Ultralow-Latency VLSI Architecture Based on a Linear Approximation Method for Computing *N*th Roots of Floating-Point Numbers

Fei Lyu, Xiaoqi Xu, Yu Wang, Yuanyong Luo, Yuxuan Wang, and Hongbing Pan

***Abstract*— State-of-the-art approaches that perform root computations based on the COordinate Rotation Digital Computer (CORDIC) algorithm suffer from high latency in performing multiple iterations. Therefore, root computations based on the CORDIC algorithm cannot meet the strict latency requirements of some applications. In this paper, we propose a methodology for performing *N*th root computations on floating-point numbers based on the piecewise linear (PWL) approximation method. The proposed method divides an *N*th root computation into several subtasks approximated by the PWL algorithm. It determines the widest segments of the subtasks and the smallest fractional width needed to satisfy the predefined maximum relative error Max_Err_r. Our design is coded in Verilog HDL and synthesized under TSMC 40 nm CMOS technology. The synthesized results show that our design can reach the highest frequency of 2.703 GHz with an area consumption of 2608.84 μ m^2 and a power consumption of 2.4476 mW. Compared with one state-of-the-art architecture, our design saves 91.60%, 89.84%, and 63.33% of the area, power, and latency @1.89GHz frequency, respectively, while reducing Max_Err_r by 57.30%. In addition, it saves 94.52%, 92.68%, and 73.17% of the area, power, and delay @1.89GHz frequency, respectively, and reduces Max_Err_r by 1.65% when compared with the other state-of-the-art design.**

***Index Terms*— *N*th root computation, floating-point numbers, piecewise linear (PWL) approximation method, relative error.**

I. Introduction

LOW-ORDER root computation, principally including square and cube roots, is commonly used in many fields, such as scientific computation, digital signal processing, and digital image synthesis [1]–[4]. On the other hand, there is an increasing demand for high-order root computation in fields such as 3-D graphics and atmospheric simulation [5]. However, the computational complexity makes hardware implementation difficult. Thus, root computation with an arbitrary base is a needed solution for both low- and high-order roots. Although floating-point numbers are common in most digital systems, their roots are usually computed by software that struggles to meet the requirements of high speed and low latency [6]. However, the hardware implementation of calculating arbitrary roots of floating-point numbers is a challenging task. It has been confirmed that the COordinate Rotation Digital Computer (CORDIC) algorithm is effective in calculating the *N*th roots of floating-point numbers [7]. Numerous iterations are needed in the CORDIC algorithm, and this leads to high latency as well as high area and power consumption. Until now, there have been no suitable methods to compute the *N*th roots of floating-point numbers for real-time resource-constrained platforms.

The Newton-Raphson (NR) method and digit-recurrence algorithm are another two methods commonly used in the implementation of *N*th root computation. However, they both cannot unify a fixed VLSI architecture to compute an arbitrary *N*th root. The complexity of the hardware implementation increases along with an increasing value of *N*.

In the NR method, the values of x in $f(x) = 0$ are approximated by iterations based on an approximated initial solution. The NR method has gained wide acceptance in computing roots with fixed bases [8]–[12]. Most of the studies focus on root computation for fixed-point numbers. There are also studies of root computation for floating points by the NR method. In [13], an implementation based on the NR method is presented to calculate the cube roots of floating points on the FPGA platform. Square roots and division of single-precision floating-point numbers are implemented based on Goldschmidt's algorithm, which originated from the NR method [14]. Nonetheless, an initial guess is necessary in the NR method, and its value seriously impacts the accuracy of the output. Aside from the NR method, digit recurrence usually appears in the computation of fixed roots [15]–[20]. In this method, several digits of the results are obtained per iteration based on the values of residuals. In [21], according

Manuscript received August 1, 2020; revised October 16, 2020; accepted November 12, 2020. Date of publication December 1, 2020; date of current version January 12, 2021. This work was supported in part by the Scientific Research Foundation for the High-Level Talents of Jinling Institute of Technology under Grant jit-b-201907 and in part by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant SF1017088127469. This article was recommended by Associate Editor J.-Y. Kim. *(Corresponding authors: Fei Lyu; Yu Wang.)*

Fei Lyu is with the School of Electronics and Information Engineering, Jinling Institute of Technology, Nanjing 211169, China (e-mail: lyufei@jit.edu.cn).

Xiaoqi Xu and Yu Wang are with the School of Electronics Engineering, Nanjing Xiaozhuang University, Nanjing 211171, China (e-mail: xxiaoqi277@163.com; wangyu@njxzc.edu.cn).

Yuanyong Luo is with the Department of Turing Architecture Design, HiSilicon, Huawei Corporation, Shenzhen 518129, China (e-mail: luoyuanyong@yeah.net).

Yuxuan Wang and Hongbing Pan are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: 542590982@qq.com; phb@nju.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2020.3038417.

Digital Object Identifier 10.1109/TCSI.2020.3038417

to an optimized composite iterative algorithm, single-precision floating-point square roots of low and high precision are implemented. Based on an analysis of digit-by-digit integer restoring and non-restoring algorithms, Li *et al.* proposed an improved digit-recurrence method to implement the cube root of floating points [22]. Nevertheless, the complexity of the digit-recurrence method explodes with large N. Therefore, it does not meet the requirements of high-order root computation.

The above two algorithms are mainly employed in fixed root computation. Recently, several researchers have devoted attention to arbitrary root computation based on the CORDIC methodology. CORDIC, which is composed of shift and add operations, is easily implemented by a circuit. However, the traditional CORDIC algorithm, limited by a narrow convergence range, is not appropriate for Nth root computations with a large input range. Luo *et al.* introduced non-positive iteration indexes to expand the convergence range of CORDIC [23]. When calculating $R^{1/N}$, the range of the input R is expanded to $[10^{-6}, 10^{6}]$ and $[10^{-10}, 10^{11}]$ by adding three and four nonpositive iterations to CORDIC, respectively. CORDICs in hyperbolic vectoring (HV), linear vectoring (LV) and hyperbolic rotation (HR) modes are involved in the computing flow of the Nth root calculation. Clearly, the non-positive iterations that are added to the three kinds of CORDICs increase the number of clock cycles and consumption of power and area. To overcome the shortcomings of the above research, Suresh *et al.* utilized shift operations to enhance the input range and proposed the binary hyperbolic CORDIC, which can directly compute logarithms and exponents with a base of 2 [24]. The binary hyperbolic CORDIC is derived from the generalized hyperbolic CORDIC (GH CORDIC), which can directly compute $\log_b R$ and b^R with arbitrary values b [25]. The input R is expressed by $R = 2^k \times r$, and the range of r is [1, 2] in [24]. The analysis of the implementation in [24] neglects the hardware resources used to ascertain the value of k. There is no doubt that the acquisition of k consumes many hardware resources and much time. Wang *et al.* exploit the advantages of GH CORDIC and floating-point numbers in Nth root computations [7]. Parallel and pipeline computing methodologies are adopted in the design, and the implementation results indicate the superiority of this method in terms of time latency and power and area consumption. The CORDIC algorithm is appropriate for Nth root computations with high precision. However, each iteration brings about additional clock cycles. So high latency is inescapable in implementing CORDIC algorithm. The two state-of-the-art root computation architectures in [7] and [24] based on CORDIC occupy 58 and 81 clock cycles, respectively, to obtain a high accuracy, with errors of 6.0510×10^{-7} and 7.6598×10^{-6}. Currently, various applications, such as speech recognition, machine learning and financial analysis, are free from the constraint of high accuracy [26], [27]. In these applications, accuracy is reduced to pursue low latency. Data formats with low accuracy are employed, such as the half-precision floating-point format. Even though the accuracy is reduced to 2^{-9}, the numbers of clock cycles required by the methods of [7] and [24] are 30 and 41, respectively, as stated in section V-C. Hence, root computation by CORDIC cannot fulfill the real-time requirements of low- or moderate-accuracy applications.

The piecewise linear (PWL) approximation method is widely used in the computation of unary functions. Various PWL methods based on uniform segments have been proposed [28]–[33]. These studies focus on balancing the number of segments and the accuracy as well as optimizing the coefficients of each linear function. However, the error of each segment has difficulty achieving the required level of consistency. To control the error of each segment, PWL methods with nonuniform segments have been proposed. [34] proposes a PWL-based method of obtaining nonuniform segments on the X axis by uniform partitioning on the Y axis. It uses the error-flattening algorithm and achieves more accurate results than previous methods. However, the amount of hardware resources consumed is not optimized. Thus, Liu *et al.* propose an error-flattened, non-uniform-region linear-approximation algorithm [35]. The hardware implementation is optimized by shift-and-add units to be applied in embedded graphics systems. The above PWL approaches all focus on logarithmic or antilogarithmic converters and are not suitable for other unary functions. To improve the versatility of the PWL method, a universal PWL approximation method is proposed for the computation of unary nonlinearity functions [36]. This method is self-adaptive to obtain the widest segments with a predefined software error. Based on the research in [36], a novel quantizer is proposed to completely simulate the hardware behavior and determine the required bit width to satisfy a predefined error in hardware implementation [37]. Meanwhile, the hardware circuit of the method is optimized to reduce the circuit area.

