterative formula of Hyperbolic CORDIC (5) for $i \le 0$ can be rewritten as follows

$$x_{i+1} = x_i + \sigma_i(1-2^{-2^{-i+1}})y_i \tag{10a}$$
$$y_{i+1} = y_i + \sigma_i(1-2^{-2^{-i+1}})x_i \tag{10b}$$
$$z_{i+1} = z_i - \sigma_i tanh^{-1}(1-2^{-2^{-i+1}}) \tag{10c}$$

where σ_i is same as shown in (6). The iterative formula of LV CORDIC for $i \le 0$ is same as shown in (9) [21]. If n is large, the convergence of LV CORDIC depends on m which is expressed as follows

$$z_n \leftarrow z_0 + \frac{y_0}{x_0} = VecL_z(m, n, x_0, y_0, z_0) \le 2^{m+1} \tag{11}$$

where $VecL_z(.)$ represents the z component output of LV CORDIC. If n is large, the x, y, z of the Hyperbolic CORDIC tends to the results shown in (12) and (13a) for the rotation and vectoring modes respectively.

$$\begin{aligned} x_n &\leftarrow \frac{1}{K_H}(cosh(z_0)x_0 + sinh(z_0)y_0) \\ &= RotH_x(m, n, x_0, y_0, z_0) \end{aligned} \tag{12a}$$
$$\begin{aligned} y_n &\leftarrow \frac{1}{K_H}(sinh(z_0)x_0 + cosh(z_0)y_0) \\ &= RotH_y(m, n, x_0, y_0, z_0) \end{aligned} \tag{12b}$$
$$z_n \leftarrow 0 = RotH_z(m, n, x_0, y_0, z_0) \tag{12c}$$
$$x_n \leftarrow \frac{1}{K_H}\sqrt{x_0^2 - y_0^2} = VecH_x(m, n, x_0, y_0, z_0) \tag{13a}$$
$$y_n \leftarrow 0 = VecH_y(m, n, x_0, y_0, z_0) \tag{13a}$$
$$z_n \leftarrow z_0 + tanh^{-1}(\frac{y_0}{x_0}) = VecH_z(m, n, x_0, y_0, z_0) \tag{13a}$$

where $RotH_c(.)$ and $VecH_c(.)$ represent the rotation mode and vectoring mode of the Hyperbolic CORDIC respectively and c represents the output of x or y or z component. For different values of m, the improvement in the convergence range of CORDIC is summarized in the Table II. It can be noted from Table II that the convergence range of CORDIC increases as m increases. Now the steps involved in (1) can be computed using HV, HR and LV CORDIC as follows [9], [14]:

Step1: The logarithm can be computed using HV CORDIC. Consider inputs to the HV CORDIC as $x_0 = R+1$, $y_0 = R-1$

TABLE III
CONVERGENCE RANGE OF R AND N

m	$R \le$	$N >$
0	66.6775	2.0999335
1	$2.067 * 10^3$	1.9085
2	$1.0562 * 10^6$	1.7338
3	$1.3844 * 10^{11}$	1.6034

and $z_0 = 0$. The $ln(R)$ can be computed as:

$$\begin{aligned} ln(R) &= 2 \times VecH_z(m, n, R+1, R-1, 0) \\ &= 2 \times tanh^{-1}(\frac{R-1}{R+1}) \end{aligned} \tag{14}$$

The $ln(R)$ value can be obtained by shifting the HV CORDIC output shown in (14) by 1 bit to the left.

Step2: The division operation can be performed using LV CORDIC by considering inputs to the LV CORDIC as $x_0 = N$, $y_0 = ln(R)$ and $z_0 = 0$

$$\frac{ln(R)}{N} = VecL_z(m, n, N, ln(R), 0) \tag{15}$$

Step3: The exponential computation can be performed using the HR CORDIC by considering inputs as $x_0 = K_H$, $y_0 = 0$ and $z_0 = \frac{ln(R)}{N}$. The outputs are $x_n = RotH_x(m, n, K_H, 0, \frac{ln(R)}{N}) = cosh(\frac{ln(R)}{N})$, $y_n = RotH_y(m, n, K_H, 0, \frac{ln(R)}{N}) = sinh(\frac{ln(R)}{N})$. The exponential is computed by adding HR CORDIC outputs as follows

$$\begin{aligned} exp(\frac{ln(R)}{N}) &= RotH_x(m, n, K_H, 0, \frac{ln(R)}{N}) \\ &\quad + RotH_y(m, n, K_H, 0, \frac{ln(R)}{N}) \end{aligned} \tag{16}$$

In R^N computation, the input to the HR CORDIC is $z_0 = ln(R) \times N$. The above equation can be rewritten as

$$\begin{aligned} exp(ln(R) \times N) &= RotH_x(m, n, K_H, 0, ln(R) \times N) \\ &\quad + RotH_y(m, n, K_H, 0, ln(R) \times N) \end{aligned} \tag{17}$$

From the above three step, the $R^{\frac{1}{N}}$ computation will depend on the m. The convergence of R and N is summarized in the Table III for different values of m.

From Table III, (14), (15), (16) and (17), it can be inferred that the steps involved in $R^{\frac{1}{N}}$ and R^N computations will have the following limitations.

- The convergence of R and N values depends on the negative index boundary (m) of CORDIC.
- The input range of R and N increases as m increases.
- As m increases, the iterative stages in CORDIC also increases resulting in increase in the hardware complexity, latency and power consumption.

Addressing all the limitations, in the next section we introduce a low complexity architecture design methodology for N^{th} root and N^{th} power computations.

III. PROPOSED METHODOLOGY AND ARCHITECTURE

In this section, a low complexity architecture design methodology for N^{th} root and N^{th} power computation is proposed.

A. Proposed Methodology

The $R^{\frac{1}{N}}$ and R^N computations can be performed using binary logarithm- binary inverse logarithm relation.

$$R^{\frac{1}{N}} = 2^{(\frac{log_2(R)}{N})} \tag{18a}$$
$$R^N = 2^{(log_2(R)\times N)} \tag{18b}$$

The binary logarithm $log_2(.)$ and binary exponential $2^{(.)}$ can not be performed by Hyperbolic CORDIC as it is due to rotation matrix R_H. Hence, we will investigate the properties of R_H and introduce new R_H to compute the binary logarithm $log_2(.)$ and binary exponential $2^{(.)}$. From (2), it can be observed that the determinant of R_H is $cosh^2(z) - sinh^2(z) = 1$. In R_H, the $cosh(z)$ is an even function and the $sinh(z)$ is an odd function. The natural exponential can be computed using $cosh(z)$ and $sinh(z)$ as follows

$$e^z = cosh(z) + sinh(z) \tag{19a}$$
$$e^{-z} = cosh(z) - sinh(z) \tag{19b}$$

Now, we will define two new functions $cosh_b(z)$ and $sinh_b(z)$ which are even and odd functions for inverse binary logarithm computation. The binary inverse logarithm computation can be expressed as

$$2^z = cosh_b(z) + sinh_b(z) \tag{20a}$$
$$2^{-z} = cosh_b(z) - sinh_b(z) \tag{20b}$$

By solving (20), the $cosh_b(z)$ and $sinh_b(z)$ can be obtained as follows

$$cosh_b(z) = \frac{2^z + 2^{-z}}{2} \tag{21a}$$
$$sinh_b(z) = \frac{2^z - 2^{-z}}{2} \tag{21b}$$

From (21), it can be noted that the defined $cosh_b(z)$ is an even function, $sinh_b(z)$ is an odd function and $cosh_b^2(z) - sinh_b^2(z) = 1$. Hence, the $cosh(z)$ and $sinh(z)$ can be replaced with $cosh_b(z)$ and $sinh_b(z)$ in R_H for binary logarithm and inverse binary logarithm computation and (2) can be rewritten as

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{vmatrix} cosh_b(z) & sinh_b(z) \\ sinh_b(z) & cosh_b(z) \end{vmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{22}$$

The Hyperbolic CORDIC in (22) is intended for computation of binary logarithm and inverse binary logarithm. Therefore, it is named as Binary Hyperbolic CORDIC. Taking $cosh_b(z)$ term out from (22), the $\frac{sinh_b(z)}{cosh_b(z)}$ is considered as $tanh_b(z)$ then (22) can be rewritten as

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = cosh_b(z) \begin{bmatrix} 1 & tanh_b(z) \\ tanh_b(z) & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{23}$$

From (21), $tanh_b(z) = \frac{2^z - 2^{-z}}{2^z + 2^{-z}}$. As mentioned section II, the CORDIC performs the rotation iteratively with predefined angle α_i instead of rotation directly through z, where $\alpha_i = tanh_b^{-1}(2^{-i})$. Assume that $tanh_b(z) = t$ and t can be derived as follows

$$t = \frac{2^z - 2^{-z}}{2^z + 2^{-z}} \tag{24}$$

The above equation can be rewritten as

$$t = \frac{2^{2*z} - 1}{2^{2*z} + 1} \tag{25}$$

From the above equation $z = tanh_b^{-1}(t)$ can be derived as follows

$$tanh_b^{-1}(t) = \frac{1}{2} log_2 \frac{(1+t)}{(1-t)}; \ |t| < 1 \tag{26}$$

The rotation angle z can be decomposed in terms of predefined angle α_i as

$$z = \sum_{i=1}^{n} \sigma_i \alpha_i = \sum_{i=1}^{n} \sigma_i tanh_b^{-1}(2^{-i}); \ \sigma_i = \pm 1 \tag{27}$$

The decomposition factor (σ_i) can be determined by the mode of operation. The rotation formula for i^{th} iteration corresponding angle α_i using (23).

