Low-Complexity Methodology for Complex Square-Root Computation

Suresh Mopuri and Amit Acharyya

***Abstract*— In this brief, we propose a low-complexity methodology to compute a complex square root using only a circular coordinate rotation digital computer (CORDIC) as opposed to the state-of-the-art techniques that need both circular as well as hyperbolic CORDICs. Subsequently, an architecture has been designed based on the proposed methodology and implemented on the ASIC platform using the UMC 180-nm Technology node with 1.0 V at 5 MHz. Field programmable gate array (FPGA) prototyping using Xilinx' Virtex-6 (XC6v1x240t) has also been carried out. After thorough theoretical analysis and experimental validations, it can be inferred that the proposed methodology reduces 21.15% slice look up tables (on FPGA platform) and saves 20.25% silicon area overhead and decreases 19% power consumption (on ASIC platform) when compared with the state-of-the-art method without compromising the computational speed, throughput, and accuracy.**

***Index Terms*— Complex square root, coordinate rotation digital computer (CORDIC), square root.**

I. Introduction

Complex numbers have been used significantly in scientific community for the real-time data representation and system modeling, including electronic circuits, electromagnetism, communication systems, and signal processing algorithms [1]–[6]. However, existing real valued square-root computation methods [7]–[13] cannot be used directly to compute complex square root without requiring additional hardware. On the other hand, the state-of-the-art architecture for complex square-root computation was designed using the coordinate rotation digital computer (CORDIC) involving two circular and one hyperbolic CORDICs. However, a hyperbolic CORDIC requires more iterations to obtain the same precision and accuracy when compared with a circular CORDIC, resulting in more computational complexity in terms of power consumption and silicon area overhead when implemented on chip [14]. Motivated by the forementioned facts, we introduce here in this brief a low-complexity methodology for computation of the complex square root using only two circular CORDICs unlike the state-of-the-art method where a hyperbolic CORDIC is also necessary.

II. Theoretical Background

Considering a complex number $z = p + jq$, conventional complex square root is computed as follows [4]:

$$\sqrt{p+jq} = \sqrt{\frac{\sqrt{p^2+q^2}+p}{2}} + j\sqrt{\frac{\sqrt{p^2+q^2}+q}{2}}. \tag{1}$$

The implementation of (1) requires three square roots and two multiplications [5], and also have the problem of intermediate overflows. An alternate method was provided in [6] to eliminate this flaw. However, it requires more computations including several tests. To avoid this computational complexity, a CORDIC-based architecture was introduced in [14] based on the following equation:

$$\sqrt{p+jq} = \sqrt{R}\cos\left(\frac{\psi}{2}\right) + j\sqrt{R}\sin\left(\frac{\psi}{2}\right) \tag{2}$$

where $R = \sqrt{p^2+q^2}$ and $\psi = \tan^{-1}(q/p)$. To compute (2) on hardware, it requires two circular and one hyperbolic CORDICs [14]. For paucity of page, detailed discussion on the CORDIC [15] is omitted here. However, the fundamental working principle of a circular CORDIC is as follows:

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \mathrm{Rot}_x(x_0, y_0, \theta) \\ \mathrm{Rot}_y(x_0, y_0, \theta) \end{bmatrix} \tag{3}$$

where x_0, y_0, and x_f, y_f are the initial and final components of the vector, respectively. Angle of rotation $\theta = \mathrm{Vec}_{\theta/x}(x_0, y_0, x_c/y_c)$. $\mathrm{Rot}_{x/y}(.)$ denotes the x/y component of the rotation-mode CORDIC output and $\mathrm{Vec}_{\theta/x}(.)$ denotes the θ/x output of the vectoring-mode CORDIC, equating one of the x_f/y_f co-ordinates with either of x_c/y_c, respectively. It can be noted that a hyperbolic CORDIC requires more iterations to obtain the same precision and accuracy when compared with a circular CORDIC [15], which results additional computational complexity in terms of area and computational speed.