In this paper, we propose an Nth-root computation architecture based on the PWL method using the floating-point format for the input and output data. Floating-point numbers with various precisions are adopted widely, in personal computers and supercomputers, and have an expanded range of numbers compared to fixed points. Moreover, the exponent and mantissa are stored separately to simplify the individual computation of each part. Typically, when the accuracy of our design is superior to that of [7], our design saves 91.60%, 89.84% and 63.33% of the area, power and latency @1.89GHz frequency, respectively. With the accuracy at the same level, our design saves 94.52%, 92.68%, and 74.55% of the area, power and latency @1.89GHz frequency when compared to the method of [24].

The notable contributions of this paper are as follows:

- The computation of $R^{1/N}$ is a binary function that takes R and N as inputs. This is the first study to apply the PWL method to the computation of a binary function.
- The flow of the Nth root computation contains two subprocesses that are based on the PWL approach. With the restriction of the predefined relative error, we use a software-aided method to optimize the segments of the PWL method, the corresponding coefficients and the width of the fractional bits.
- The hardware implementation is compared with those of state-of-the-art designs with the same level of relative error. The proposed architecture has an absolute advantage in area, power consumption, latency and so on.

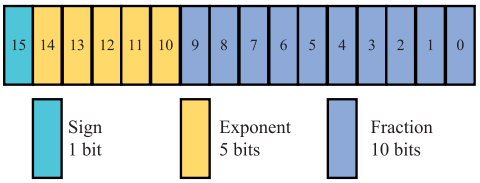

Fig. 1. Half-precision BFP format in IEEE 754-2008.

- Half-precision floating-point numbers are used as the input and output data to illustrate the feasibility of the proposed methodology. The computation flow is also adaptable to any customized floating-point format with a mantissa of width 8 to 16 bits. These more precise input and output data contribute to the accuracy of the Nth root computation.

The rest of this paper is organized as follows: Section II presents the theoretical background of the proposed architecture, including the half-precision floating-point format, the evaluation metrics for the accuracy, the PWL method, and the software-aided hardware design technique. Section III introduces the proposed computation flow and the segmentation and quantization operations. Section IV details the hardware implementation of the proposed methodology. Section V compares our design with state-of-the-art architectures. Section VI concludes this paper.

II. Theoretical Background

A. Half-Precision Floating-Point Format in IEEE 754-2008

The half-precision binary floating-point (BFP) format, which occupies 16 bits, is known as binary16 and is defined in the IEEE 754-2008 standard. The half-precision BFP, as those of other precisions, has three fields: the sign, exponent, and mantissa. The standard half-precision BFP format in IEEE 754-2008 is shown in Fig. 1. The mantissa is composed of a fraction part of 10 bits and a hidden integer part, which is assigned as 1. The exponent, with a width of 5 bits, is encoded with a zero offset of 15 (01111_2). The signs "0" and "1" are used to indicate the positive and negative values of the BFP, respectively. The value of the half-precision BFP B given in bit representation is expressed as

$$B = (-1)^S \times 1.F \times 2^{E-15}, \tag{1}$$

where S, F and E denote the sign, fraction and exponent, respectively. The mantissa M is given in the form $1.F$, specifically $1.F_9F_8 \ldots F_1F_0$.

B. Evaluation Metrics of Accuracy

When the input is limited to a small extent, the PWL method is suitable for approximating a function of one variable. The maximum absolute error MAE is employed to indicate the accuracy of the PWL approach [36]. The MAE is defined as

$$MAE = \max|f - h|, \tag{2}$$

where f is the true value calculated by MATLAB with long format decimal significant figures and h is computed by linear approximation functions. To facilitate the comparison with other studies in [7] and [24], we select the relative error (Err_r), including the average (Avg_Err_r) and maximum (Max_Err_r), as the metrics of accuracy to evaluate the computation results of $R^{1/N}$. Err_r, Avg_Err_r, and Max_Err_r are expressed as

$$Err_r = \left|\frac{f-h}{f}\right| \tag{3}$$

$$Avg\_Err_r = \frac{\sum_{i=1}^{NUM} Err_r}{NUM} \tag{4}$$

$$Max\_Err_r = \max(Err_r), \tag{5}$$

where NUM denotes the number of computation results.

C. PWL Approach

The nonlinear function $f(x)$ is approximated by the piecewise linear function $h(x)$ in the PWL approach. The range of the input $[c, d]$ is divided into subintervals $[c_i, d_i]_{i=1,2,3..}$, and the nonlinear function $f(x)$ in each subinterval is approximated by a linear function $h_i(x)$. The start and end points $(c_i, f(c_i))$ and $(d_i, f(d_i))$ of $f(x)$ are selected in each subinterval, and the linear function of the i^{th} segment can be expressed as

$$h_i(x) = k_i x + b_i \tag{6}$$

$$k_i = \frac{f(d_i) - f(c_i)}{d_i - c_i} \tag{7}$$

$$b_i = f(d_i) - k_i \times d_i. \tag{8}$$

Then, we introduce a method to reduce the MAE given in [36]. The error of each point can be calculated by

$$Err = f_{[c,d]} - h_{[c,d]}, \tag{9}$$

where $f_{[c,d]}$ and $h_{[c,d]}$ denote the true and approximated values of function $f(x)$ with all input in range $[c, d]$. The y-intercept b_i is updated as

$$b_i = f(d_i) - k_i \times d_i + \frac{\max(Err) + \min(Err)}{2}. \tag{10}$$

The MAE is minimized to

$$MAE = \frac{\max(Err) - \min(Err)}{2}. \tag{11}$$

Clearly, the large number of segments improves the approximation precision but brings a great deal of area consumption for the coefficient storage. With the method of MAE minimization, the segment is wider than it was before and has the same MAE.

D. Software-Aided Hardware Design

Segmentation Scheme Under a Predefined Error: It is crucial to determine the widest piecewise interval with a controllable MAE, indicated by mae_sw. mae_sw is used to restrain the MAE produced by the segmentation operation but not the hardware implementation. The wide piecewise interval leads to a small number of segments and thus economizes on

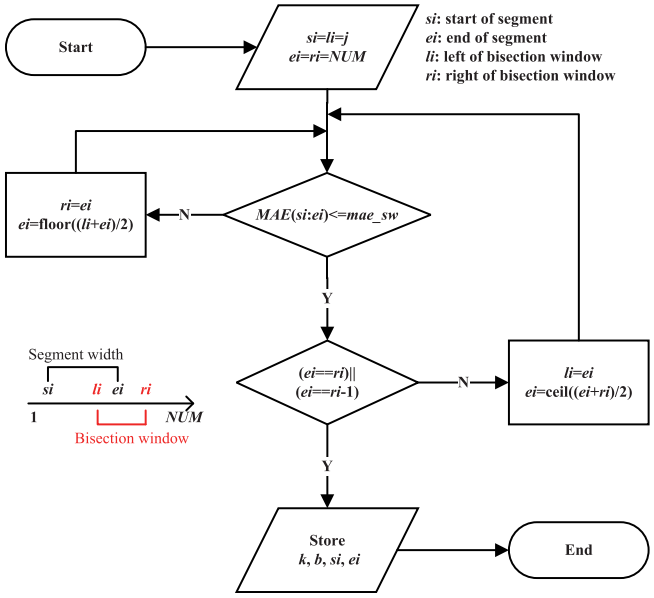

Fig. 2. Calculation flow chart of the segmentation scheme based on the bisection method.

area. Sun *et al.* propose an innovative segmentation scheme confirmed to be more accurate and efficient than those in previous studies [36]. We introduce this method and apply it to study the implementation of Nth root computation. The calculation flow chart of the segmentation scheme based on the bisection method is shown in Fig. 2. Assume that the beginning of the segment indexed by si is confirmed, and the front range of the input has been partitioned optimally. To obtain the greatest width with the restriction of mae_sw, the width of the segment is controlled by moving the pointer to the end of the segment ei. To accelerate the computation, the algorithm is optimized by the bisection method. The end of the segment is restricted by a bisection window with the right and left pointer ri and li. The pointer ei always stays at the midpoint of the bisection window. Meanwhile, ri and li are altered to reduce the width of the bisection window according to the relationship between the current MAE and predefined mae_sw. The values of the MAE are calculated by (11). If the width of the bisection window cannot be decreased further, the optimal value of the end of the segment is obtained. The coefficients of the linear function computed by (7) and (10) are stored.