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = cosh_b(\sigma_i \alpha_i) \begin{bmatrix} 1 & tanh_b(\sigma_i \alpha_i) \\ tanh_b(\sigma_i \alpha_i) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{28}$$

From (21), the $sinh_b(.)$ is an odd function and $cosh_b(.)$ is an even function so that $tanh_b(.)$ is an odd function. Since $\sigma_i = \pm 1$, (28) can be rewritten as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = cosh_b(\alpha_i) \begin{bmatrix} 1 & \sigma_i tanh_b(\alpha_i) \\ \sigma_i tanh_b(\alpha_i) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{29}$$

where $tanh_b(\sigma_i \alpha_i) = \sigma_i tanh_b(\alpha_i)$. Using (27) and (29), the iterative formula of CORDIC is expressed as follows

$$x_{i+1} = x_i + \sigma_i(2^{-i} y_i) \tag{30a}$$
$$y_{i+1} = y_i + \sigma_i(2^{-i} x_i) \tag{30b}$$
$$z_{i+1} = z_i - \sigma_i(tanh_b^{-1}(2^{-i})) \tag{30c}$$

where i is an integer starts with 1, $\sigma_i = sign(z_i)$ and $\sigma_i = -sign(y_i)$ for rotation and vectoring mode respectively. The scale factor $cosh_b(\alpha_i)$ is independent of decomposition factor σ_i so that instead of scaling during each iteration, the magnitude of final output could be scaled by final scale factor K_b. The K_b is computed using the following equation

$$K_b = \prod_{i=1}^{n} cosh_b(\alpha_i) \tag{31}$$

By substituting $cosh_b(z) = \frac{1}{\sqrt{1 - tanh_b^2(z)}}$ and $tanb(\alpha_i) = 2^{-i}$ in (31), the above equation can be rewritten as

$$K_b = \prod_{i=1}^{n} \frac{1}{\sqrt{1 - tanh_b^2(\alpha_i)}} = \prod_{i=1}^{n} \frac{1}{\sqrt{1 - 2^{-2i}}} \tag{32}$$

From (7) and (32), it can be noted that the scale factors K_H and K_b are same. The maximum z can be computed as

$$|z|_{max} = \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} tanh_b^{-1}(2^{-i}) \tag{33}$$

As $n \to \infty$, the convergence of Binary Hyperbolic rotation mode (BR) CORDIC and Binary Hyperbolic vectoring mode (BV) CORDIC have been tabulated in Table IV.

TABLE IV
CONVERGENCE OF BINARY HYPERBOLIC CORDIC

BR CORDIC	BV CORDIC	K_b
$z_0 \le 1.6132$	$tanh_b^{-1}(\frac{y0}{x0}) \le 1.6132$	1.2075

TABLE V
CONVERGENCE OF IMPROVED BINARY HYPERBOLIC CORDIC

m	BR CORDIC $\|z_0\| \le$	BV CORDIC $\|tanh_b^{-1}(\frac{y_0}{x_0})\| \le$	Scale factor(K_b)
0	3.0169	3.0169	1.8256
1	5.4940	5.4940	5.2461
2	9.992	9.992	59.4108
3	18.4925	18.4925	1.0755×10^4

From the Table IV, it can be noted that the scale factor of the Binary Hyperbolic CORDIC is same as the conventional Hyperbolic CORDIC. The convergence range of the Binary Hyperbolic CORDIC is improved when compared with the conventional Hyperbolic CORDIC shown in the Table II. The convergence limit shown in the Table IV is not adequate for implementation logarithm and exponential computations. The convergence range of Binary Hyperbolic CORDIC can be improved by considering negative indices as $i = -m, -m+1, \cdots, 1, \cdots, n$ and replacing the term 2^{-i} with the term $(1-2^{-2^{-i+1}})$ as mentioned in section II. Now the convergence range and scale factor depend on the negative index boundary (m). For different m values the convergence range and scale factor of the proposed improved Binary Hyperbolic CORDIC has been summarized in Table V. It can be noted from Table V that the the convergence range of the Binary CORDIC increases as m increases. If n is considered large, the x, y, z of the Binary Hyperbolic CORDIC tend to the results shown in (34) and (35a) for the rotation and vectoring modes respectively.

$$x_n \leftarrow \frac{1}{K_b}(cosh_b(z_0)x_0 + sinh_b(z_0)y_0) = RotH_x^b(m, n, x_0, y_0, z_0) \tag{34a}$$

$$y_n \leftarrow \frac{1}{K_b}(sinh_b(z_0)x_0 + cosh_b(z_0)y_0) = RotH_y^b(m, n, x_0, y_0, z_0) \tag{34b}$$

$$z_n \leftarrow 0 = RotH_z^b(m, n, x_0, y_0, z_0) \tag{34c}$$

$$x_n \leftarrow \frac{1}{K_b}\sqrt{x_0^2 - y_0^2} = VecH_x^b(m, n, x_0, y_0, z_0) \tag{35a}$$

$$y_n \leftarrow 0 = VecH_y^b(m, n, x_0, y_0, z_0) \tag{35b}$$

$$z_n \leftarrow z_0 + tanh_b^{-1}(\frac{y_0}{x_0}) = VecH_z^b(m, n, x_0, y_0, z_0) \tag{35c}$$

where $RotH_c^b(.)$ and $VecH_c^b(.)$ represent the rotation mode and vectoring mode of Binary Hyperbolic CORDIC respectively. The Hyperbolic CORDIC can be extended for other logarithm and inverse logarithm computations by storing appropriate predefined angles.

Our main goal now is to perform the computations shown in (18), which can be performed using the Binary Hyperbolic CORDIC shown in (34) and (35a). The computations shown in (18) is now divided into the following three steps.

Step1: The first step is binary logarithm $log_2(R)$ computation. Consider inputs to the Binary Hyperbolic Vectoring mode (BV) CORDIC as $x_0 = R+1$, $y_0 = R-1$ and $z_0 = 0$. The output z_n is expressed using (35a) and (26) as follows

$$log_2(R) = 2 \times VecH_z^b(m, n, R+1, R-1, 0) = 2 \times tanh_b^{-1}(\frac{R-1}{R+1}) \tag{36}$$

From the Table IV, the convergence limit of the BV CORDIC is $tanh_b^{-1}(\frac{y_0}{x_0}) \le 1.6132$. From (36), $\frac{R-1}{R+1} \le tanh_b(1.6132) = 0.8069$ and $R \in [\frac{1}{9.36}, 9.36]$.

The range of $R \in [\frac{1}{9.36}, 9.36]$ is limited and may not adequate for many practical applications where $R \notin [\frac{1}{9.36}, 9.36]$. Therefore, the range of R can be increased by considering negative indices like the state of the art design [9] and as shown in Table V. But this technique increases the hardware complexity, latency and power consumption. Hence, we will introduce a **normalization procedure** which enhances the convergence limit without increasing the hardware complexity and latency.

Consider the working range of R as $r \in [1, 2]$. If $R \notin [1, 2]$, the R can be scaled down to the working range by performing a simple shifting operation. For example, if $2^{(k)} < R \le 2^{k+1}$ where k is an integer, the R is scaled down to working range by right shifting R by k bits. The R value can be expressed as:

$$R = 2^k * r; \quad r \in [1, 2] \tag{37}$$

Now the $log_2(R)$ can be computed as follows

$$log_2(R) = log_2(2^k * r) = k + log_2(r) \tag{38}$$

Now the $log_2(r)$ can be computed using the BV CORDIC by considering inputs $x_0 = (r+1)$, $y_0 = (r-1)$, $z_0 = 0$ and $m = \times$ (don't care condition).

$$log_2(r) = 2 \times VecH_z^b(\times, n, r+1, r-1, 0) \tag{39}$$

where $m = \times$ denotes that the computation in (39) will not depend on the negative index boundary (m). The computations shown in (37) and (38) can be performed with shift and add operations. From (39), it can be noted that the logarithm computation is independent of m which reduces the hardware complexity and latency. As an example, consider $R = 49$ then $2^5 < R \le 2^6$ and $k = 5$. After shifting R by $k = 5$ bits to right, r will become 1.53125. The $log_2(r)$ is computed using BV CORDIC as shown in (39) and $log_2(r) = 0.6147$. Thereafter $log_2(r)$ should be brought to its original value by adding $k = 5$ i.e, $log_2(R) = 5.6147$ as shown in (38).