III. Proposed Methodology and Architecture

A. Proposed Methodology

Consider a complex number $z = p + jq$, whose magnitude is $R = \sqrt{p^2+q^2}$. From (2), the proposed methodology has been divided into three steps. The first step is Cartesian to polar conversion, which computes the magnitude (R) and arctangent (ψ), as shown in Fig. 1(a). To compute R and ψ, consider the inputs to the CORDIC as $x_0 = p$ and $y_0 = q$, and operate the CORDIC in the vectoring mode until the final y-component becomes zero, which can be expressed as follows:

$$R = \mathrm{Vec}_x(p, q, x_c/y_c); \quad \psi = \mathrm{Vec}_\theta(p, q, x_c/y_c). \tag{4}$$

The next step in the proposed methodology is the computation of the square root of magnitude $\sqrt{R}$. In the state-of-the-art method, $\sqrt{R}$ is computed using the hyperbolic CORDIC as given in [14]. But, here, we propose a methodology for the computation of $\sqrt{R}$ using only circular CORDIC in the following fashion. From the trigonometric identities

$$\cos 2\Phi = 2\cos^2\Phi - 1. \tag{5}$$

Consider $|R| \leq 1$. If $|R| > 1$, it can be easily scaled down to 1 or less by performing a simple shifting operation. For example, if $2^{(l-2)} < R \leq 2^l$ where l is an even number, the $|R|$ is scaled down to less than 1 by right shifting by l bit. After this shifting, when $\sqrt{R}$ computation is over, the final value is left shifted by $(l/2)$ bit to get the actual value. If $R_{\mathrm{lower}} < R \leq R_{\mathrm{upper}}$, the l value can be computed from R using Table I, where R_{lower} and R_{upper} are the lower and upper boundaries of R, respectively. For example, considering $R = 23$, from Table I as $16 < R \leq 64$, then l will be equal to 6. After right shifting R by 6 bit, R will become 0.359375 and $\sqrt{R} = 0.5994$, which is then readjusted to the actual value $\sqrt{R} = 4.7952$ by left shifting 0.5994 by $(l/2) = 3$ bit.

Manuscript received February 28, 2017; revised June 9, 2017; accepted July 25, 2017. This work was supported in part by the DST-SERB (Grant ECR/2015/00148) and in part by the AA's Visvesvaraya Young Faculty Fellowship by MEITY and CAD tools supported under the SMDP-C2S Program. *(Corresponding author: Amit Acharyya.)*

The authors are with the Department of Electrical Engineering, IIT Hyderabad, Hyderabad 50205, India (e-mail: ee13p0004@iith.ac.in; amit_acharyya@iith.ac.in).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2740343

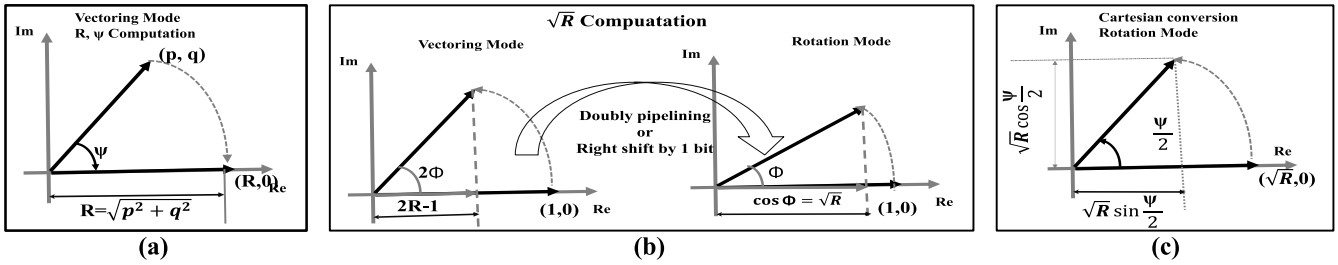

Fig. 1. Geometrical representation of proposed methodology. (a) Cartesian to polar conversion. (b) Square-root computation. (c) Polar to Cartesian conversion.

TABLE I
COMPUTATION OF l FROM R (b = WORD LENGTH)

R_{Range}		Scaling
R_{lower}	R_{upper}	l
0	1	0
1	4	2
4	16	4
16	64	6
⋮	⋮	⋮
$2^{(b-2)}$	2^b	b