Self-adaptive Quantizer for Intermediate Data: The fractional word length of the coefficient of the linear functions and temporary results is another factor that affects the accuracy of the hardware implementation results. A software quantizer that can be used to model the hardware computation accurately is proposed in [37] in order to determinate the fractional bits of the intermediate data, and the calculation flow chart of the quantizer is shown in Fig. 3. The large number of fractional bits contributes to improving the accuracy but costs more in area and power for storage. The number of fractional bits, named qw in Fig. 3, starts at the minimum possible value. Then, the MAE is calculated by the intermediate data rounded or truncated to simulate the hardware computation.

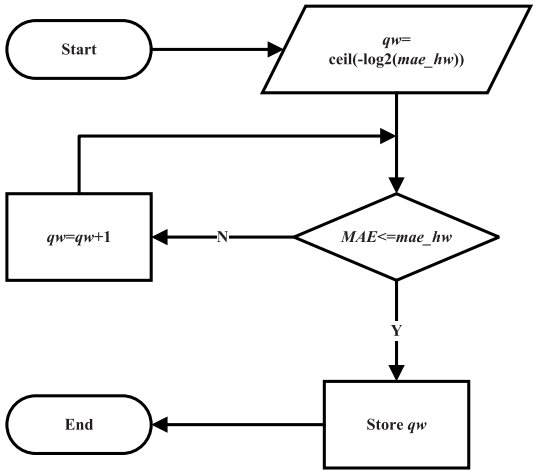

Fig. 3. Calculation flow chart of the quantizer.

The value of qw increases by one until the computed MAE is no larger than the predefined MAE of the hardware mae_hw. To enhance the operation precision, the coefficients of the linear function, including the slope k_i and y-intercept b_i, are quantized by being rounded to be consistent with the number of fractional bits according to the MATLAB statement as

$$aq = \text{round}(a \times 2^{qw}) \times 2^{-qw}, \tag{12}$$

where a and aq are the coefficients before and after quantization, respectively. The temporary results in the calculation process are quantized by truncating the additional fractional bits, and this operation can be simulated in MATLAB using the following statement:

$$aq = \text{floor}(a \times 2^{qw}) \times 2^{-qw}. \tag{13}$$

We use the above statements simulating the computation of hardware by MATLAB software to control the MAE of hardware within a predefined value mae_hw.

III. Proposed Approach

A. Proposed Methodology

There is no doubt that the large coverage of the input brings about an increase in the number of segments and the area on chip for the coefficient storage. Next, we aim to restrict the range of the PWL input without compromising the ranges of N and R in $R^{1/N}$. $R^{1/N}$ can be rewritten as the following identical expression:

$$R^{\frac{1}{N}} = 2^{\frac{log_2 R}{N}}. \tag{14}$$

As a 16-bit BFP number, the value of R is obtained by the codification of BFP in (1):

$$R = M_R \times 2^{E_R - 15}, \tag{15}$$

where M_R includes the mantissa and the hidden mantissa before the decimal point and E_R denotes the exponent. The sign bit in (1) is not considered due to the positive range of R. According to (14) and (15), we obtain

$$R^{\frac{1}{N}} = 2^{\frac{(E_R-15)+log_2(M_R)}{N}}. \tag{16}$$

The computation of the Nth root is resolved into several subtasks by (16). There are three difficulties in the computation of (16): the base-2 logarithm, division by N and base-2 power. Considering that the range of N is small in practical engineering applications, we use a look-up table (LUT) to store the values of $1/N$, and the division is substituted by multiplication. Note that M_R is restricted to the range [1, 2) due to the definition of BFP numbers, and it accords with the requirement of a small input range in the PWL approach. Then, the objective function $\log_2(x)$ approximated by the piecewise linear function $h\_log2(x)$ is separated and defined as

$$\log_2(x) \approx h\_log2(x), x \in [1, 2), \tag{17}$$

where the range of the input is consistent with the M_R range of [1, 2). Next, the focus shifts to the implementation of the base-2 power operation. For ease of description, (16) is simplified as

$$R^{\frac{1}{N}} = 2^P, \tag{18}$$

where P is defined as

$$P = \frac{(E_R - 15) + log_2(M_R)}{N}. \tag{19}$$

The value of P can be computed by the PWL method and the multiplication and shift operations. We separate P into an integer part PI and a fractional part PF. The decomposed form is expressed as

$$P = PI + PF, \tag{20}$$

where the range of PF is normalized to [0, 1). (18) has the alternative formula

$$2^P = 2^{PI} \times 2^{PF}. \tag{21}$$

In (20), as an integer number, PI corresponds to the exponential part of the result obtained by adding 15. Because the input range is [0, 1), the computation of 2^{PF} is suited to the PWL method, and the relationship between the objective function 2^x and the piecewise linear function $h\_pow2(x)$ is expressed as

$$2^x \approx h\_pow2(x), x \in [0, 1). \tag{22}$$

The output of the computation is limited to [1,2) and matches the format of the BFP fraction by hiding the 1 before the decimal point.

B. Segmentation Scheme for $R^{1/N}$

We have broken down the computation of $R^{1/N}$ into several constituent tasks. Therefore, $log_2 x$ and 2^x with small input ranges are suitable for being approximated by PWL functions. Max_Err_r, defined in (5) for $R^{1/N}$, depends on the *MAE*, defined in (2), of $\log_2(x)$ and 2^x based on PWL approximation. We search for the maximum value of *MAE* for a predefined Max_Err_r by the self-adaptive control method.

The pseudocode of the proposed self-adaptive control method is outlined in Table I. In the ***Initialization*** part, the inputs of R and N are initialized on lines 1 to 3. Then, the variables *rmae*, *lmae* and *mae_sw* are initialized for the optimization of the *MAE* in the PWL method as described in section II-D. The *rmae* and *lmae* are the right and left ends of the bisection window used to restrict the value of the *MAE* stored in the variable *mae_sw*. The loop to optimize the *MAE* is controlled by F, which is initialized as 1. In the ***calculation of Max_Err_r***, all possible inputs of R and N are traversed to calculate Err_r. All the values of Err_r are stored in an array variable named Arr_Err_r. The functions seg_log2(a, b, c, *mae_sw*) and seg_pow2(a, b, c, *mae_sw*) are the implementations of the segmentation scheme in Fig. 2. The variables a, b, and c in the above two functions represent the fractional bit number of the input and the start and end of the input range, respectively. The start and end points of the segments and the corresponding coefficients of the linear functions are stored in the variables *SM_log2* and *SM_pow2*. The variables f and h are the results of $R^{1/N}$ derived by MATLAB's built-in functions and by the computation process in Fig. 4. Max_Err_r is the maximum of all the values in the array Arr_Err_r. In the ***optimization of mae_sw*** part, if Max_Err_r is no larger than the predefined Err_r_sw and the width of the bisection window is small enough, the value of *mae_sw* is the maximum value that satisfies the limit Err_r_sw, and the while loop is terminated. The large width of the bisection window indicates that it is possible to increase the value of *mae_sw*. Therefore, the left end of the bisection window *mae* moves to the current value of *mae_sw*, and *mae_sw* moves to the mid-value of the new bisection window. In addition, if Max_Err_r is larger than the predefined Err_r_sw, the value of mae_sw should be decreased on line 34. The right side of the bisection window *rmae* is shifted to the left by line 33 simultaneously. Then, the while loop continues until the value of Max_Err_r and the width of the bisection window meet the requirements.

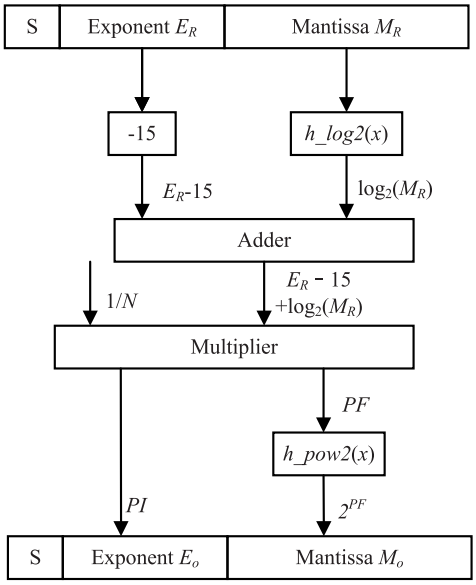

Fig. 4. Computational flow of the proposed methodology.