Step2: The next step in root computation is the division operation using LV CORDIC. The inputs to the LV CORDIC are $x_0 = N$, $y_0 = log_2(R)$ and $z_0 = 0$. From (11), the output converge to the following result.

$$\frac{log_2(R)}{N} = VecL(m, n, N, log_2(R), 0) \tag{40}$$

The second step in the R^Ncomputation is multiplication i.e, $log_2(R) \times N$ which can be performed using any conventional multiplier.

Step3: The final step is binary inverse logarithm computation. The $2^{\frac{log_2(R)}{N}}$ can be computed using Binary Rotation mode CORDIC (BR CORDIC). Consider the inputs to BR CORDIC as $x_0 = K_b$, $y_0 = 0$ and $z_0 = \frac{log_2(R)}{N}$. So that outputs of BR CORDIC can be expressed using (34)

$$x_n = RotH_x^b(m, n, K_b, 0, \frac{log_2(R)}{N}) = cosh_b(\frac{log_2(R)}{N}) \tag{41a}$$

$$y_n = RotH_y^b(m, n, K_b, 0, \frac{log_2(R)}{N}) = sinh_b(\frac{log_2(R)}{N}) \tag{41b}$$

The binary inverse logarithm can be computed by adding BR CORDIC outputs which is expressed as follows

$$2^{(\frac{log_2(R)}{N})} = cosh_b(\frac{log_2(R)}{N}) + sinh_b(\frac{log_2(R)}{N}) = R^{\frac{1}{N}} \tag{42}$$

In R^N computation, z_0 will be $log_2(R) \times N$. From Table IV, it is observed that the BR CORDIC has convergence limit which limits the input range for exponential computation i.e, $z_0 = log_2(R) \times N \leq 1.6132$ or $z_0 = \frac{log_2(R)}{N} \leq 1.6132$. However, this limit $z_0 \leq 1.6132$ is not adequate for practical applications. Therefore, it can be enhanced by considering negative indices for BR CORDIC as shown in the Table V but this technique results the increase hardware complexity and latency. Hence, we introduce a **normalization procedure** to enhance the convergence limit.

Consider a real number $P \leq 1$. If $P > 1$, it can be split into two parts as shown in the following equation

$$P = P_I + P_F \tag{43}$$

where P_I and P_F represents the integer and fractional parts of P respectively. Now 2^P can be computed using following equation

$$2^P = 2^{P_I + P_F} = 2^{P_I} * 2^{P_F} \tag{44}$$

Now P_F is less than 1. The 2^{P_F} can be compute using basic BR CORDIC considering inputs as $x_0 = K_b$, $y_0 = 0$ and $z_0 = P_F \leq 1.6132$. The 2^{P_F} can be computed as

$$2^{P_F} = RotH_x^b(\times, n, K_b, 0, P_F) + RotH_y^b(\times, n, K_b, 0, P_F) \tag{45}$$

where $m = \times$ denotes the computation shown in (45) independent of m. After 2^{P_F} computation, it could be brought to the original value by shifting P_I bits to the left. The steps shown in (43) and (43) can be performed using simple shifting operation which made the BR CORDIC independent of m resulting in the reduction of hardware complexity. As an example, consider $\frac{log_2(R)}{N} = P = 5.36$ then $P_I = 5$, $P_F = 0.36$ and $2^{0.36} = 1.2834$. Now 2^{P_F} should be brought to its original value by shifting $P_I = 5$ bits to the left i.e, $2^P = 41.0696$. Therefore, the $2^{(\frac{log_2(R)}{N})}$ or $2^{(log_2(R) \times N)}$ can be computed by using above normalization procedure and the BR CORDIC. From (39) and (45), the logarithm and exponential computations are independent of the CORDIC converge limit (m) which reduces the hardware complexity and latency in root power computations. A similar normalization procedure for inverse binary logarithm was presented in [8]. But the implementation details of the normalization procedure and 2^{P_F} computation were not provided. In the proposed approach, we presented the implementation details of normalization procedure and Binary Hyperbolic CORDIC algorithm to perform 2^{P_F} computation. The hardware complexity of the proposed Binary Hyperbolic CORDIC is same as conventional Hyperbolic CORDIC which is discussed in section IV-C. The proposed Binary Hyperbolic CORDIC algorithm facilitates us to apply the pre-log normalization and pre-exponential normalization procedures for the binary logarithm and binary exponential computations which reduces the number of iterations.

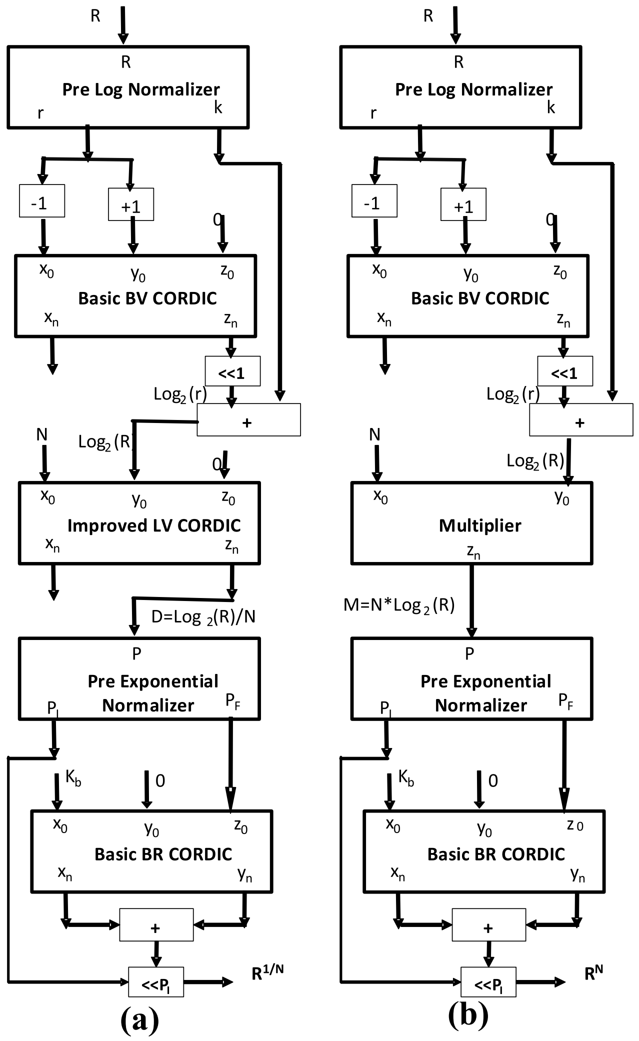

Fig. 1. Computational flow for the proposed methodology a) N^{th} Root computation b) N^{th} Power computation.

B. Illustration of the Proposed Methodology

The computational flow of the proposed methodology is depicted in Fig.1(a) and Fig.1(b) for the $R^{\frac{1}{N}}$ and R^N computations respectively. The functionality of the proposed methodology has been illustrated with an example here. Consider two real numbers $R = 67.55$ and $N = 4.78$. In the $R^{\frac{1}{N}}$ and R^N computations the first step is binary logarithm computation.

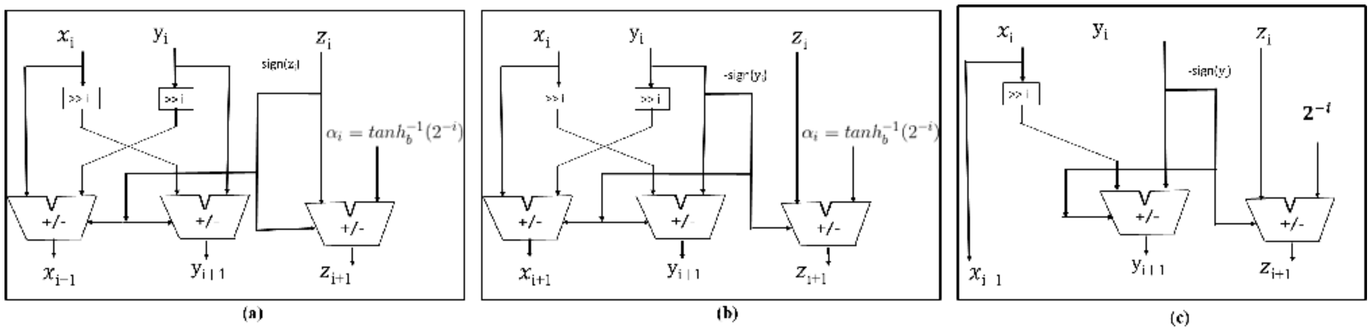

Fig. 2. Iteration structure for a) BR CORDIC, b) BV CORDIC and c) LV CORDIC.

