Article

Low-Latency Hardware Implementation of High-Precision Hyperbolic Functions Sinh*x* and Cosh*x* Based on Improved CORDIC Algorithm

Wenjia Fu [1], Jincheng Xia [1], Xu Lin [1], Ming Liu [2,*] and Mingjiang Wang [1,*]

[1] Shenzhen Key Laboratory of IoT Key Technology, Harbin Institute of Technology, Shenzhen 518000, China; 19S152116@stu.hit.edu.cn (W.F.); 19S152115@stu.hit.edu.cn (J.X.); 20S152153@stu.hit.edu.cn (X.L.)
[2] School of Microelectronics, Shenzhen Institute of Information Technology, Shenzhen 518000, China
* Correspondence: lium@sziit.edu.cn (M.L.); mjwang@hit.edu.cn (M.W.); Tel.: +86-0755-8922-6908 (M.L.); +86-0755-8655-5455 (M.W.)

Abstract: CORDIC algorithm is used for low-cost hardware implementation to calculate transcendental functions. This paper proposes a low-latency high-precision architecture for the computation of hyperbolic functions sinh*x* and cosh*x* based on an improved CORDIC algorithm, that is, the QH-CORDIC. The principle, structure, and range of convergence of the QH-CORDIC are discussed, and the hardware circuit architecture of functions sinh*x* and cosh*x* using the QH-CORDIC is plotted in this paper. The proposed architecture is implemented using an FPGA device, showing that it has 75% and 50% latency overhead over the two latest prior works. In the synthesis using TSMC 65 nm standard cell library, ASIC implementation results show that the proposed architecture is also superior to the two latest prior works in terms of total time (latency × period), ATP (area × total time), total energy (power × total time), energy efficiency (total energy/efficient bits), and area efficiency (efficient bits/area/total time). Comparison of related works indicates that it is much more favorable for the proposed architecture to perform high-precision floating-point computations on functions sinh*x* and cosh*x* than the LUT method, stochastic computing, and other CORDIC algorithms.

Keywords: hyperbolic functions; CORDIC; high-precision floating point; low latency; hardware configurable architecture

check for updates

Citation: Fu, W.; Xia, J.; Lin, X.; Liu, M.; Wang, M. Low-Latency Hardware Implementation of High-Precision Hyperbolic Functions Sinh*x* and Cosh*x* Based on Improved CORDIC Algorithm. *Electronics* **2021**, *10*, 2533. https://doi.org/10.3390/electronics10202533

Academic Editor: Luis Gomes

Received: 14 September 2021
Accepted: 13 October 2021
Published: 17 October 2021

1. Introduction

Scientific computing has penetrated almost all scientific and engineering computing and is widely used in energy survey, game rendering, meteorology and oceanography, finance and insurance, computer-aided design, etc. As for numerical precision during computation, different fields have different requirements. Currently, IEEE's 64-bit floating-point (FP) standard is accurate enough for most scientific applications. However, a higher level of numerical precision is required for the rapidly growing number of important scientific computing applications such as climate modeling, fluid mechanics, etc. This means that these applications require hundreds or more digits to achieve meaningful numerical results. Furthermore, high demand for real-time computing is usually put forward in these scientific computing applications.

In scientific computing, hyperbolic functions such as sinh*x* and cosh*x* find wide applications in engineering fields such as signal processing, power transmission, aerospace, statistics, etc. [1,2]. Hyperbolic functions were typically implemented only in software until recently, wherein their hardware implementation has become important; this is largely due to the performance gains of hardware systems compared with software implementations. Extensive literature exists describing hardware implementation of functions sinh*x* and cosh*x*. Look-up table (LUT) approach, polynomial approximation technique, and coordinate rotation digital computer (CORDIC) algorithm are three typical computational implementation methods of sinh$x$ and cosh$x$ functions [3]. In recent years, the stochastic computing method has also attracted much attention.

LUT method [4,5] is considered simple and swift since it computes functions sinh$x$ and cosh$x$ with stored values in memory blocks via the interpolation method. For this method, the number of entries in memory blocks has to be cautiously chosen, as its computational accuracy and required hardware area cannot compromise each other.