C. Quantification Operation for $R^{1/N}$

In the quantification operation, the binary calculations are accurately simulated by MATLAB to determine the error of

TABLE I

PSEUDOCODE OF THE PROPOSED SELF-ADAPTIVE SEGMENTATION METHOD FOR $R^{1/N}$

Initialization:		
1.	N=1:1:63;	% generate the input vector and their length
2	M_R=1:1/2^{10}:(2-1/2^{10});	
3.	E_R=-14:1:15;	
4.	*LN*=length(N)	
5.	*LMR*=length(M_R)	
6.	*LER*=length(E_R)	
7.	*rmae*=1;	% right and left window of bisection method
8.	*lmae*=0;	
9.	*mae_sw*=1;	% initialize *mae_sw*
10.	F=1	% flag for while loop
Calculation of Max_Err$_r$:		
11.	While F==1 do	
12.	*SM_log2*=seg_log2(10,1,2,*mae_sw*)	% Segment operation in Fig. 2. The start, end and coefficients are output from the fuction.
13.	*SM_pow2*=seg_pow2(20,0,1,*mae_sw*)	
14	for i=1:1:*LN*	% Calculate *Err$_r$* of each input data.
15	for j=1:1:*LER*	
16	for m=1:1:*LMR*	
17	Calculate h by PWL	
18	Calculate f by the inbuilt function	
19	Calculate *Err$_r$* by (6)	
20	Store *Err$_r$* in *Arr_Err$_r$*	
21	end for	
22	end for	
23	end for	
24.	*Max_Err$_r$*=max(*Arr_Err$_r$*)	
Optimization of mae_sw:		
25.	if *Max_Err$_r$* <=*Err$_r$_sw*	% Find the maximum value of *mae_sw* with the restrict of *Max_Err$_r$* using bisection method.
26.	if (*rmae*-*lmae*<10^{-5})	
27.	F=0;	
28.	else	
29.	*lmae*=*mae_sw*;	
30.	*mae_sw*=(*rmae*+*lmae*)/2;	
31.	end if	
32.	else	
33.	*rmae*=*mae_sw*;	
34.	*mae_sw*=(*rmae*+*lmae*)/2;	
35.	end if	
36.	end while	

TABLE II

PSEUDOCODE OF THE COMPUTATION FLOW OF $R^{1/N}$ WITH QUANTIFICATION

1	*kq_log*=round(*k_log**2^(*qw*))*2^(-*qw*);	% Round the data stored on chip to *qw* bit of fractional part, including k, b, and $1/N$.
2	*bq_log*=round(*b_log**2^(*qw*))*2^(-*qw*);	
3	*kp_pow*=round(*k_pow**2^(*qw*)) *2^(-*qw*);	
4	*bp_pow*=round(*b_pow**2^(*qw*)) *2^(-*qw*);	
5	*inv_N*=round((1/*N*)*2^(*qw*))*2^(-*qw*);	
6	*hq_log2*=(floor(*kq_log***M$_R$**2^(*qw*)) +*bq_log** 2^(*qw*))*2^(-*qw*);	% Corresponding to (17)
7	P=floor((*inv_N**(*E$_R$*+*hq_log2*))*2^(*qw*)) *2^(-*qw*);	% Corresponding to (19)
8	*PI*=floor(P)	% Corresponding to (21)
9	*PF*=*P*-*PI*	
10	*hq_pow2*=(floor(*kq_pow***PF**2^(*qw*)) + *bq_pow**2^(*qw*)) *2^(-*qw*);	% Corresponding to (22)
11	*E$_o$*=*PI*	% *E$_o$* is the Exponet of the result
12	*M$_o$*= floor(*hq_pow2**2^(10))*2^(-10);	%*M$_o$* is the Mantissa of the result
13	*Out*= (2^*E$_o$*)**M$_o$*;	% *Out* is the result calculated by (1)

[a] The variables *kq_log* and *bq_log* express the quantified slope and y-intercept of $h\_log2(x)$ in (17).
[b] The variables *kq_pow* and *bq_pow* express the quantified slope and y-intercept of $h\_pow2(x)$ in (22).
[c] The function round and floor are used to simulate the quantification operation by rounding in (12) and truncating in (13), respectively.

the hardware implementation with different fractional bits. As described in section II-D, there are two methods to quantify infinite and finite decimals with certain fractional bits, either rounded or truncated. In general, the data preprocessed and stored in LUTs are rounded to reduce the accuracy loss, and intermediate results produced by the hardware calculation units are truncated to be easily implementable by hardware. The quantification operation is carried out based on the effectiveness of segmentation. Max_Err_r after quantification, named Err_r_hw, is guaranteed to be larger than Err_r_sw because of the quantification error. We use $Err_r\_sw = QF \times Err_r\_hw$ to define their relationship, and QF is smaller than one.

The logic of the quantification operation is similar to the process in Fig. 3. There are two differences between them. The first is that Max_Err_r serves as the evaluation metric of accuracy, replacing *MAE*. The calculation formula of Max_Err_r is given in (5). The other difference is that the $R^{1/N}$ computation contains two PWL approaches and several operations of addition and multiplication. The pseudocode of the computation process with the quantification operation is shown in Table II.

D. Error-Tolerant Allocation of Segmentation and Quantification

The hardware error Err_r_hw is caused by two operations: segmentation and quantification. There is no doubt that the error caused by segmentation operation Err_r_sw is smaller than Err_r_hw. So we define their relationship as $Err_r\_sw = QF \times Err_r\_hw$, where QF is smaller than one. A large QF is beneficial for reducing the number of segments but causes a large width of the fractional bits. Err_r_hw must be larger than 2^{-10} because the fraction part of the computation result is 10 bits. Therefore, we set Err_r_hw as $2^{-9}(1.9531 \times 10^{-3})$.

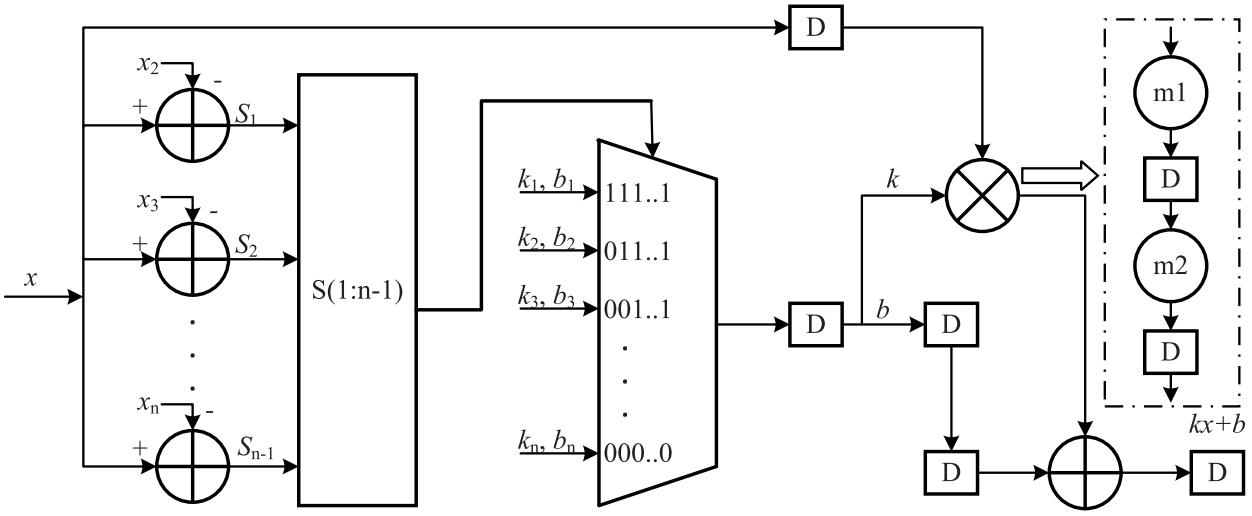

Fig. 5. Hardware architecture of the PWL method with 4 pipeline stages. x_2, x_3,..., and x_n are the 2^{nd}, 3^{rd},..., and n^{th} starting points, respectively. S_1, S_2,..., and S_{n-1} indicate the signs of the subtraction results.

TABLE III
ERROR-TOLERANT ALLOCATION OF SEGMENTATION AND QUANTIFICATION BY *QF*

QF	0.6	0.5	0.4	0.3	0.2	0.1
mae_sw ($\times 10^{-4}$)	9.3460	7.7438	6.0463	4.6158	3.0708	1.5354
Segment number of $\log_2(x)$	7	8	9	10	12	17
Segment number of 2^x	7	8	9	10	12	17
Width of fractional bits	16	14	14	13	13	13

To obtain the optimal value of *QF*, the segment numbers of $\log_2(x)$ and 2^x for various *QF* are obtained and are listed in Table III. The number of segments increases and the width of the fractional bits decreases with a smaller *QF*. When *QF* is not larger than 0.3, the width of the fractional bits reduces to 13 and remains constant. The large width of the computation units, especially multipliers, results in not only great area and power consumption but also long latency. Therefore, we set *QF* as 0.3 to obtain a small width of the fractional bits with a possibly minimal number of segments. The detailed information of the segments of $\log_2(x)$ and 2^x is shown in Tables IV and V, respectively.