Step1: The binary logarithm can be computed using pre-log normalization and BV CORDIC (using (37), (38) and (39),) as shown in Fig.1(a) and Fig.1(b). The pre log normalization can be performed using shifting operation. The input to the pre-log normalizer is $R = 67.55$ then $2^6 < R \leq 2^7$ and the outputs are $r = 1.05546875$ and $k = 6$. Now $log_2(r)$ can be computed using BV CORDIC. Consider the inputs to the BV CORDIC as $x_0 = r + 1 = 2.05546875$, $y_0 = r - 1 = 0.5546875$ and $z_0 = 0$. The output of BV CORDIC is $z_n = \frac{1}{2}log_2(r) = 0.0389$. The $log_2(r)$ is obtained by shifting z_n one bit to left then $log_2(r) = 0.0779$. The $log_2(R)$ can computed by adding $k = 6$ to the $log_2(r)$ then the $log_2(R) = 6.0779$.

Step2: The second step in $R^{\frac{1}{N}}$ is division computation. The division can be performed using LV CORDIC. The inputs to the LV CORDIC are $x_0 = N = 4.78$, $y_0 = log_2(R) = 6.0779$ and $z_0 = 0$. The output of LV CORDIC is $z_n = \frac{log_2(R)}{N} = 1.2715$ which is treated as D. The second step in R^N is multiplication operation. The inputs to the multiplier are $x_0 = N = 4.78$, $y_0 = log_2(R) = 6.0779$ then the output $z_n = 29.0523$ is treated as M.

Step3: The final step of the $R^{\frac{1}{N}}$ and R^N computations is binary inverse logarithm computation. The binary inverse logarithm can be computed using pre-exponential normalization and BR CORDIC (using (43), (44) and (45)) as shown in Fig.1(a) and Fig.1(b). In $R^{\frac{1}{N}}$ computation, the input to the pre- exponential normalization is $P = D = 1.2715$. The outputs are $P_I = 1$, $P_F = 0.2715$. Now 2^{P_F} can be computed using basic BR CORDIC considering inputs as $x_0 = K_b$, $y_0 = 0$ and $z_0 = P_F = 0.2715$. The outputs of BR CORDIC are $x_n = 1.0178$, $y_n = 0.1893$. The 2^{P_F} can be obtained by adding x_n, y_n then the $2^{P_F} = 1.2071$. The 2^P could be obtained by shifting the 2^{P_F} by $P_I = 1$ bits to the left then $2^D = 2.4141 = R^{\frac{1}{N}} = 67.55^{\frac{1}{4.78}}$. In R^N computation, the input to the pre- exponential normalization is $P = M = 29.0523$. The outputs are $P_I = 29$, $P_F = 0.0523$. Now 2^{P_F} can be computed using basic BR CORDIC considering inputs as $x_0 = K_b$, $y_0 = 0$ and $z_0 = P_F = 0.0523$. The outputs of BR CORDIC are $x_n = 1.0007$, $y_n = 0.0363$. The 2^{P_F} can be obtained by adding x_n, y_n then $2^{P_F} = 1.0369$. The 2^P could be obtained by shifting the 2^{P_F} by $P_I = 29$ bits to the left. Then $2^M = 5.5669 \times 10^8 = R^N = 67.55^{4.78}$.

C. Proposed Architecture

We implemented the architecture for the $R^{\frac{1}{N}}$ and R^N computations in pipeline manner using the proposed methodology shown in section III-A. From the proposed methodology, the binary logarithm and binary inverse logarithm can be performed using Binary Hyperbolic CORDIC. The iterative formula for the proposed Binary Hyperbolic CORDIC is shown in (30). Using (30) and (6), the iteration structure of the Binary Hyperbolic CORDIC for rotation and vectoring mode are shown in Fig.2(a) and Fig.2(b) respectively. Similarly using (9), the iteration structure of LV CORDIC is shown Fig.2(c). The iteration stages in the BV CORDIC, BR CORDIC and LV CORDIC are cascaded with each other to form a pipeline architecture. The critical path for the $R^{\frac{1}{N}}$ computation is a shift-add operation which is same as the state of the art approach [9]. The critical path for the R^N computation is a multiplication operation which is same as the state of the art approach [14]. The proposed architectures are implemented in pipeline fashion. Therefore, the output is available for every clock cycle. The throughput of the proposed approach for $R^{\frac{1}{N}}$ and R^N computation is 100% which is same as the state of the art approaches [9], [14].

IV. Experimental Results and Discussion

A. Verification of Proposed Methodology

In this subsection, the correctness of the proposed methodology has been verified by modeling in MATLAB and simulating the absolute errors. The Absolute Error (AE) is defined as

$$AE = |\frac{T - M}{T}| \tag{46}$$

where T is the true value of the N^{th} root or N^{th} power and M is measured value of the N^{th} root or N^{th} power using the proposed method. The another important criteria is Mean Absolute Error (MAE) which defined as follows

$$MAE = \frac{\sum_{j=1}^{Num} AE}{Num} \tag{47}$$

where Num denotes the number of test cases. The steps involved in the proposed approach and the state of the art approaches [9], [14] are depend on the m and n values. Before simulating the errors, the dependency of m and n values have been analyzed. In $R^{\frac{1}{N}}$ computation, the state of the approach [9] performed the software implementation as well as hardware implementation for $R \in [10^{-6}, 10^6]$ and $N \in [2, 1002]$ [9]. In order to compare our proposed architecture with the state of the art architecture [9] on a uniform platform, we also consider the same values of R and N. In the state of

TABLE VI

DEPENDENCY OF m AND n FOR PROPOSED METHODOLOGY

$R^{\frac{1}{N}}$ computation					R^N computation				
Step	Proposed		State of the art [9]		Step	Proposed		State of the art [14]	
	m	n	m	n		m	n	m	n
Logarithm	$\times$	20	2	20	Logarithm	$\times$	20	1	20
Division	3	20	2	20	Multiplication	$\times$	$\times$	$\times$	$\times$
Exponential	$\times$	20	2	20	Exponential	$\times$	20	4	20

- In $R^{\frac{1}{N}}$ computation, the state of the approach and proposed approach considered $R \in [10^{-6}, 10^6]$ and $N \in [2, 1002]$.
- In R^N computation, the state of the approach and proposed approach considered $R \in [10^{-2}, 10^2]$ and $N \in [2, 5]$.
- $m = \times$ denotes that the computation is independent of m.
- In the proposed approach, the logarithm and exponential computations are independent of m which reduces the hardware complexity and latency.

TABLE VII

VERIFICATION OF THE PROPOSED METHODOLOGY (MATLAB SIMULATION)

Parameter	$R^{\frac{1}{N}}$ computation		R^N computation	
	Proposed	State of the art [9]	Proposed	State of the art [9]
Input Range of R	$[10^{-6}, 10^6]$	$[10^{-6}, 10^6]$	$[10^{-2}, 100]$	$[10^{-2}, 100]$
Input Range of N	$[2, 1002]$	$[2, 1002]$	$[1, 5]$	$[1, 5]$
Num	$5 * 10^6$	$5 * 10^6$	$5 * 10^6$	$5 * 10^6$
$max(AE)$	$7.1943 * 10^{-5}$	$1.0216 * 10^{-4}$	$8.1478 * 10^{-4}$	$2.3212 * 10^{-2}$
MAE	$2.4446 * 10^{-6}$	$3.1287 * 10^{-5}$	$4.1595 * 10^{-6}$	$1.34 * 10^{-4}$

- $max(AE)$ denotes the maximum Absolute Error.
- MAE denotes the Mean Absolute Error.
- The proposed approach for $R^{\frac{1}{N}}$ computation is one order magnitude superior in terms of $max(AE)$ and MAE.
- The proposed approach for R^N computation is two order magnitude superior in terms of $max(AE)$ and MAE.

the art approach [9], from the Table III, the m is chosen as 2 for HV CORDIC then $R \leq 1.0562 * 10^6$. If $m = 2$, from the Table II, for the HR CORDIC, the $z_0 \leq 6.935112$ then $N = \frac{ln(10^6)}{6.935112} \geq 2$. The maximum N value is chosen as 1002. The convergence of LV CORDIC is $\frac{ln(R)}{N} \leq 2^{m+1}$. The minimum value of N is 2 then $\frac{ln(10^6)}{2} \leq 2^{m+1}$ and m should be 2. The n value is considered as 20. In the proposed approach, the input R is independent of m value. In the proposed approach from (39), (40) and (45), the division operation alone depends on m value. The convergence of LV CORDIC is $\frac{log_2(10^6)}{2} \leq 2^{m+1}$ and m should be 3. The dependency of m and n for the proposed and state of the art approach [9] have been summarized in the Table VI. From the Table VI, it can be observed that the proposed approach requires one extra iteration for the division computation because $\frac{log_2(10^6)}{2} > \frac{ln(10^6)}{2} \leq 2^{m+1}$. However, it can be noted from the Table VI that the proposed approach is independent of m for logarithm and exponential computations. This reduces the number of iterations involved in the logarithm and exponential computations. This also reduces the hardware complexity and latency which is discussed in IV-C in detail.