Since $|z| = R \leq 1$, assuming $R = \cos^2\Phi$, we will get $\Phi = (k-1)\pi + (-1)^{k-1}\cos^{-1}\sqrt{R}$ where k is an integer. For simplicity and to limit the rotations to first quadrant, we consider $k = 1$, and hence $\cos\Phi = \sqrt{R}$. Now, (5) can therefore be written as

$$\cos 2\Phi = 2R - 1. \tag{6}$$

In order to compute $\sqrt{R}$, Φ and the corresponding $\cos\Phi$ can be computed by the circular CORDIC. Considering the input to the circular CORDIC is an unit vector on the x-axis making $x_0 = 1$ and $y_0 = 0$, as shown in Fig. 1(b). Since R is known, operating the circular CORDIC in the vectoring mode until $x_c = (2R-1)$, $\theta = 2\Phi$ can then be computed by recording the overall angular sweep, as shown in Fig. 1(b). So Φ can be computed as

$$\Phi = \frac{\text{Vec}_\theta(1, 0, 2R-1)}{2} = V(1, 0, (2R-1)) \tag{7}$$

which can be accomplished just by 1-bit right shift of the numerator. Since Φ is known from (6), considering $x_0 = 1$ and $y_0 = 0$ to the circular CORDIC once again, $cos\Phi$ can be computed as

$$\cos\Phi = \sqrt{R} = \text{Rot}_x(1, 0, \Phi) = \text{Rot}_x(1, 0, V(1, 0, (2R-1))). \tag{8}$$

The final step in the proposed methodology is polar to Cartesian coordinate conversion. This can be done by considering $x_0 = \sqrt{R}$ and $y_0 = 0$, and operating the CORDIC in the rotation mode until $\theta = (\psi/2)$, as shown in Fig. 1(c). The outputs of this step real and imaginary parts are of $\sqrt{p+jq}$ as follows:

$$\sqrt{R}\cos\frac{\psi}{2} = \text{Rot}_x\left(\sqrt{R}, 0, \frac{\psi}{2}\right) \tag{9a}$$

$$\sqrt{R}\sin\frac{\psi}{2} = \text{Rot}_y\left(\sqrt{R}, 0, \frac{\psi}{2}\right). \tag{9b}$$

B. Proposed Architecture

Fig. 2 shows the architecture designed based on the proposed methodology, as described in Section III-A. Here, unlike the state-of-the-art design [14], the proposed architecture has been implemented *by reusing only the circular CORDIC*, which eliminates the requirement of the hyperbolic CORDIC and makes the architecture *less-complex*. The detailed hardware complexity analysis is given in Section IV. To enhance the speed of the architecture designed based on the proposed methodology, we use here the concept of doubly pipelining (DP)—a technique where intermediate microrotations are directly fed from the vectoring to the rotation-mode circular CORDIC immediately after these are computed, eliminating the need to wait until the overall angle is computed explicitly after the entire vectoring is done [16], [17] [see Fig. 2(a) and (b)]. Detailed discussion on DP is although omitted here due to paucity of page ([16] and [17] can be referred for the same). However, to apply DP, microrotations should be computed or made available on-the-fly.

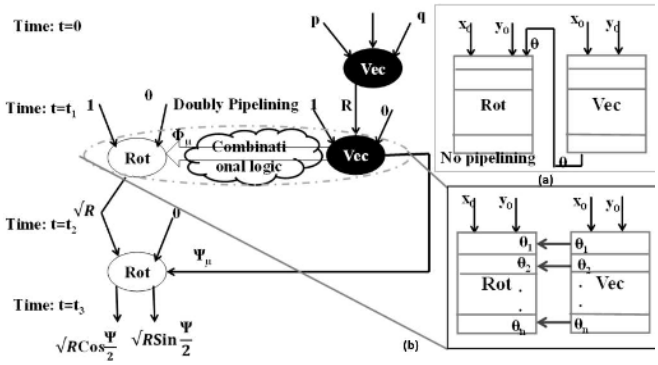

Fig. 2. (a) Without pipelining. (b) Doubly pipelined architecture with a detailed signal flow for the proposed methodology.

TABLE II
MICROROTATION TABLE FOR COMPUTATION ($\theta_1 + \theta_2/2$)

μ_1	μ_2	μ_{12}
0	0	00-Clock wise
0	1	01-No rotation
1	0	10-No rotation
1	1	11- Anti clock wise

From (7), (8), and Fig. 1(b), doubly pipeline can be used in the design to increase the computational speed, but the microrotations are not available for angle ϕ. Hence, a procedure is introduced here to compute the microrotations for ϕ from 2ϕ. Considering two angles θ_1 and θ_2 with microrotations μ_1 and μ_2, respectively, and assuming microrotation 0 and 1 corresponding to clockwise and anticlockwise directions, respectively, microrotation $\mu_{12} = (\theta_1 + \theta_2/2)$ can be computed as shown in Table II. Therefore, the microrotations for ϕ from 2ϕ can be computed by considering $\theta_1 = 2\phi$ and $\theta_2 = 0$. Similarly, from (9) and Fig. 1(c), the microrotations are required for $(\psi/2)$ instead of ψ, which can be calculated by considering $\theta_1 = \psi$ and $\theta_2 = 0$. Hence, to meet the above requirement, as shown in Fig. 1(b) and (c), the rotation-mode CORDIC is designed for rotating the given vector by angle $(\theta/2)$ instead of θ using microrotations shown in Table II. The architecture is divided into two modules—Vectoring and Rotation—as shown in Fig. 3. The pseudocode for the proposed architecture is given in Fig. 4. In the vectoring module,