IV. HARDWARE ARCHITECTURE

In this section, we introduce the proposed hardware architecture illustrated in Fig. 4 according to the computation flow given in section III-A. To obtain high performance, the hardware architecture is divided into suitable pipeline stages, and parallelization technology is adopted wherever possible. Our design is coded by Verilog HDL and synthesized under TSMC 40 nm CMOS technology. The synthesized results show that our design can reach the highest frequency of 2.703 GHz with 2608.84 μ m^2 area and 2.4476 mW power consumption.

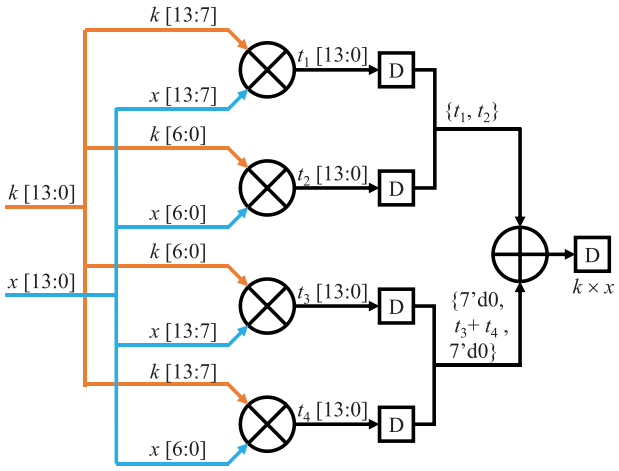

Fig. 6. Hardware architecture of the multiplier implemented by a two-stage pipeline architecture.

A. Implementation of the PWL Module

In the computation flow, the implementations of the PWLs $h\_log2(x)$ and $h\_pow2(x)$ have serious impacts on the performance of the hardware architecture. Sun *et al.* use a multiplexer to select the appropriate coefficients for both the slope and y-intercept [36]. We use a similar design, and the pipeline is redefined as shown in Fig. 5. The width of the multiplier is no less than the width of the fractional bits, 13, which is obtained by the quantification operation. The multiplier is implemented by a two-stage pipeline architecture as shown in Fig. 6, and the output of the adder is stored in a register to reduce the critical path. The circuit of the PWL method requires 4 clock cycles.

B. Normalization

It is easy to calculate the range of the theoretical values of all data in the computation flow. However, the PWL method

TABLE IV
PARAMETERS OF THE PWL METHOD FOR $\log_2(x)$

i	**1**	**2**	**3**	**4**	**5**
slope *k*	1.392333984375	1.2958984375	1.2059326171875	1.1221923828125	1.0440673828125
y-intercept *b*	-1.391845703125	-1.288330078125	-1.1845703125	-1.0806884765625	-0.9766845703125
start point	1	1.07421875	1.154296875	1.240234375	1.3330078125
end point	1.0732421875	1.1533203125	1.2392578125	1.33203125	1.431640625
i	**6**	**7**	**8**	**9**	**10**
slope *k*	0.9716796875	0.904541015625	0.8419189453125	0.78369140625	0.7384033203125
y-intercept *b*	-0.8729248046875	-0.76953125	-0.6661376953125	-0.5626220703125	-0.4766845703125
start point	1.4326171875	1.5390625	1.6533203125	1.7763671875	1.908203125
end point	1.5380859375	1.65234375	1.775390625	1.9072265625	2

TABLE V
PARAMETERS OF THE PWL METHOD FOR 2^x

i	**1**	**2**	**3**	**4**	**5**
slope *k*	0.72314453125	0.7852783203125	0.85009765625	0.9173583984375	0.9871826171875
y-intercept *b*	0.99951171875	0.991943359375	0.9765625	0.9530029296875	0.9210205078125
start point	0	0.1214599609375	0.238037109375	0.35009765625	0.4578857421875
end point	0.121337890625	0.2379150390625	0.3499755859375	0.457763671875	0.561767578125
i	**6**	**7**	**8**	**9**	**10**
slope *k*	1.0595703125	1.134521484375	1.212158203125	1.292236328125	1.359619140625
y-intercept *b*	0.88037109375	0.8306884765625	0.7718505859375	0.7034912109375	0.64013671875
start point	0.5618896484375	0.6622314453125	0.7591552734375	0.85302734375	0.94384765625
end point	0.662109375	0.759033203125	0.8529052734375	0.9437255859375	1

introduces errors in the results of every step and changes the range boundary. We use MATLAB to simulate every step and calculate all the ranges of the intermediate variables, and we list them in Table VI. We find that the errors probably make 2^{PF} less than 1, which is inconsistent with the formalization of the mantissa of the output M_o. Therefore, we normalize 2^{PF} to the mantissa of the output M_o as

$$M_o = \begin{cases} 2^{PF} & , \text{if } 2^{PF} \geq 1 \\ 1 & , \text{if } 2^{PF} < 1 \end{cases}. \tag{23}$$

The signs of the input and output are always equal to 0 to ensure positive values. The exponent of the output E_o is normalized by adding 15.

C. Word Length Setting

The width of the fractional bits is determined to be 13 by the quantification operation with the restriction of the redefined $Err_r\_hw\ 2^{-9}$. The widths of the integral bits are dynamically altered in the computation flow to save storage for the temporary data and to reduce the word lengths of the inputs of the multipliers as much as possible. The settings of the integral width depend on the actual ranges of the intermediate variables, as listed in Table VI.

V. IMPLEMENTATION RESULTS AND COMPARISON

We replicate the hardware implementation in [7], [24] and ensure that their relative error is at the same level as that of our design. To reduce the accuracy, the word length and iteration number of CORDIC need to be decreased. Then, we code the proposed architecture and the state-of-the-art architectures of Nth root computation proposed in [7] and [24]. Additionally, all the coded architectures are synthesized using a Synopsys Design Compiler (DC) and TSMC 40 nm CMOS technology. Based on the synthesized results, comparisons of their hardware implementations are carried out.

A. Accuracy

The iteration number and the fractional word length are both influential factors of the accuracy of the state-of-the-art architectures in [7] and [24] based on the CORDIC algorithm. We set the objective error to be 2^{-10} (9.8×10^{-4}) considering that the length of the fraction part is 10 bits in half-precision BFP format. It is a rule of thumb that n+1 CORDIC iterations are executed to obtain n bits of output precision [38]. Therefore, the number of CORDIC iterations is set to 11 to achieve the accuracy objective in the state-of-the-art designs in [7] and [24]. The negative index boundary m, which determines the iteration index number as $i = -m, -m+1, .., 0$, is adopted

TABLE VI
RANGE AND NUMBER OF BITS OF THE INTERMEDIATE VARIABLES

Intermediate variables	Theoretical range	Actual range	Sign bit	Integral bit	Fractional bit	Total bit
E_R	[1,30]	[1,30]	0	5	0	5
M_R	[1,2)	[1,2)	0	1	10	11
E_R - 15	[-14,15]	[-14,15]	1	4	0	5
$\log_2(M_R)$	[0,1)	(0,1)	1	0	13	14
E_R-15+$\log_2(M_R)$	[-14,16)	(-14,16)	1	4	13	18
$1/N$	(0,0.5]	(0,0.5]	0	0	13	13
$[E_R$-15+$\log_2(M_R)]/N$	[-7,8]	(-7,8)	1	3	13	17
PI	[-7,7]	[-7,7]	1	3	0	4
PF	[0,1)	[0,1)	1	0	13	14
2^{PF}	[1,2)	(0.9,2)	1	1	13	15
M_o	[0,1)	[0,1)	0	0	10	10
E_o	[8,22]	[8,22]	0	5	0	5

[a] The intermediate variables corresponding to the process flow in Fig. 4.

TABLE VII
ERROR OF OUR DESIGN AND THE REPRODUCTIONS IN [7] AND [24]

	Our design	Paper [7]	Paper [24]
Range of R	$[10^{-4}, 65504]$	$[10^{-4}, 65504]$	$[10^{-4}, 64]$
Range of N	[2,63]	[2,63]	[2,63]
Max_Err_r	1.8372×10^{-3}	4.3025×10^{-3}	1.8681×10^{-3}
		-57.30%	**-1.65%**
Avg_Err_r	3.6045×10^{-4}	7.8912×10^{-4}	4.3372×10^{-4}
		-54.32%	**-16.89%**

to enhance the coverage of LV CORDIC in [23]. We set m as 2 to be consistent with the value used in [24]. Then, the focus moves along the word length of the fractional part. According to another rule of thumb, $\log_2(n)$ bits should be added in the computation process to be accurate to the n^{th} bit [39], [40]. Therefore, we set the fractional length as 13 bits by adding 3 ($\approx \log_2 10$) bits at the least significant bit (LSB) position in the replication of the architecture proposed in both [7] and [24]. We set the length of the input as 20 bits, including 1 bit for the sign, 6 bits for the integer part and 13 bits for the fractional part, in the replication of the architecture design in [24]. The computation flow in [24] begins with the shift operation. The length of the fractional part is expanded to 18 bits to avoid precision loss in the shift operation. The fractional width remains at 18 bits in the Basic BV CORDIC unit. It is reduced to 13 bits in the next computation steps.