Similarly, for R^N computation, the dependency analysis of m and n is carried out here. From (39) and (45), it is evident that the proposed approach is independent of m where as the state of the art approach [14] depends on m. Consider R as $R \in [10^{-2}, 100]$ and $ln(100) = 4.0652$. From the Table III, to compute $ln(100)$ the m should be 1 for HV CORDIC. The HR CORDIC has been used for exponential computation. The input to the HR CORDIC is $z_0 = ln(R) \times N$. From the Table II, if $m = 4$ for HR CORDIC the $z_0 \leq 24.255$ then $ln(R) \times N \leq 24.255$ and $N \leq 5.2669$. Therefore, the range of N can be increased by increasing m which results additional hardware complexity and latency. Hence, the proposed methodology for R^N computation does not have any limitation on its input ranges of R and N. The dependency of m and n for R^N computation has been summarized in the Table VI. For the m and n values shown in the Table VI, the proposed and the state of the art approaches [9], [14] are coded in MATLAB and simulated the absolute errors using (46) and (47). The Num is chosen as 5 million, the R and N are generated randomly. The results are summarized in Table VII. From the Table VII, it is evident that the proposed approach for $R^{\frac{1}{N}}$ computation is one order superior and the proposed approach for R^N computation is two order superior in terms of maximum AE and MAE when compared with the respective state of the art approaches [9], [14]. The MAE of the proposed Binary Hyperbolic CORDIC is same as the conventional Hyperbolic CORDIC. However, the proposed approach achieved better MAE than the state of the art approach due to the input ranges of the CORDIC. For example, in root computation, it can be noted from the Table VII, the input range of R and N for the state of the art approach and proposed are same. However, for logarithm computation the proposed Binary Hyperbolic CORDIC has been operated for the input of $[1, 2]$. Where as in the state of the art approach the conventional Hyperbolic CORDIC has been operated for the input of $[10^{-6}, 10^6]$. Similarly, for exponential computation, the proposed Binary Hyperbolic CORDIC has been operated for [0, 1]. But, in the sate of the art approach, the conventional Hyperbolic CORDIC operates within range of $[0, 6.935112]$.

B. Word Length Analysis

In this subsection, before implementing the proposed architectures on hardware, first we will analyze the input word

TABLE VIII

WORD LENGTHS REQUIRED FOR THE STATE OF THE ART ARCHITECTURE AND PROPOSED ARCHITECTURE FOR $R^{\frac{1}{N}}$ COMPUTATION

Operation	State of the art [9]						Proposed					
	Module	Parameter	Sign bit	Integer bit	Fractional bit	Total bit	Module	Parameter	Sign bit	Integer bit	Fractional bit	Total bit
Logarithm	HV CORDIC	R	1	20	27	48	Pre Log Normalizer	R	1	20	27	48
							BV CORDIC	r	1	2	45	48
							Adder	k, $log_2(r)$	1	5	27	33
Division	LV CORDIC	$ln(R)$, N	1	10	27	38	LV CORDIC	$log_2(R)$, N	1	10	27	38
Exponential							Pre Exponential Normalizer	$\frac{log_2(R)}{N}$	1	4	27	32
	HR CORDIC	K_H, $\frac{ln(R)}{N}$	1	11	27	39	BR CORDIC	K_b, P_F	1	2	27	30
	Adder	$cosh(.)$, $sinh(.)$	1	11	27	39	Adder	$cosh_b(.)$, $sinh_b(.)$	1	2	27	30

- The state of the approach [9] and proposed approach considered $R \in [10^{-6}, 10^6]$ and $N \in [2, 1002]$.
- The word length required by the BR CORDIC is lesser than the word length required by the HR CORDIC for same R and N values which reduces the hardware complexity.

lengths required for each step. First, let us analyze the word lengths required by the state of the art approach for $R^{\frac{1}{N}}$ computation. The state of the art approach [9] implemented their design for $R \in [10^{-6}, 10^6]$ and $N \in [2, 1002]$. The state of the art design [9] is chosen the fractional part of data as 27 bits. The maximum value of R is 10^6 and $log_2(10^6) \approx 20$. The integer part of R will be 20 bit. The input data for next module (LV CORDIC) is N and $ln(R)$. The integer part of the input data for the LV CORDIC depends on maximum of $ln(R)$ and N. The maximum value of ln(R) i.e, $ln(2^{20}) = 13.863 \approx 2^4$. Four bits are necessary to represent the $ln(.)$ value. The maximum value of N is $1002 \approx 2^{10}$. Ten bits are required to represent the N value. Therefore, the integer part of input for LV CORDIC is chosen as 10 bit. The input to the final step depends on maximum of $sinh(\frac{ln(R)}{N})$ and $cosh(\frac{ln(R)}{N})$. The $sinh(\frac{ln(2^{20})}{2}) = sinh(\frac{ln(2^{20})}{2}) = 512.0005$ hence the integer part for input of HR CORDIC is 10 bit. But, the integer part of the input data for HR CORDIC as 11 bits to avoid the truncation errors due to the iteration formula. An extra sign bit is added in front of every input data and the word length requirements for each step are tabulated in Table VIII.

Next, we will analyze the input data word lengths required for the proposed architecture for $R^{\frac{1}{N}}$ computation. We will consider the same input range for R and N as the state of the art approach [9] to compare the proposed architecture with the state of the art architecture [9] on a uniform platform and the integer part and fractional part of R chosen as 20 and 27 respectively. Here, we followed the word length selection methodology presented in the state of the art approach [9]. For hyperbolic CORDIC, the convergence range and precision will depend on its positive index(m) and negative index(n) boundaries respectively. The fractional word length will depend on the n value. In the hyperbolic CORDIC, the iterations 4, 13, …., $3k + 1$ need to be repeated. The negative index boundary (n) considered in MATLAB simulation is 20. For $n = 20$, the iterations 4 and 13 need to be repeated in hyperbolic CORDIC and the number of effective iterations will be $n_{eff} = 22$. To achieve n bit precision in the output of CORDIC, the internal registers should have $log_2(n)$ extra bits at the LSB position [15]. Therefore, the fractional word length is $n_{eff} + log_2(n_{eff}) = 27$. The first step is pre-log normalization as shown in Fig.1(a). The output of first step is r which is fed as input to the BV CORDIC. From (37), the $r \in [1, 2]$. The maximum of R is 2^{20}, therefore, to bring the R is between $r \in [1, 2]$, it is required to shift the R by $k = 19$ bits to the right. The least significant 19 bits may be ignored while performing normalization. However, we considered additional 18 bits in the fractional part of r to improve the accuracy in logarithm computation. The fractional part of r is to be set as 45. The integer part of r is considered as 2 bit because $r \in [1, 2]$. After performing the logarithm computation, the least significant 18 bits of the fractional part will be ignored. The consideration of additional 18 bit in fractional part of r improves the accuracy of logarithm computation. The next step is compensation of logarithm by adding K to the $log_2(r)$. The maximum value of k is 19. The integer part of the k is to be set as 5 bits. The input data for LV CORDIC is N and $log_2(R)$. The word length of the input data for the LV CORDIC depends on maximum of N and $log_2(R)$. The maximum value is 1002, therefore, the integer part for N and $log_2(R)$ will be chosen as 10 bits. The maximum input to the pre exponential normalization step is $P = \frac{log_2(2^{20})}{2} = 10$. The integer part of input P to be set as 4 bit. The input to the final CORDIC depends on maximum of $sinh_b(P_F)$ and $cosh_b(P_F)$. The P_F is less than equal to 1 so that $max\{cosh_b(P_F)\} = max\{sinh_b(P_F)\} \approx 1.25$. The integer part required for input of BR CORDIC is 2 bit. The final output is shifted by P_I bits to the original value. An extra sign bit is added in front of every input data and settings are summarized in the Table VIII.