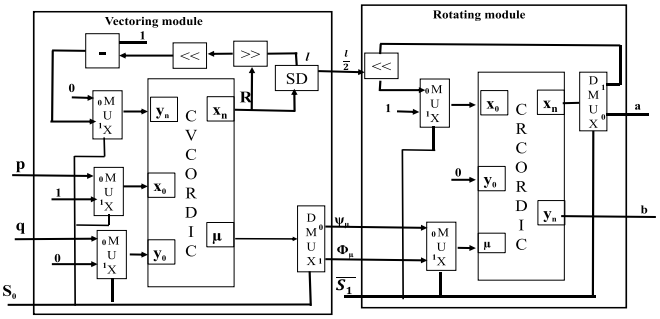

Fig. 3. Architecture designed based on the proposed methodology.

Vectoring Module
S_0=0
CVCORDIC: x_0=p, y_0=q
then $x_n=\sqrt{p^2+q^2}$, $\psi = tan^{-1}\left(\frac{q}{p}\right)$
R=x_n, $\psi_\mu=\mu$
if ($2^{l-2} \le R < 2^l$)
R=R>>l;
S_0=1
CVCORDIC: x_0=1, y_0=0, y_n=2R-1
then $2\Phi=\cos^{-1}(2R-1)$
$\Phi_\mu=\mu$
Rotation Module
S_1=0
CRCORDIC: x_0=1, y_0=0, $\Phi_\mu=\mu$
then $x_n=\sqrt{R}$,
$\sqrt{R}=\sqrt{R}<<\frac{l}{2}$;
S_1=1
CRCORDIC: $x_0=\sqrt{R}$, y_0=0, y_n=2R-1
then $x_n=\sqrt{R}\cos\frac{\psi}{2}$, $y_n=\sqrt{R}\sin\frac{\psi}{2}$

Fig. 4. Pseudocode for architectural implementation based on the proposed methodology.

the circular vectoring-mode CORDIC (CVCORDIC) has been used and it takes one of two inputs based on the selection line s_0. When $s_0 = 0$, the input vector to the CVCORDIC $[x_0, y_0] = [p, q]$. The CVCORDIC will rotate the input vector until y-component becomes zero. Then outputs of the CVCORDIC are $x_n = \sqrt{p^2+q^2} = R$ and the microrotations μ will correspond to the angle $\psi = \tan^{-1}(q/p)$. The output of this module will become $\psi_\mu = \mu$. To compute the scaling value l, a combinational circuit is designed using Table I, which is called the scaling determiner, as shown in Fig. 3. The magnitude R will be shifted by l bit to right to bring the R value less than 1. Now, $(2R-1)$ can be computed by shifting R 1-bit left and then subtracting 1. The final outputs of the circular CORDIC and the hyperbolic CORDIC are needed to be multiplied with CORDIC scaling factors $K_c = 1.646760258121$ and $K_h = 1.207497067763095$, where c and h denote circular and hyperbolic CORDIC, respectively [15].

When $s_0 = 1$, the input vector to CVCORDIC $[x_0, y_0]$ will be $[1, 0]$. Now the CVCORDIC rotates the input vector until y-component becomes $(2R-1)$. The output of CVCORDIC microrotation μ corresponds to angle 2ϕ. Then the vectoring module output is ϕ_μ. The microrotations from the vectoring module ψ_μ, ϕ_μ will be inputs to the rotation module.

Circular rotation mode CORDIC (CRCORDIC), like the vectoring mode, is used that takes one of the two inputs based on the selection line s_1 like the vectoring module. When $s_1 = 0$, the input vector to CRCORDIC $[x_0, y_0] = [1, 0]$ and the input microrotation $\mu = \phi_\mu$. The outputs of CRCORDIC are $x_n = \cos\phi = \sqrt{R}$ and $y_n = \sin\phi = \sqrt{1-R}$. Now, $\sqrt{R}$ will be brought to its original value by shifting $(l/2)$ bit to left. When $s_1 = 1$, the input vector to CRCORDIC $[x_0, y_0] = [\sqrt{R}, 0]$ and input microrotations $\mu = \psi_\mu$. Then the outputs of CRCORDIC are $x_n = \sqrt{R}\cos(\psi/2) = a$ and $y_n = \sqrt{R}\sin(\psi/2) = b$.