The iteration number and the fractional word length are modified to lower the accuracy of the state-of-the-art designs to the level of our design. The other parts of the repetition maintain high conformity with the state-of-the-art designs in [7] and [24]. We list the error criteria in Table VII. Meanwhile, we illustrate the approximation results and the relative errors Err_r of computing $R^{\frac{1}{N}}$ by proposed the PWL method with N of 2, 3, 5, 15, 23, 39, 41, 57, and 63 in Fig. 7. Clearly, the accuracy of the three designs is at the same level. When compared to the reproduction in [7], our design reduces Max_Err_r and Avg_Err_r by 57.30% and 54.32%, respectively. Compared with the reproduction in [24], our design decreases Max_Err_r and Avg_Err_r by 1.65% and 16.89%, respectively. The large disparity of accuracy in [7] and [24] demonstrates that the rule of thumb limits the error of the method based on CORDIC to a wide rather than a precise range. However, the PWL method used in our proposed architecture is effective at restricting Max_Err_r.

B. Implementation Results

The application-specific integrated circuit (ASIC) implementation of the proposed architecture is synthesized with TSMC 40 nm CMOS technology. The benchmarks of the state-of-the-art architecture in [7] and [24] are synthesized using the same technology to evaluate the proposed architecture. The synthesized results of both the area and power are listed in Table VIII for comparison. Also, we use the histogram to illustrate the comparison of area and power in Fig. 8 (a) and (b).

1) Area and power consumption: From Table VIII, we can see that the proposed design has extreme advantages in both area and power consumption when compared to the state-of-the-art designs in [7] and [24]. Compared with the architecture in [7], our design costs 91.6% to 93.4% less area and 89.84% to 90.79% less power with frequencies from 1 to 2.38 GHz. Compared with [24], our work reduces the area by 94.52% to 95.58% and the power by 92.68% to 93.28% with frequencies from 1 to 2.564 GHz.

2) Computational Ability and Energy-Efficiency Comparison: The number of floating point operations per second (FLOPS) is used to indicate the computational ability of floating-point numbers. Corresponding to FLOPS in floating point systems, samples per second (SPS) is the criterion of the computational ability of fixed-point numbers. Considering that a fully pipelined structure (a valid output in every cycle)

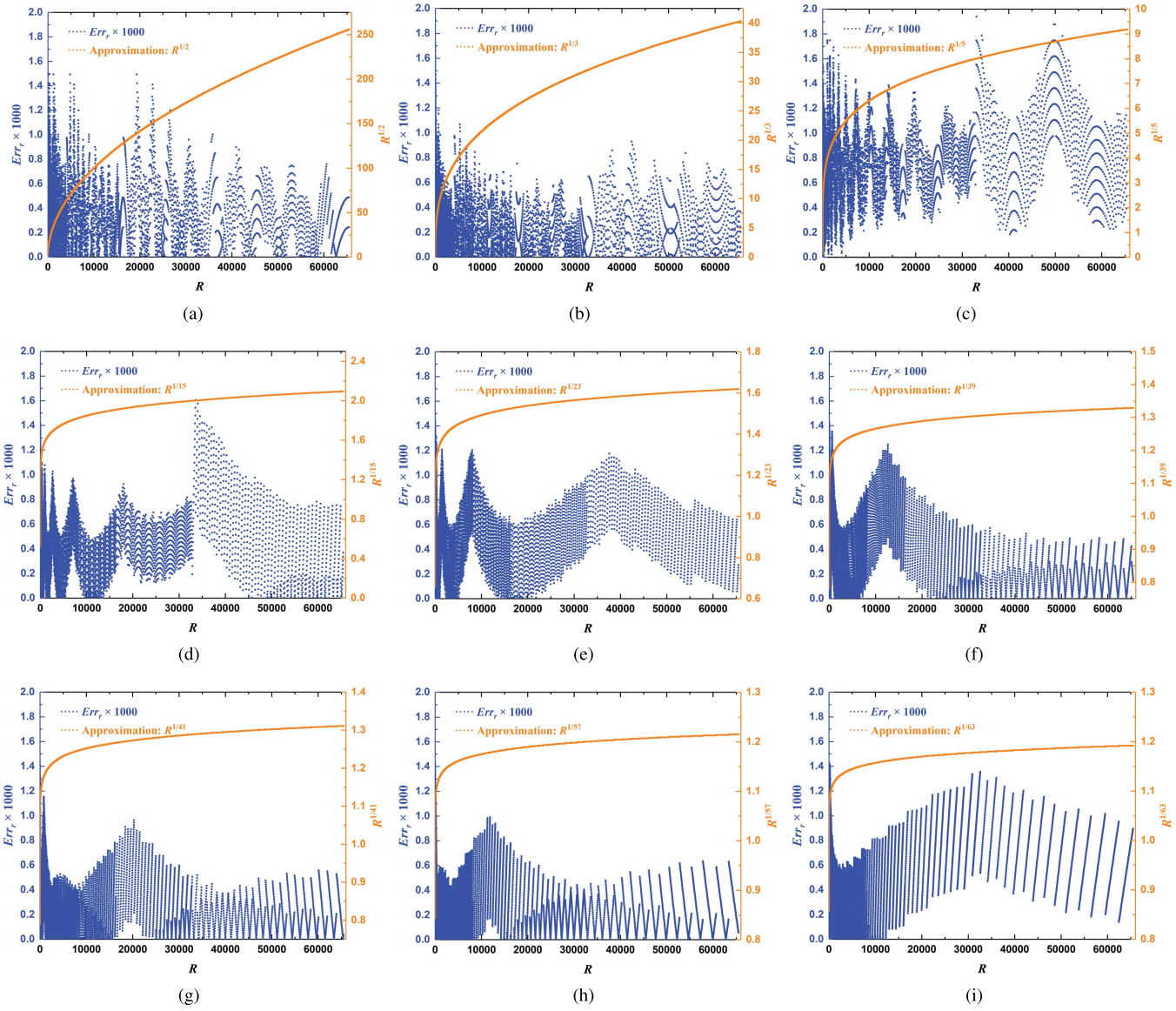

Fig. 7. Approximation results and the relative errors Err_r of computing $R^{\frac{1}{N}}$ with N of 2 (a), 3 (b), 5 (c), 15 (d), 23 (e), 39 (f), 41 (g), 57 (h), and 63 (i).

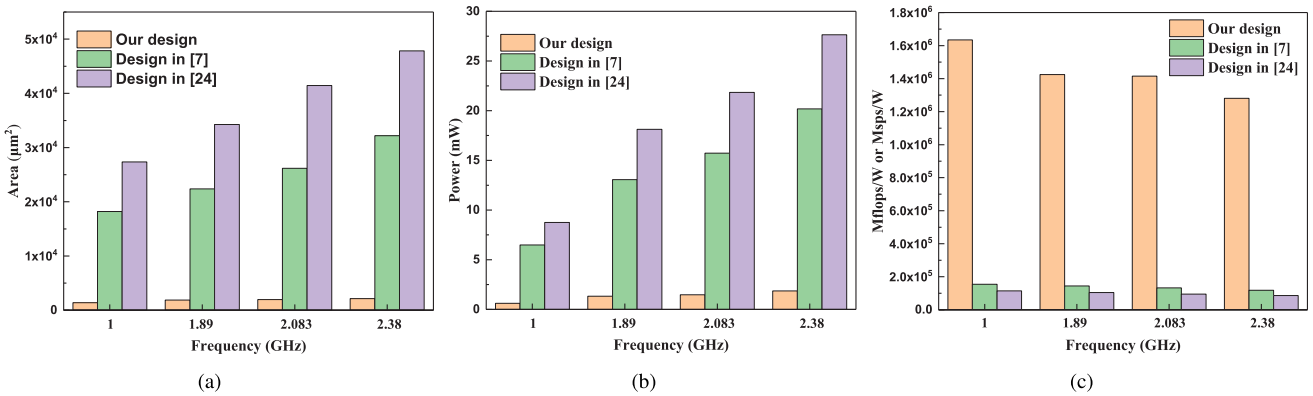

Fig. 8. Comparison of area (a), power (b) and Mflops/W (c) between our design and the state-of-the-art designs in [7] and [24].