From the Table VIII, it can be noted that an extra step is required in proposed approach in logarithm computation i.e, pre log normalization which involves only shift operations. The input word length to HV CORDIC and BV CORDIC

TABLE IX
WORD LENGTHS REQUIRED FOR THE STATE OF THE ART ARCHITECTURE AND PROPOSED ARCHITECTURE FOR R^N COMPUTATION

Operation	State of the art [14]						Proposed					
	Module	Parameter	Sign bit	Integer bit	Fractional bit	Total bit	Module	Parameter	Sign bit	Integer bit	Fractional bit	Total bit
Logarithm	HV CORDIC	R	1	7	27	35	Pre Log Normalizer	R	1	7	27	35
							BV CORDIC	r	1	2	32	35
							Adder	k, $log_2(r)$	1	5	27	33
Multiplication	Multiplier	$ln(R)$, N	1	5	27	33	Multiplier	$log_2(R)$, N	1	5	27	33
Exponential							Pre Exponential Normalizer	$log_2(R)\times N$	1	4	27	32
	HR CORDIC	K_H, $ln(R)\times N$	1	34	27	62	BR CORDIC	K_b, P_F	1	2	27	30
	Adder	$cosh(.)$, $sinh(.)$	1	34	27	62	Adder	$cosh_b(.)$, $sinh_b(.)$	1	2	27	30

- The state of the approach [14] and proposed approach considered $R \in [10^{-2}, 10^2]$ and $N \in [2, 5]$.
- The word length required by the BR CORDIC is lesser than the word length required by the HR CORDIC for same R and N values which reduces the hardware complexity.

is same. The proposed approach will have more accuracy in the logarithm computation due to the range of $r \in [1, 2]$ and the fractional part of r consists of 45 bits instead of 27 bits. Similarly, an extra step is required in exponential computation i.e, pre exponential normalization which involves only shift operations. The input word length required by the BR CORDIC is lesser compared to the HR CORDIC due to pre exponential normalization which reduces the hardware complexity.

Now, we will analyze the input data word length required for R^N computation for m and n shown in the Table VI. We consider the input range of R as $R \in [10^{-2}, 100]$ as mentioned in the Table VII. The fractional part R is chosen as 27 bits to achieve an average precision of 10^{-7}. The integer part of R depends on maximum R value. The integer part will be $log_2(100) \approx 7$ bit. In the state of the art approach [14], the exponential computation depends on m as shown in Table VI. From Table VI, $m = 4$ which limits the N value ($N \leq 5.2669$). Hence, we consider N as $N \in [1, 5]$. The multiplier word length depends on maximum value $ln(R) = ln(100) = 4.6051$ and $N = 5$. The maximum multiplier output is $ln(100) * 5 = 23.0258$. Therefore, the integer part of multiplier is chosen as 5 bit. The word length for HR CORDIC depends on $sinh(ln(100) \times 5) = cosh(ln(100) \times 5) = 1.71 \times 10^{10}$. The integer part required by HR CORDIC is 34 bits. The word lengths required by the state of art approach [14] are tabulated in Table IX for R^N computation. In proposed approach, the logarithm and exponential computations are independent of CORDIC convergence limit (m). The similar word length analysis of $R^{\frac{1}{N}}$ computation is performed for R^N computation and the input word lengths required in each step are tabulated in the Table IX. From the Table IX, it can be noted that, the word length required by the BR CORDIC is 32 bit lesser compared to the HR CORDIC. This reduces the hardware complexity in exponential computation. The hardware complexity analysis is performed in the following subsection.

C. Hardware Complexity and Timing Analysis

In this subsection, we analyze the performance of the proposed architecture and compare with the state of the art architecture [9], [14] in terms of the hardware complexity, latency and throughput. Throughout the analysis we keep a generalized view on CORDIC stages m, n and word-length as b. A Ripple Carry Adder (RCA) and Conventional Array Multiplier (CAM) are considered here to provide comparison on a uniform platform. A b-bit RCA requires b full adders (FA). A bXb CAM requires $b(b-2)$ FA plus b half adders (HA) and b^2 AND gates. In addition, one FA cell requires 24 transistors, one HA cell consist of 12 transistors and a two input AND gates consists of 6 transistors. Based on the approach presented in [9] and [18], Transistor count for the proposed architecture is expressed in terms of Transistor Count (TC) of RCA and CAM. We can calculate $TC_{RCA} = 24b$ and $TC_{CAM} = 6b(5b-6)$. In the Hyperbolic CORDIC, each iteration requires six add operations for $i > 0$ and for $i \leq 0$, each iteration requires eight add operations. In the LV CORDIC, each iteration requires two add operations for all values of i. In conventional Hyperbolic CORDIC, for $i > 0$ the critical path the critical path is one shift and one add operation but for $i \leq 0$ the critical path is one shift and two add operations [9]. The state of the art approach [9] used a folding-delay technique to maintain critical path as one shift and one add operation. The consequence of the folding-delay technique is the iteration $i \leq 0$ requires two clock cycles and the iteration $i > 0$ requires one clock cycle [9]. For LV CORDIC each iteration requires one clock cycle.

In the state of the art approaches [9], [14], the natural logarithm is computed using HV CORDIC along with two additional add operations as shown in (14). The total TC involved in the natural logarithm computation for the state of the art design [9] is expressed as follows

$$TC_{natural\_log} = (8 \times (m+1) + 6 \times (n) + 2) \times TC_{RCA} \tag{48}$$

TABLE X
TRANSISTOR COUNT AND CLOCK CYCLE ANALYSIS

$R^{\frac{1}{N}}$ computation					R^N computation				
Step	Proposed		State of the art [9]		Step	Proposed		State of the art [14]	
	TC	CLK	TC	CLK		TC	CLK	TC	CLK
Logarithm	155160	26	182016	29	Logarithm	113552	26	126000	25
Division	43776	23	41952	23	Multiplication	31482	1	31482	1
Inverse logarithm	95760	24	146952	29	Inverse logarithm	95760	24	257424	33
Total	294696	73	370920	81	Total	240594	51	414906	59
TS	+20.55%				TS	+42.01%			

- TC denotes Transistor Count.
- CLK denotes Number of clock cycles.
- TS denotes transistor saving.
- The proposed approach saves 8 clock cycle latency.

The number of clock cycles required for the natural logarithm computation is expressed as

$$CLK_{natural\_log} = 2 * m + n + 3 \tag{49}$$

In the proposed approach, the binary logarithm is computed using basic BV CORDIC along with two additional add operations as shown in (37), (38) and (39). The compensation with k is performed by add operation. The TC involved in the $log_2(r)$computation is $TC_{log_2(r)} = (6 \times (n) + 2) \times TC_{RCA}$. The total TC a require for the binary logarithm computation can be expressed as follows

$$TC_{binary\_log} = TC_{log_2(r)} + TC_{RCA} \tag{50}$$

From (37), (38) and (39), the number of clock cycles required for the binary logarithm computation is expressed as

$$CLK_{binary\_log} = n + 4 \tag{51}$$

The division operation has been performed using LV CORDIC in the proposed design and the state of the art design as shown in (15) and (40). The TC involved in the division computation is expressed as

$$TC_{div} = (2 \times (m + n + 1) \times TC_{RCA}) \tag{52}$$

The number of clock cycles required for the division computation is expressed as

$$CLK_{div} = m + n + 1 \tag{53}$$

The second step in R^N computation is multiplication operation which is performed using CAM and number of clock cycles required by CAM is 1. The final step in the state of the art design is natural exponential computation. The natural exponential is computed using HR CORDIC and one add operation as shown in (16). The TC involved in the natural exponential computation is expressed in the following equation

$$TC_{natural\_exp} = (8 \times (m + 1) + 6 \times (n) + 1) \times TC_{RCA} \tag{54}$$

The number of clock cycles required for the natural exponential computation using (16) can be expressed as

$$CLK_{natural\_exp} = 2 * m + n + 3 \tag{55}$$

The binary inverse logarithm in the proposed design is computed using basic BR CORDIC and one add operation as shown in (44) and (45). The TC involved in the binary inverse logarithm computation is given by

$$TC_{binary\_inv\_log} = (6 \times (n) + 1) \times TC_{RCA} \tag{56}$$

The number of clock cycles required for the binary inverse logarithm computation using (44) and (45) is given by

$$CLK_{binary\_inv\_log} = n + 2 \tag{57}$$

The total TC and clock cycles required for each step have been summarized in Table X for the values of m and n shown in the Table VI and word lengths shown in the Table VIII and Table IX. From the Table VI, n is considered as 20. In conventional and Binary Hyperbolic CORDIC as per CORDIC convergence theorem $i = 4$, 13 are to be repeated which results additional complexity and latency. The repeated iterations are also accounted in TC and CLK computation, summarized in Table X for root and power computations. The TS (Transistor Saving) is defined as follows

$$TS = 1 - \frac{TC_{proposed}}{TC_{Stateoftheart}} \tag{58}$$

In the proposed approach, if the pre-log normalization and pre-exponential normalization procedures are not performed the number of iterations and word lengths required for the proposed Binary Hyperbolic CORDIC is same as conventional Hyperbolic CORDIC. Therefore, the hardware complexity of the proposed approach is same as [9] and [14] when normalization is not performed. As can be seen from Table X that the proposed approach saves 20.55% and 42.01% TS for $R^{\frac{1}{N}}$ and R^N computations when compared with the state of the art approaches [9], [14] respectively. The proposed approach also saves 8 clock cycle latency compared with the the state of the art approaches [9], [14].