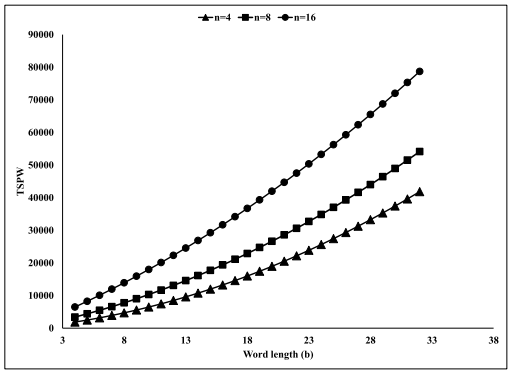

Fig. 5. Transistor saving as a function of wordlength.

TABLE III
TRANSISTOR COUNT (TC) COMPARISON WITH THE STATE OF THE APPROACH FOR DIFFERENT WORDLENGTHS AND CORDIC STAGES

b	State of the art			Proposed		
	$n=4$	$n=8$	$n=16$	$n=4$	$n=8$	$n=16$
4	$0.7*2^{13}$	$1.3*2^{13}$	$1.2*2^{14}$	$1.07*2^{12}$	$0.9*2^{13}$	$0.8*2^{14}$
8	$1.06*2^{14}$	$0.8*2^{15}$	$1.3*2^{15}$	$0.7*2^{14}$	$1.1*2^{14}$	$0.9*2^{15}$
16	$0.8*2^{16}$	$1.1*2^{16}$	$0.8*2^{17}$	$1.2*2^{15}$	$0.8*2^{16}$	$1.2*2^{16}$
32	$1.4*2^{17}$	$0.8*2^{18}$	$1.1*2^{18}$	$1.08*2^{17}$	$1.2*2^{17}$	$0.8*2^{18}$

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Hardware Complexity Analysis

In this section, we analyze the performance of the proposed design in terms of the hardware complexity for arithmetic operations. The proposed architecture has two CORDICs, but the state-of-the-art method has three CORDICs. Throughout the analysis, we keep a generalized view of number of stages in CORDIC as n and wordlength is b, and provide a comparison on a uniform platform considering only a ripple carry adder (RCA) and a conventional array multiplier (CAM). Each CORDIC stage consists of two additions and two subtractions. The CORDIC requires a multiplication operation to multiply the final outputs with k_c and k_h. Considering a b-bit RCA requires b full adders (FA), and 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.

Denoting transistor count by TC, the $TC_{\text{RCA}} = 24$ b and $TC_{\text{CAM}} = 6b(5b-6)$. The total TC involved in the state of the art comprising of three numbers of CORDICs can be expressed as $12 * n * TC_{\text{RCA}} + 5 * TC_{\text{CAM}}$. The TC comparison with the state of the art [14] is shown in Table III. It is evident from Table III that the proposed design is approximately one order better than the state-of-the-art approach in terms of complexity represented by TC. Transistor saving (TS) for the arithmetic operations is expressed in terms of TC. TS with respect to the state-of-the-art method can be computed as

$$\text{TS} = 4 * n * TC_{\text{RCA}} + TC_{\text{CAM}}. \tag{10}$$

Fig. 5 shows the TS per wordlength (TSPW) for the proposed architecture compared with the state-of-the-art architecture [14] for the number of stages in CORDIC and different wordlengths. As can be seen from Fig. 5, the TSPW for different wordlengths is 29% compared with the state-of-the-art approach.