is adopted in the proposed architecture and in [7] and [24], the value of FLOPS or SPS equals the frequency of the circuit. Therefore, the FLOPS and SPS are determined by the maximum frequencies of the synthesized results. The maximum frequencies of the proposed design and the compared designs in [7] and [24] are 2.703, 2.38 and 2.564 GHz,

TABLE VIII
IMPLEMENTATION RESULTS OF OUR DESIGN COMPARED WITH THE STATE-OF-THE-ART DESIGNS IN [7] AND [24]

Frequency	Item	Proposed	Paper [7]	Paper [24]
1 GHz	Area (μm^2)	1391.91	18204.24	27346.70
			-92.35%	-94.91%
	Power (mW)	0.6118	6.4884	8.7472
			-90.57%	-93.01%
1.89 GHz	Area (μm^2)	1879.48	22385.63	34271.23
			-91.60%	**-94.52%**
	Power (mW)	1.3268	13.0545	18.1218
			-89.84%	**-92.68%**
2.083 GHz	Area (μm^2)	1953.10	26183.88	41435.18
			-92.54%	-95.29%
	Power (mW)	1.4717	15.7212	21.8398
			-90.64%	-93.26%
2.38 GHz	Area (μm^2)	2125.74	32186.18	47831.21
			-93.40%	-95.56%
	Power (mW)	1.8579	20.1810	27.6430
			-90.79%	-93.28%
2.564 GHz	Area (μm^2)	2463.96		55465.10
				-95.58%
	Power (mW)	2.2083		31.3230
				-92.95%
2.703 GHz	Area (μm^2)	2608.84		
	Power (mW)	2.4476		

respectively, meaning that our design has the strongest computational ability. The maximum frequency of the reproduction of [24] is larger than that stated in the original because the smaller width of the addition operations achieves a short critical path. The maximum frequency of the reproduction in [24] is consistent with that stated in the original because the multiplier has the same width, which determines the critical path of the design.

The number of millions of FLOPS or SPS per watt, expressed as Mflops/W and Msps/W, indicates the energy efficiency when the dataflow is processed in the circuits of floating-point and fixed-point systems, respectively. Mflops/W and Msps/W are expressed by the following uniform equation:

$$\frac{10^6 \times f}{mw}\text{Mflops/w (or Msps/W)}, \tag{24}$$

where f is the frequency of the circuit with GHz as the unit and mw is the power with mW as the unit. By (24), the Mflops/W and Msps/W are obtained and are listed in Table IX and Fig. 8 (c). These results indicate that our design performs the best in terms of energy efficiency with all possible frequencies.

C. Timing Analysis

As stated in section IV-A, each PWL module requires two clock cycles for a two-stage pipelined multiplier, one clock cycle for the multiplexer and one clock cycle for the output. Therefore, the number of clock cycles of PWL, denoted as CLK_{PWL}, is 4. Additionally, we need one clock cycle for the adder and two clock cycles for the two-stage pipelined multiplier to obtain the output of the (19). As a result, the number of clock cycles required by the proposed architecture is given by

$$CLK_{proposed} = 2 \times CLK_{PWL} + 2 + 1 = 11. \tag{25}$$

In the state-of-the-art design [7], the latency of LV CORDIC is covered by parallelism. The number n of iterations of both GHV and GHR CORDIC is set as 11. The 4^{th} iteration repeats once more to guarantee convergence. The iterations of both GHV and GHR CORDIC are carried out (w+1) times. The number of iterations of LV CORDIC is set as (n+1) to fully utilize the computation cycles of GHV CORDIC by parallelism. The multiplier of the two-stage pipeline is used to shorten the critical path. Each adder adopts a cycle, as do the input and output stages. Importantly, the clock cycles in [7] include the cycles in a GHV CORDIC, a GHR CORDIC, a multiplier, two adders, and input and output processing. The total number of clock cycles is

$$CLK_{[7]} = 2 \times (n+1) + 2 + 2 + 2 = 30. \tag{26}$$

Similar to [7], if the numbers of iterations of both GHV and GHR CORDIC are n, their final iteration times are both (n+1) in [24]. The number of iterations of LV CORDIC is (n+m+1) because of the introduction of negative iterations. The time of each negative iteration m is set as 2 to be consistent with [24], and each negative iteration needs 1 clock cycle for the positive iterations. Moreover, two additional clock cycles are needed for the adders before and after BV CORDIC. One additional clock cycle is needed by the shift operation. Therefore, the clock cycle delay of the architecture in [24] is

$$CLK_{[24]} = (n+1+2) + (n+m+1) + (n+1+1) = 41. \tag{27}$$

The clock cycles of the proposed architecture and the state-of-the-art architecture in [7] and [24] are summarized in Table X. To highlight the advantage of the proposed architecture in terms of latency in clock cycle, the *CLKS* (clock cycle latency

TABLE IX
MFLOPS/W OF OUR DESIGN COMPARED WITH THE STATE-OF-THE-ART DESIGNS IN [7] AND [24]

Freq.	Our design (Mflops/W)	Paper [7] (Mflops/W)	Paper [24] (Msps/W)
1 GHz	1.6345×10^6	1.5412×10^5	1.1432×10^5
1.89 GHz	1.4245×10^6	1.4478×10^5	1.0429×10^5
2.083 GHz	1.4154×10^6	1.3250×10^5	9.5376×10^4
2.38 GHz	1.2810×10^6	1.1793×10^5	8.6098×10^4
2.564 GHz	1.1611×10^6		8.1857×10^4
2.703 GHz	1.1043×10^6		

TABLE X
LATENCY OF OUR DESIGN COMPARED WITH THE STATE-OF-THE-ART DESIGNS IN [7] AND [24]

	Our design	Paper [7]	Paper [24]
CLK	11	30	41
$CLKS_{[7]}$	**+ 63.33%**		
$CLKS_{[24]}$	**+ 73.17%**		
***T* (ns)**	4.0696	12.6050	15.9906
$TS_{[7]}$	**+ 67.71%**		
$TS_{[24]}$	**+ 74.55%**		

saving) is defined as follows:

$$CLKS = 1 - \frac{CLK_{proposed}}{CLK_{stateoftheart}}. \tag{28}$$

Clearly, CORDIC requires a number of clock cycles to execute the iterations. Although the pipelines of CORDIC are sometimes optimized by parallelism to reduce the latency of clock cycles, the latency is also not adoptable in high-demand real-time applications. The PWL method solves this problem well. In the computation of the PWL function, a multiplexer, a multiplier and an adder replace the iteration operations in the CORDIC method. The proposed architecture saves 63.33% and 73.17% of the latency in the clock cycles compared to the state-of-the-art architectures based on CORDIC in [7] and [24], respectively.

Then, the time delay criterion for the timing performance in actual working conditions is calculated by the following equation:

$$T = \frac{CLK}{f}. \tag{29}$$

Here, *CLK* stands for the latency of clock cycles obtained by (25), (26) and (27). f represents the operating frequency of the implementation hardware. *CLK* is constant in an implemented design. The time delays achieve the minimum value with the maximum frequency f_{max} and are listed in Table X. To quantify the superiority of our design in terms of time latency, the time latency saving (*TS*) is defined as

$$TS = 1 - \frac{T_{proposed}}{T_{stateoftheart}}. \tag{30}$$

Table X shows that our design saves 67.71% and 74.55% of the time latency when compared with the methods of [7] and [24], respectively. The time latency of our design maintains a competitive performance that is attributable to the high frequency of the circuit.

VI. CONCLUSION

In this paper, we propose a VLSI architecture for the general computation of *N*th roots based on the PWL method proposed in [36]. We use half-precision floating-point numbers as the input and output to verify the feasibility of the proposed architecture. The software, based on MATLAB, includes segmentation and quantification operations. The segmentation process is self-adaptive to designate the smallest number of segments of the subtasks approximated by the PWL method with a predefined relative error. The quantification transaction following the segmentation operation confirms the smallest fractional widths needed to meet the requirement of the relative error of the hardware implementation. We reproduce the state-of-the-art implementations in [7] and [24] and bring their accuracy into correspondence with our design. Our design and the compared designs are modeled using Verilog HDL and synthesized under TSMC 40 nm CMOS technology at various frequencies. The synthesis results show that the maximum frequency of 2.703 GHz is larger than that of both state-of-the-art architectures. Our design reduces the area consumption by 91.6% and 94.52% @1.89GHz frequency compared to the designs in [7] and [24], respectively. The power consumption of our design is reduced by 89.84% and 92.68% @1.89GHz frequency. In addition, the proposed design saves latency of 19 and 30 clock cycles.