D. Implementation Results

The proposed architectures and the state of the art architectures are coded in VHDL for per word lengths shown in the Table VIII and Table IX. The ASIC implementation was done for the proposed architecture at TSMC 45nm CMOS technology @ $VDD = 1.08V$ and clock frequency @ $1GHz$ with the help of Synopsis Design Compiler (DC) and IC compiler. The synthesis results of ASIC implementation are shown in Table XI. The state of the art approach [9]

TABLE XI
HARDWARE IMPLEMENTATION FOR THE PROPOSED AND STATE OF THE ART ARCHITECTURES

Parameter	$R^{\frac{1}{N}}$ computation		R^N computation	
	Proposed	State of the art [9]	Proposed	State of the art [14]
ASIC Technology	TSMC 45nm			
Power consumption (mw)	35.9	42.7	30.69	47.76
Leakage Power consumption (μw)	33.948	40.3856	29.02	44.526
Dynamic consumption (mw)	35.8661	42.2961	30.6610	47.3147
MSPS/W	27.885×10^3	23.419×10^3	32.583×10^3	20.983×10^3
Area ($sq.\mu m$)	109598.18	135947.26	94181	152068.73
$max\{AE\}$	8.5×10^{-5}	7.6598×10^{-4}	$8.5849 * 10^{-4}$	$6.3896 * 10^{-2}$
MAE	7.6598×10^{-6}	7.3852×10^{-5}	8.2492×10^{-6}	9.5642×10^{-4}
Maximum Frequency (GHz)	1.98	1.98	1.63	1.63
FPGA	Xilinx Virtex-6 (XC6v1x240t)			
Slice LUT	14706	17684	12416	17303

- The proposed design saves 19.38% on chip area and 15.86% power consumption for $R^{\frac{1}{N}}$ computation.
- The proposed design saves 38% on chip area and 35.67% power consumption for R^N computation.

performed its ASIC implementation @ 1GHz. To compare the proposed approach with the state of the art approach on a uniform platform, the synthesis frequency is selected as 1GHz. From the Table XI, it can be noted that the maximum clock frequency of the proposed design is same as the state of the art approach. The critical path of the proposed design is same as the state of the art deign as mentioned in section III-C. Due to this reason, the maximum operating frequency of the proposed design is same as the state of the art design.

From the Table XI, the proposed design for $R^{\frac{1}{N}}$ computation saves 19.38% on chip area and 15.86% power consumption when compared with the state of the art architecture [9]. Similarly, from the Table XI, the proposed design for R^N computation saves 38% on chip area and 35.7% power consumption when compared with the state of the art architecture [14]. It can be noted that, the proposed methodology is independent of the technology node. However to provide insight into the FPGA implementation of the proposed methodology, FPGA prototyping is performed on Xilinx Virtex-6 (XC6v1x240t). From the Table XI, it can be observed that the proposed design saves 20.25% and 39.32% LUT consumption for root and power computations respectively. In order to evaluate the energy efficiency, sampling rate per watt criterion [9] is adopted here by assuming the sampling rate equals to $1000 \times f$ MSPS (Million Samples Per Second). The sampling rate per watt can be expressed as $\frac{1000 \times f}{p(w)} MSPS/W$ where f is frequency in GHz and p is power consumption in watts. From the Table XI, it can be noted that at 1GHz frequency the proposed approach can process 4.436 million additional root computations per second per watt (joule) when compared with the state-of-the-art method [9]. Similarly, the proposed approach can process 11.6 million additional power computations per joule when compared with the state-of-the-art method [14].

E. Accuracy

In order to better understand the accuracy of the proposed approach, here we will illustrate an example for root computation. The Binary Hyperbolic CORDIC performs $n_{eff} = 22$ iterations so that 22 bits are accurate. The input range for BV CORDIC and BR CORDIC are $[\frac{1}{9.36}, 9.36]$ and $[0, 1.6132]$ respectively. However, in the proposed approach we are limiting the input ranges to $r \in [1, 2]$ for BV CORDIC and $P_F \in [0, 1]$ for BR CORDIC which gives better precision. The bit position error of BV CORDIC and BR CORDIC are simulated for the input range of $r \in [1, 2]$ and $P_F \in [0, 1]$ based on approach presented in [9] and [17]. The bit position error of Binary Hyperbolic CORDIC is shown in Fig.3.

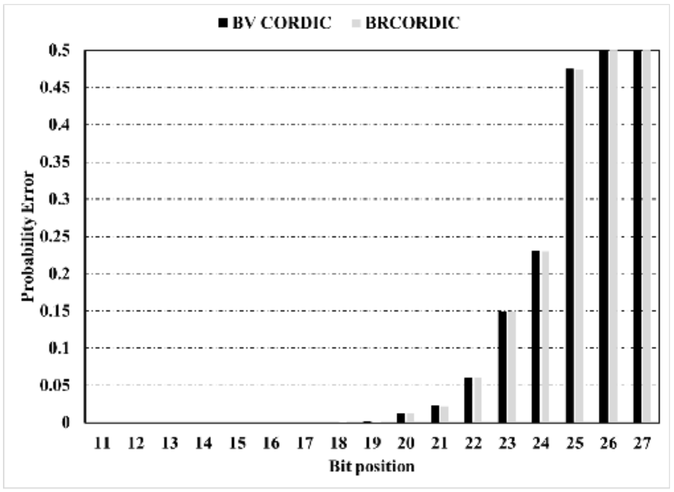

Fig. 3. Bit position Error of BV CORDIC and BR CORDIC.

Let us consider $R = 999999.22721512243151664733 = 11110100001000111111.00111010001\ 01010110001010011$. After the range reduction, we have r having 48 bits and $r = 1.1110100001000111111001110100010101011000101001$ and $k = 19 = 10011$. From Fig.3, it is evident that 23 bits approximately accurate for the BV CORDIC. The output of BV CORDIC is $log_2(r)$ =0.1110111001111011001100$XXXX$ where $XXXX$ will be noise. After adding k to the $log_2(r)$ then $log_2(R) = 10011.1110111001111011001100XXXX$.

TABLE XII
COMPARISON WITH FLOATING-POINT EXPONENTIATION UNITS

Parameter	Proposed	[22]
FPGA	Xilinx Virtex-6 (XC6v1x240t)	Xilinx Virtex-5 (xc5vfx100T)
Word length	FxP (8,27)	FP (8,27)
LUT Sclices	12416	1828
Registers	314	1260
DSP 48 slices	3	11
BRAMs	0	7
Maximum Clock frequency (MHz)	380	214

Consider $N = 2$ and the result of the division is $P = 01001.11110111001111011001 1XXXXXX$. Now $P_I = 9(01001)$ and $P_F = 0.11110111001111011001 1$. The 2^{P_F} can be computed using BR CORDIC. It can be noted from Fig.3 that 23 bits approximately accurate for the BR CORDIC. The 2^{P_F} =1.1111001111111111111111000$XXXX$ and it can be brought to original value by shifting $P_I = 9$ to the left. The $R^{(1/2)}$ i.e, 2^P =1111100111.11111111111001 $XXXX$ but actual $R^{(1/2)}$ is 1111100111.111111111100 1101010110101101. In this example, the absolute error is $2^{-14} = 6.1035 \times 10^{-5}$ and considering 39 bit output 26 bit are approximately accurate.

The HDL simulation is carried out for proposed architectures using (46) and (47) for 5 million test cases. The MAE and maximum AE are tabulated in the Table XI. The MAE for proposed root computation is $7.6958 \times 10^{-6} \approx 2^{-17}$. Considering 39 bit output, 29 bits are approximately accurate for the proposed root computation. From the Table XI, the MAE for the state of the art [9] root computation is $7.3852 \times 10^{-5} \approx 2^{-14}$ and considering 39-bit output, 26 bits are approximately accurate. Similarly, from the Table XI, the MAE for the proposed power computation is $8.2492 \times 10^{-6} \approx 2^{-17}$. Therefore, considering 62-bit output, 52 bits are approximately accurate in the proposed power computation. From the Table XI, the MAE for the state of the power computation [14] is $9.5642 \times 10^{-4} \approx 2^{-10}$. Therefore 45 bits are accurate in 62-bit output.