Polynomial approximation technique [6,7] employs Maclaurin series to represent functions sinh$x$ and cosh$x$. Maclaurin series is an infinite sum of derivatives derived from the Taylor series approximation at zero, which demands a mass of multipliers and adders. Although look-up tables can be used to store values of factorials, design area and design memory of this technique still seem inefficient.

As a classic iterative algorithm, the CORDIC algorithm [8] was firstly proposed by Jack E. Volder in 1959. Only shift and addition operations are applied in this algorithm to compute functions sinh$x$ and cosh$x$. It takes much fewer registers and fewer clock cycles to calculate functions sinh$x$ and cosh$x$, making CORDIC the most suited algorithm for the realization of hardware [3,9,10]. However, the CORDIC algorithm calculates vector rotations in one of two modes: rotation and vectoring [11]; as such, it is well characterized as having the latency of a serial multiplication. Moreover, the CORDIC algorithm may not be able to satisfy area requirements in specific applications. Three versions of parallel CORDIC with sign precomputation have been proposed in previous literature—P-CORDIC [12], Flat-CORDIC [13,14], and Para-CORDIC [15]. They have helped in reducing the logic delay and hardware area of the CORDIC prototype.

Gaines firstly introduced stochastic computing [16] for arithmetic digital representation circuits in the late 1960s. Its properties, which are simple arithmetic units [17], fault tolerance, and allowance for high clock rates [18], result in extremely low hardware cost and power consumption, but its disadvantages, including degradation of accuracy and long latency [19], cannot be ignored in some cases. Overall, this technique is likely to find more applications in massively parallel computation driven by the very low-cost hardware [20].

Generally, the LUT method is the fastest to compute hyperbolic functions, but it consumes a large area of silicon. Polynomial approximation achieves excellent approximation with low maximum error in a finite domain of definition but is not fast, as it usually makes use of multipliers in hardware architectures. CORDIC units are commonly used in systems that require a low hardware cost. However, in some applications, even the CORDIC method may not be able to satisfy the area requirements. Stochastic computing attains high frequency and low power consumption at the expense of computing accuracy and long latency.

Among the four above hardware methods, there are no existing architectures reported in the literature to perfectly merge the features of high precision, high accuracy, and low latency, which is an urgent task for some scientific computing applications. In this paper, a novel architecture based on the CORDIC prototype is proposed to fill in this gap. This architecture, called quadruple-step-ahead hyperbolic CORDIC (QH-CORDIC), is demonstrated to be well suited to calculate hyperbolic functions sinh$x$ and cosh$x$ in high-precision FP format with low latency. It is coded in Verilog Hardware Description Language (Verilog HDL) to implement the two functions. A detailed comparison between the proposed architecture and previously published work is also discussed in this paper.

This paper is organized as follows: The principle and range of convergence (ROC) of the basic CORDIC algorithm are reviewed in Section 2. In Section 3, the proposed QH-CORDIC architecture based on basic CORDIC is demonstrated, including its general architecture, ROC, and validity of computing exponential function e^x, which is the main component of hyperbolic functions sinh$x$ and cosh$x$. In Section 4, the overall architecture of the quadruple precision FP hyperbolic functions sinh$x$ and cosh$x$ and the architectures of three internal main modules are detailed. Section 5 compares the FPGA implementation results of our proposed architecture with previously published work and reports the ASIC implementation results of the proposed architecture. Finally, Section 6 concludes this paper.