B. Timing Analysis

Considering a vector $[p, q]$ as the input to the DP CORDIC at $t = 0$ and assuming CORDIC comprising of n stages, it requires

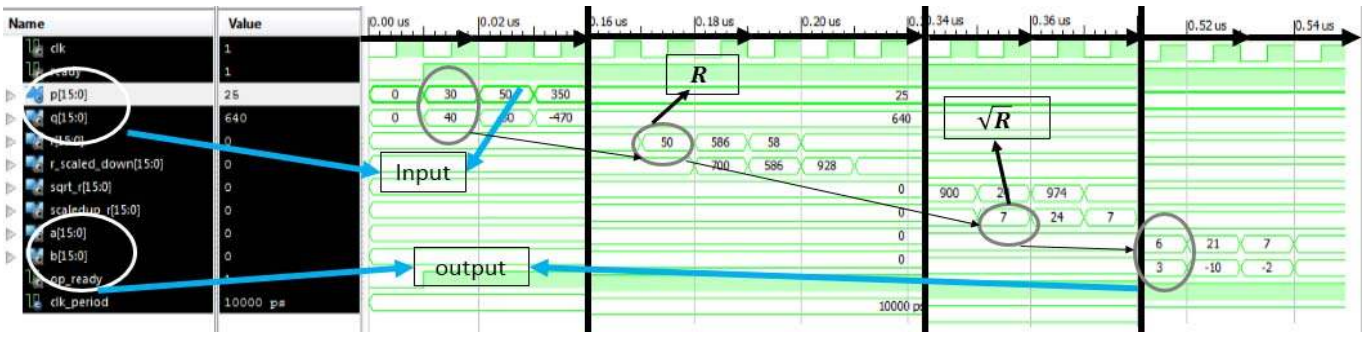

Fig. 6. Register transfer level simulation snapshot for the proposed architecture.

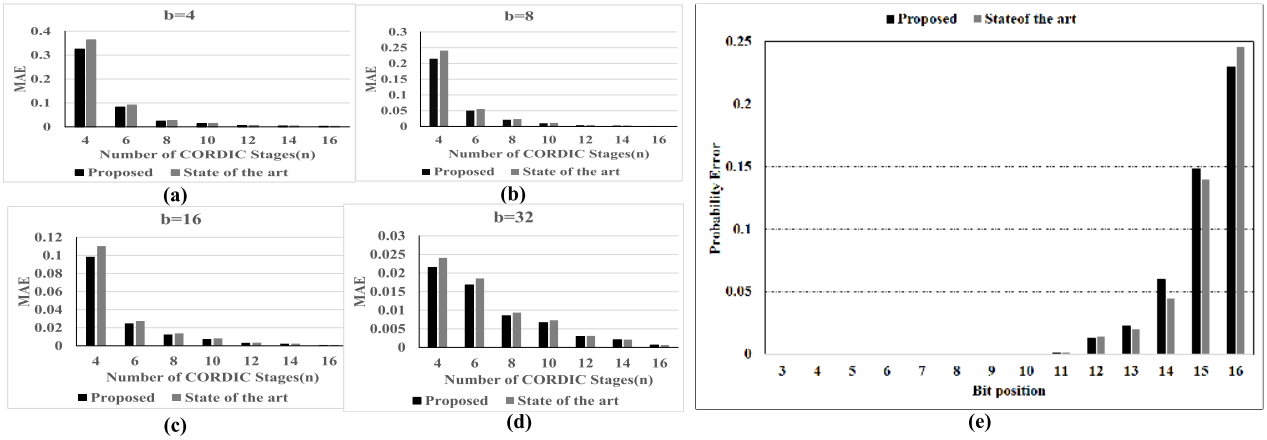

Fig. 7. MAE comparison for the proposed architecture for different input wordlengths. (a) $b = 4$. (b) $b = 8$. (c) $b = 16$. (d) $b = 32$. (e) Probability of error versus comparison for the proposed architecture with respect to bit position for $n = 16$.

n clock cycles to compute R and ψ_μ (see Fig. 2). During the time between $t = 0$ and t_1 (Fig. 2), the microrotations (ψ_μ) are stored in a buffer. Then onward, the CORDIC will be operated in the DP mode that computes $\sqrt{R}$ [during t_1 to t_2 (Fig. 2)]. This step also requires n clock cycles. In the next step at (t_2+1), the microrotation information stored in the first step is applied to the rotation-mode CORDIC instead of the explicit angle from the vectoring-mode CORDIC. The final output appears at t_3. The latency of the designed architecture is $3n$ clock cycles. Reusing only the circular CORDIC results in 100% throughput. On the other hand, unlike the proposed methodology, the state-of-the-art design [14] uses the hyperbolic CORDIC, which has to repeat few iterations when compared with the circular CORDIC, resulting in more latency $\geq$ $(3n + 2)$ clock cycles.