F. Discussion on Floating-Point Exponentiation Units

The power computation method was presented in [22] for floating-point numbers based on logarithm- exponential relation in (1b). A fair comparison is not possible between the proposed approach and [22] on a uniform platform due to the following reasons. The approach in [22] performs the power computation for floating point numbers where as proposed approach performs the computation for fixed point numbers. The another reason is implementation technology node used are different in proposed approach and [22]. However, to provide more insight to the readers in this subsection, we tabulated the comparison results between the proposed approach and [22] in Table XII. In [22], the logarithm and exponential computations are performed using a second order polynomial approximation. The polynomial approximation methods will be implemented using Look up tables for low-precision. From the Table XII, it can be noted that the approach in [22] requires 7 BRAMs where as the proposed approach requires 0 BRAMs. For higher precision, a generic polynomial approximation approach has been used which increase the resource consumption. A generalized Hyperbolic CORDIC algorithm is presented in [23] to compute arbitrary logarithm and exponential computations. The CORDIC approach has significant advantage over approximation approach in terms of area and power consumption for logarithm and exponential computations when high precision is required [23]. The proposed approach computes logarithm and exponential computations using CORDIC. Therefore, the proposed approach will have significant advantage over the approach in [22] when high precision is required.

V. CONCLUSION

In this paper, we proposed a low complexity N^{th} root and N^{th} power computation architecture design methodology for real time applications. The state of the art approaches [9], [14] performs the N^{th} root and N^{th} power computation based on natural logarithm-exponential relation using Hyperbolic CORDIC. In the state of the art approaches [9], [14], the CORDIC negative index boundary (m) poses limitation on R and N values and also increases the hardware complexity and latency. The proposed approach performs the $R^{\frac{1}{N}}$ and R^N computations based on the binary logarithm- binary logarithm relation shown in [8]. The proposed approach is independent of the CORDIC negative index boundary (m) which reduces the hardware complexity and latency. Subsequently, low complexity architectures have been designed for N^{th} root computation and N^{th} power computation using VHDL and synthesized under the $TSMC40 - nm$ CMOS technology @ 1 GHz frequency. The synthesis results shows that the proposed N^{th} root architecture saves 19.38% on chip area and 15.86% power consumption when compared with the state of the art N^{th} root architecture [9]. Similarly, the proposed N^{th} power architecture saves 38% on chip area, 35.67% power consumption when compared with the state of the art N^{th} power architecture [14]. The proposed approach for $R^{\frac{1}{N}}$ computation is one order superior and the proposed approach for R^N computation is two order superior in terms of maximum absolute error and mean absolute error when compared with the respective state of the art approaches [9], [14]. The critical path for the proposed approach is same as the state of the art approach. The throughput of the proposed approach is 100%.

ACKNOWLEDGMENT

All the Computer Aided Design tools are supported under the Special Manpower Development Program (SMDP) of the Ministry of Electronics and Information Technology (MEITY), Government of India.

REFERENCES

[1] A. S. Glassner, *Principles of Digital Image Synthesis*. San Mateo, CA, USA: Morgan Kaufmann, 1995.

[2] D. Harris, "A powering unit for an Open GL lighting engine," in *Proc. 35th Asilomar Conf. Signals, Syst., Comput.*, Nov. 2001, pp. 1641–1645.

[3] S. P. Mohanty, N. Ranganathan, and R. K. Namballa, "VLSI implementation of visible watermarking for secure digital still camera design," in *Proc. 17th Int. Conf. VLSI Design*, Mumbai, India, Jun. 2004, pp. 1063–1068.

[4] W. Liu and A. Nannarelli, "Power efficient division and square root unit," *IEEE Trans. Comput.*, vol. 61, no. 8, pp. 1059–1070, Apr. 2012.

[5] A. Seth and W.-S. Gan, "Fixed-point square roots using L-b truncation," *IEEE Signal Process. Mag.*, vol. 28, no. 6, pp. 149–153, Nov. 2011.

[6] H. Kabuo *et al.*, "Accurate rounding scheme for the Newton-Raphson method using redundant binary representation," *IEEE Trans. Comput.*, vol. 43, no. 1, pp. 43–51, Jan. 1994.

[7] P. Montuschi, J. D. Bruguera, L. Ciminiera, and J.-A. Piñeiro, "A digit-by-digit algorithm for *m*th root extraction," *IEEE Trans. Comput.*, vol. 56, no. 12, pp. 1969–1706, Dec. 2007.

[8] A. Vázquez and J. D. Bruguera, "Composite iterative algorithm and architecture for q-th root calculation," in *Proc. IEEE Symp. Comput. Arith. (ARITH)*, Jul. 2011, pp. 52–61.

[9] Y. Luo, Y. Wang, H. Sun, Y. Zha, Z. Wang, and H. Pan, "CORDIC-based architecture for computing nth root and its implementation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4183–4195, Dec. 2018.

[10] A. A. Liddicoat and M. J. Flynn, "Parallel square and cube computations," in *Proc. 34th Asilomar Conf. Signals, Syst. Comput.*, vol. 2, Oct./Nov. 2000, pp. 1325–1329.

[11] J. E. Stine and J. M. Blank, "Partial product reduction for parallel cubing," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Porto Alegre, Brazil, Mar. 2007, pp. 337–342.

[12] S. Bui, J. E. Stine, and M. Sadeghian, "Experiments with high speed parallel cubing units," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Tampa, FL, USA, Jul. 2014, pp. 48–53.

[13] 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 *Proc. 48th Midwest Symp. Circuits Syst.*, Vol. 2, Aug. 2005, pp. 1462–1465.

[14] J.-A. Pineiro, M. D. Ercegovac, and J. D. Bruguera, "High-radix iterative algorithm for powering computation," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, Santiago de Compostela, Spain, Jun. 2003, pp. 204–211. doi: 10.1109/ARITH.2003.1207680.

[15] P. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009.

[16] S. Aggarwal, P. K. Meher, and K. Khare, "Scale-free hyperbolic CORDIC processor and its application to waveform generation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 2, pp. 314–326, Feb. 2013.

[17] A. Acharyya, K. Maharatna, B. M. Al-Hashimi, and J. Reeve, "Coordinate rotation based low complexity N-D fast ICA algorithm and architecture," *IEEE Trans. Signal Process.*, vol. 59, no. 8, pp. 3997–4011, Aug. 2011.

[18] S. Mopuri and A. Acharyya, "Low-complexity methodology for complex square-root computation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 11, pp. 3255–3259, Nov. 2017.

[19] S. Aggarwal, P. K. Meher, and K. Khare, "Concept, design, and implementation of reconfigurable CORDIC," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 4, pp. 1588–1592, Apr. 2016.

[20] S. Mopuri, S. Bhardwaj, and A. Acharyya, "Coordinate rotation-based design methodology for square root and division computation," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 66, no. 7, pp. 1227–1231, Jul. 2019.

[21] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of convergence of the CORDIC algorithm," *IEEE Trans. Comput.*, vol. 40, no. 1, pp. 13–21, Jan. 1991.

[22] F. de Dinechin, P. Echeverría, M. López-Vallejo, and B. Pasca, "Floating-point exponentiation units for reconfigurable computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 1, May 2013, Art. no. 4.

[23] Y. Luo, Y. Wang, Y. Ha, Z. Wang, S. Chen, and H. Pan, "Generalized hyperbolic CORDIC and its logarithmic and exponential computation with arbitrary fixed base," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 9, pp. 2159–2169, Sep. 2019.

Suresh Mopuri received the B.Tech. degree (Hons.) in electronics and communication engineering from the Sri Venkateswara University College of Engineering, Tirupati, India, in 2012. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, Indian Institute of Technology Hyderabad (IIT Hyderabad), as an External Student. He joined as a Research Scholar with the Indian Institute of Technology, Hyderabad. He is also a Scientist with the Tracking Systems Group, Indian Space Research Organization (ISRO). His research interests include signal processing algorithms, VLSI architectures, low power design techniques, radar signal processing, and weather signal processing.

Amit Acharyya (M'11) received the Ph.D. degree from the School of Electronics and Computer Science, University of Southampton, U.K., in 2011. He is currently an Associate Professor with the Indian Institute of Technology Hyderabad (IIT Hyderabad), India. His research interests include VLSI systems design for real-time resource-constrained applications, machine learning and signal processing hardware architecture design, edge computing, health-care technology, hardware security, and design for testability and reliability.