REFERENCES

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

[2] J. Harrison, T. Kubaska, S. Story, and P. T. P. Tang, "The computation of transcendental functions on the IA-64 architecture," *Intel Technol. J.*, vol. 4, no. 7, 1999.

[3] H.-C. Shin, J.-A. Lee, and L.-S. Kim, "A minimized hardware architecture of fast phong shader using Taylor series approximation in 3D graphics," in *Proc. Int. Conf. Comput. Design. VLSI Comput. Processors*, 1998, pp. 286–291.

[4] P. Soderquist and M. Leeser, "Area and performance tradeoffs in floating-point divide and square-root implementations," *ACM Comput. Surv.*, vol. 28, no. 3, pp. 518–564, Sep. 1996.

[5] A. S. Glassner, *Principles of Digital Image Synthesis*, vol. 1. Amsterdam, The Netherlands: Elsevier, 1995.

[6] M. F. Cowlishaw, "Decimal floating-point: Algorism for computers," in *Proc. 16th IEEE Symp. Comput. Arithmetic*, Jun. 2003, pp. 104–111.

[7] Y. Wang, Y. Luo, Z. Wang, Q. Shen, and H. Pan, "GH CORDIC-based architecture for computing *N*th root of single-precision floating-point number," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 864–875, Apr. 2020.

[8] C. V. Ramamoorthy, J. R. Goodman, and K. H. Kim, "Some properties of iterative square-rooting methods using high-speed multiplication," *IEEE Trans. Comput.*, vol. C-21, no. 8, pp. 837–847, Aug. 1972.

[9] 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.

[10] M. Allie and R. Lyons, "A root of less evil," *IEEE Signal Process. Mag.*, vol. 22, no. 2, pp. 93–96, Mar. 2005.

[11] 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.

[12] S. Aslan, H. Salamy, and J. Saniie, "A high-level synthesis and verification tool for application specific *k*th root processing engine," in *Proc. IEEE 56th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2013, pp. 1051–1054.

[13] C. M. Guardia and E. Boemo, "FPGA implementation of a binary32 floating point cube root," in *Proc. 9th Southern Conf. Program. Log. (SPL)*, Nov. 2014, pp. 1–6.

[14] R. Bhoyar, P. Palsodkar, and S. Kakde, "Design and implementation of goldschmidts algorithm for floating point division and square root," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2015, pp. 1588–1592.

[15] Y. Li and W. Chu, "Implementation of single precision floating point square root on FPGAs," in *Proc. 5th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 1997, pp. 226–232.

[16] N. Burgess and C. Hinds, "Design issues in radix-4 SRT square root & divide unit," in *Proc. Conf. Rec. 35th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2001, pp. 1646–1650.

[17] A. J. Thakkar and A. Ejnioui, "Design and implementation of double precision floating point division and square root on FPGAs," in *Proc. IEEE Aerosp. Conf.*, Mar. 2006, p. 7.

[18] 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. 1696–1706, Dec. 2007.

[19] A. Pineiro, J. D. Bruguera, F. Lamberti, and P. Montuschi, "A radix-2 digit-by-digit architecture for cube root," *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 562–566, Apr. 2008.

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

[21] A. V'zquez and J. D. Bruguera, "Composite iterative algorithm and architecture for q-th root calculation," in *Proc. IEEE 20th Symp. Comput. Arithmetic*, Jul. 2011, pp. 52–61.

[22] Y. Li and W. Chu, "On the improved implementations and performance evaluation of digit-by-digit integer restoring and non-restoring cube root algorithms," in *Proc. Int. Conf. Comput., Inf. Telecommun. Syst. (CITS)*, Jul. 2016, pp. 1–5.

[23] 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.

[24] S. Mopuri and A. Acharyya, "Low complexity generic vlsi architecture design methodology for N^{th} root and N^{th} power computations," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 12, pp. 4673–4686, Dec. 2019.

[25] 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. 2156–2169, Sep. 2019.

[26] W. Liu, Q. Liao, F. Qiao, W. Xia, C. Wang, and F. Lombardi, "Approximate designs for fast Fourier transform (FFT) with application to speech recognition," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 12, pp. 4727–4739, Dec. 2019.

[27] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, May 2016.

[28] H. Kim, B.-G. Nam, J.-H. Sohn, J.-H. Woo, and H.-J. Yoo, "A 231-MHz, 2.18-mW 32-bit logarithmic arithmetic unit for fixed-point 3-D graphics system," *IEEE J. Solid-State Circuits*, vol. 41, no. 11, pp. 2373–2381, Nov. 2006.

[29] T.-B. Juang, S.-H. Chen, and H.-J. Cheng, "A lower error and ROM-free logarithmic converter for digital signal processing applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 12, pp. 931–935, Dec. 2009.

[30] S. Paul, N. Jayakumar, and S. P. Khatri, "A fast hardware approach for approximate, efficient logarithm and antilogarithm computations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 2, pp. 269–277, Feb. 2009.

[31] D. De Caro, N. Petra, and A. G. M. Strollo, "Efficient logarithmic converters for digital signal processing applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 10, pp. 667–671, Oct. 2011.

[32] R. Gutierrez and J. Valls, "Low cost hardware implementation of logarithm approximation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 12, pp. 2326–2330, Dec. 2011.

[33] D. M. Ellaithy, M. A. El-Moursy, G. Hamdy, A. Zaki, and A. Zekry, "Accurate piecewise uniform approximation logarithmic/antilogarithmic converters for GPU applications," in *Proc. 29th Int. Conf. Microelectron. (ICM)*, Dec. 2017, pp. 1–4.

[34] M. Zhu, J. Xiao, W. Wanggen, and H. A. Yajun, "Error flatten logarithm approximation for graphics processing unit," in *Proc. ICM*, Dec. 2011, pp. 1–6.

[35] C.-W. Liu, S.-H. Ou, K.-C. Chang, T.-C. Lin, and S.-K. Chen, "A low-error, cost-efficient design procedure for evaluating logarithms to be used in a logarithmic arithmetic processor," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1158–1164, Apr. 2016.

[36] H. Sun, *et al.*, "A universal method of linear approximation with controllable error for the efficient implementation of transcendental functions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 1, pp. 177–188, Jan. 2020.

[37] H. Dong, *et al.*, "PLAC: Piecewise linear approximation computation for all nonlinear unary functions," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9 pp. 1–14, Sep. 2020.

[38] 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.

[39] 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.

[40] D. R. Llamocca-Obregón and C. P. Agurto-Ríos, "A fixed-point implementation of the expanded hyperbolic CORDIC algorithm," *Latin Amer. Appl. Res.*, vol. 37, no. 1, pp. 83–91, 2007.

Fei Lyu received the B.S. degree in information engineering from Shanghai University, Shanghai, China, in 2011, and the Ph.D. degree from the School of Electronic Science and Engineering, Nanjing University, Nanjing, China, in 2017. He is currently a Lecturer with the School of Electronics and Information Engineering, Jinling Institute of Technology, Nanjing. His research interests include VLSI design, CMOS sensors, and reconfigurable computing.

Xiaoqi Xu is currently pursuing the B.S. degree with the School of Electronic Engineering, Nanjing Xiaozhuang University. Her research interests include VLSI design and central processing unit.

Yu Wang received the B.S. degree from the School of Information Science and Engineering, Hebei University of Science and Technology, Hebei, China, in 2012, and the Ph.D. degree from the School of Electronic Science and Engineering, Nanjing University, Nanjing, China, in 2017. She is currently a Lecturer with the School of Electronic Engineering, Nanjing Xiaozhuang University, Nanjing. Her research interests include VLSI design, organic electronics, semiconductor devices, and processes.

Yuanyong Luo received the B.S. degree in applied physics from Jilin University, Changchun, China, in 2016, and the Ph.D. degree in electronic science and technology from Nanjing University, Nanjing, China, in 2020.

He is currently a Senior Research Engineer with the Department of Turing Architecture Design, HiSilicon, Huawei Corporation. His research interests include novel VLSI computing method, central processing unit, and neural-network processing unit.

Yuxuan Wang received the B.S. and M.Sc. degrees in electronic science and engineering from Nanjing University, Nanjing, China, where he is currently pursuing the Ph.D. degree with the School of Electronic Science and Engineering. His current research interests include VLSI computing IP optimization and the coordinated acceleration of software and hardware in machine learning.

Hongbing Pan received the B.S. degree in applied physics and the Ph.D. degree in microelectronics and solid state electronics from Nanjing University, Nanjing, China, in 1994 and 2005, respectively.

From 2006 to 2012, he was an Associate Professor with the Institute of VLSI Design, Nanjing University, where he has been a Professor with the School of Electronic Science and Engineering. He is the author of more than 40 articles. His research interests include VLSI design, CMOS sensors, reconfigurable computing, and artificial intelligence.