C. Implementation Results

The proposed architecture (Fig. 3) and the state-of-the-art architecture are coded in VHDL for 16-bit input wordlength. As discussed in Section IV-A, the final outputs of the circular CORDIC and the hyperbolic CORDIC are multiplied with the scaling factors K_c and K_h. The field programmable gate array (FPGA) prototype for this architecture has been done on the Xilinx Virtex-6 FPGA (XC6v1x240t). The synthesis results of FPGA prototype implementation and comparison with the state-of-the-art architecture [14], the digit recurrence method-based architecture [4], and the conventional architecture [5] are shown in Table IV. From Table IV, the proposed design methodology saves 77.34%, 75.68%, and 21.15% slice look up tables (LUTs) when compared with the digit recurrence method-based architecture [4], the conventional architecture [5], and the state-of-the-art architecture [14], respectively.

TABLE IV

PERFORMANCE COMPARISON OF THE PROPOSED ALGORITHM WITH THE STATE OF THE ART ON FPGA

Design	LUTs	Registers	Max frequency (MHz)
Proposed Design	5592	1612	157.144
State of the art[14]	6774	1949	141.841
Design in [4]	24688	7296	79.523
Design in [5]	22995	2009	128.7

The ASIC implementation has been done for the proposed architecture and the state-of-the-art architecture at the UMC 180-nm technology at VDD = 1 V and clock frequency at 5 MHz. The ASIC synthesis and physical design have been done with the help of a synopsis design compiler and an IC compiler. The ASIC synthesis results and performance comparison are shown in Table V. From Table V, it can be noted that the proposed design saves the 20.25% on-chip area when compared with the state-of-the-art architecture [14]. It has been found that removal of the hyperbolic CORDIC from the state-of-the-art design flow in the proposed methodology saves 31% of leakage power and 16.32% of dynamic power when compared with the existing architecture [14], resulting in an improvement in power consumption of 19.52%. Although the ASIC synthesis has been done at 5-MHz clock frequency, the design can be operated up to 121-MHz clock frequency.

A snapshot of register transfer level simulation is shown in Fig. 6. The input to the complex square-root module is a complex number $p+jq$ whose real and imaginary parts are p and q. Considering $p =$ 30 and $q = 40$ is shown in Fig. 6. After Cartesian-to-polar coordinate

TABLE V
PERFORMANCE COMPARISON OF THE PROPOSED ALGORITHM WITH THE STATE OF THE ART ON ASIC

Parameter	Proposed	State of the art[15]
Total Design Area	$225479sq.\mu m$	$271138sq.\mu m$
Dynamic Power	$744.6335\mu W$	$889.876\mu W$
Leakage Power	$704.7794nW$	$923.261nW$
Total Power	$0.7453mW$	$0.8908mW$
Throughput	100%	100%
Energy consumption per bit (word length 16 bit)	$0.4472nJ$	$0.5345nJ$
Max frequency (MHz)	121	117

conversion, the next step is square-root computation $sqrt\_r = 7$. The final step is polar-to-Cartesian conversion, after this conversion, the outputs of the complex square-root module are $a = 6$ and $b = 3$. The same simulation is repeated for different p and q values.

D. Error Analysis

Accuracy of the designed architecture based on the proposed methodology is determined by comparing outputs from FPGA with MATLAB's inbuilt "sqrt" function. A set of 2048 randomly generated complex numbers are taken as an input and a mean absolute error (MAE) was calculated with various stages of CORDIC for different input wordlengths. Fig. 7(a)–(d) shows the variation of MAE for the proposed architecture and the hyperbolic CORDIC-based state-of-the-art architecture with different CORDIC stages $n = 4$, 8, 12, 16, and 20 and with wordlengths $b = 4$, 8, 16, and 32. From Fig. 7(a)–(d), it is evident that as the number of CORDIC stages increases, MAE decreases due to the improvement in the resolution of CORDIC. The proposed approach has 0.2% improvement in MAE for $n = 8$ and $b = 8$. Besides MAE, it is also important to analyze the precision of the designed architecture based on the proposed methodology. Hence, following the similar treatment adopted in [18], "*bit position error*" metric is computed considering $n = 16$-stage CORDIC implemented using 16-bit arithmetic. As per [18], using (E) as an absolute error, *bit position error*, $E2$, can be represented as $E2 = |(\ln(E)/\ln(2))|$. The *bit position error* for the proposed architecture has been compared with the state-of-the-art approach [14], as shown in Fig. 7(e). It is evident from Fig. 7(e) that precision up to 12 out of 16 bits is intact for both the proposed and state-of-the-art approaches. The error is more for the state-of-the-art approach at 16-bit position when compared with the proposed architecture. Performance comparison in terms of the area and power consumption as well as resource utilization is reported in Tables IV and V that still stand valid under the constraint of the same computational precision for the state of the art and the proposed design. Since the state-of-the-art approach [14] uses hyperbolic CORDIC, it requires more iterations and operations to obtain the same precision as the circular CORDIC at higher bits.

V. CONCLUSION

In this brief, a low-complexity methodology to compute a complex square root using only circular CORDIC is proposed eliminating the need of the hyperbolic CORDIC from the state-of-the-art architecture [14]. Subsequently, respective architecture has been designed using the DP technique and results have been validated using MATLAB, FPGA, and ASIC platforms resulting in saving of 20.25% on-chip area, 19.52% power consumption (ASIC), and 21.15% slice LUTs (FPGA) without compromising accuracy when compared with the state-of-the-art architecture.

REFERENCES

[1] J. Xiang, L. Guo, Y. Chen, and J. Zhang, "Study of GPS adaptive antenna technology based on complex number AACA," in *Proc. WiCOM*, Oct. 2008, pp. 1–4.

[2] M. Sima, M. Senthilvelan, D. Iancu, J. Glossner, M. Moudgill, and M. Schulte, "Software solutions for converting a MIMO-OFDM channel into multiple SISO-OFDM channels," in *Proc. WiCOM*, Oct. 2007, p. 9.

[3] K.-I. Ko and F. Yu, "On the complexity of computing the logarithm and square root functions on a complex domain," *J. Complex.*, vol. 23, no. 1, pp. 2–24, 2007.

[4] D. Wang and M. D. Ercegovac, "A design of complex square root for FPGA implementation," *Proc. SPIE*, vol. 7444, p. 74440L, Sep. 2009.

[5] D. Wang, N. Zheng, and M. D. Ercegovac, "Design of high-throughput fixed-point complex reciprocal/square-root unit," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 57, no. 8, pp. 627–631, Aug. 2010.

[6] W. Kahan, "Branch cuts for complex elementary functions or much ado about nothing's sign bit," in *The State of the Art in Numerical Analysis*. Oxford, U.K.: Clarendon Press, 1987, ch. 7.

[7] X. Wang, Y. Zhang, Q. Ye, and S. Yang, "A new algorithm for designing square root calculators based on FPGA with pipeline technology," in *Proc. 9th Int. Conf. Hybrid Intell. Syst.*, vol. 1. 2009, pp. 99–102.

[8] M. Ye, T. Liu, Y. Ye, G. Xu, and T. Xu, "FPGA implementation of CORDIC-based square root operation for parameter extraction of digital pre-distortion for power amplifiers," in *Proc. WiCOM*, Sep. 2010, pp. 1–4.

[9] I. Park and T. Kim, "Multiplier-less and table-less linear approximation for square and square-root," in *Proc. IEEE ICCD*, Oct. 2009, pp. 378–383.

[10] T. Sutikno, "An efficient implementation of the non restoring square root algorithm in gate level," *Int. J. Comput. Theory Eng.*, vol. 3, pp. 46–51, Feb. 2011.

[11] T. Sutikno, A. Z. Jidin, A. Jidin, and N. R. N. Idris, "Simplified VHDL coding of modified non-restoring square root calculator," *Int. J. Reconfigurable Embedded Syst.*, vol. 1, pp. 37–42, Mar. 2012.

[12] R. V. W. Putra, "A novel fixed-point square root algorithm and its digital hardware design," in *Proc. Int. Conf. ICT Smart Soc.*, Jun. 2013, pp. 1–4.

[13] T. J. Kwon and J. Draper, "Floating-point division and square root implementation using a Taylor-series expansion algorithm with reduced look-up tables," in *Proc. MWSCAS*, Aug. 2008, pp. 954–995.

[14] B. Yang, D. Wang, and L. Liu, "Complex division and square-root using CORDIC," in *Proc. 2nd Int. Conf. Consumer Electron., Commun. Netw. (CECNet)*, Apr. 2012, pp. 2464–2468.

[15] P. K. 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] T. Sung, H. Yu, and Y. Hu, "Doubly pipelined Cordic array for digital signal processing algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Apr. 1986, pp. 1169–1172.

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

[18] A. Acharyya, K. Maharatna, B. M. Al-Hashimi, and S. R. Gunn, "Memory reduction methodology for distributed-arithmetic-based DWT/IDWT exploiting data symmetry," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 56, no. 4, pp. 285–289, Apr. 2009.