2. Mathematical Background

2.1. Basic CORDIC Algorithm

Based on shift–addition and vector rotation, the basic CORDIC algorithm is simple and efficient. Recurrent equations of basic CORDIC by theoretical studies [21] are

$$\begin{aligned} X_{i+1} &= X_i - m\,\sigma_i\,2^{-\mathrm{i}}\,Y_i \\ Y_{i+1} &= Y_i + \sigma_i\,2^{-i}\,X_i \\ Z_{i+1} &= Z_i - \sigma_i\,\alpha_i \end{aligned} \tag{1}$$

where $m \in${1,0, −1} according to coordinate type of CORDIC (circular coordinates: m = 1; linear coordinates: m = 0; hyperbolic coordinates: m = −1), α_i represents micro-rotations according to mode type of CORDIC (rotation mode: $\alpha_i = \tan^{-1}2^{-i}$; vectoring mode: $\alpha_i = \tanh^{-1}2^{-i}$), σ_i designates rotation direction according to mode type of CORDIC (rotation mode: $\sigma_i = \mathrm{sign}(Z_i)$; vectoring mode: $\sigma_i = -\,\mathrm{sign}(Y_i)$), and $i = 0, 1, \cdots, n$ for circular coordinates or linear coordinates; i = 1, 2, $\cdots$, n for hyperbolic coordinates. Define scaling factors K and K' for m = 1 and $m = -1$ [22], respectively, as (2) and (3).

$$K = \prod_{i=0}^{n} \cos\alpha_i, m = 1 \tag{2}$$

$$K' = \prod_{i=1}^{n} \cosh\alpha_i, m = -1 \tag{3}$$

2.2. Computation of Functions Sinhx and Coshx with CORDIC

Based on the recurrent Equation (1) and appropriate choice of initial values (X_0, Y_0, and Z_0 for circular coordinates or linear coordinates; X_1, Y_1, and Z_1 for hyperbolic coordinates), a variety of functions can be generated [23]. Table 1 lists common functions that can be calculated with the CORDIC algorithm.

Table 1. Functions with CORDIC algorithm.

m	Mode [1]	Initial Values	Functions [2]	
			X_n	Y_n or Z_n
1	R	$X_0 = 1, Y_0 = 0, Z_0 = \theta$	$\cos\theta$	$Y_n = \sin\theta$
−1	R	$X_1 = 1, Y_1 = 0, Z_1 = \theta$	$\cosh\theta$	$Y_n = \sinh\theta$
−1	R	$X_1 = a, Y_1 = a, Z_1 = \theta$	ae^{θ}	$Y_n = ae^{\theta}$
1	V	$X_0 = 1, Y_0 = a, Z_0 = \pi/2$	$\sqrt{}(a^2 + 1)$	$Z_n = \cot^{-1}a$
−1	V	$X_1 = a, Y_1 = 1, Z_1 = 0$	$\sqrt{}(a^2 - 1)$	$Z_n = \coth^{-1}a$
−1	V	$X_1 = a + 1, Y_1 = a - 1, Z_1 = 0$	$2\sqrt{a}$	$Z_n = 0.5\ln a$
−1	V	$X_1 = a + 1/4, Y_1 = a - 1/4, Z_1 = 0$	$\sqrt{a}$	$Z_n = \ln(a/4)$
−1	V	$X_1 = a + b, Y_1 = a - b, Z_1 = 0$	$2\sqrt{ab}$	$Z_n = 0.5\ln(a/b)$

[1] In column mode, R represents rotation mode, while V represents vectoring mode. [2] Final values X_n and Y_n are obtained after the compensation of the scaling factors K (for m = 1) or K' (for m = −1).

From Table 1, hyperbolic functions sinh$x$ and cosh$x$ can be generated under the circumstance of rotation mode in hyperbolic coordinates. Exponential function e^x, logarithm function ln$x$, and their variant versions can be generated under the circumstance of either rotation mode or vectoring mode in hyperbolic coordinates.

2.3. Range of Convergence for Basic Hyperbolic CORDIC Algorithm

For basic CORDIC in hyperbolic coordinates, convergence conditions are expressed as in (4) [24].

$$\begin{aligned}&\left|\tanh^{-1}\tfrac{Y_1}{X_1}\right| \leq \alpha_N + \sum_{n=1}^{N-1}\alpha_n\\&\left|\tanh^{-1}\tfrac{Y_1}{X_1}\right| < 1.7433\\&\left|\tfrac{Y_1}{X_1}\right| < 0.94608\end{aligned} \tag{4}$$

where Y_1 and X_1 are initial values of CORDIC. It can be inferred that a useful domain in radian for basic CORDIC in hyperbolic coordinates must locate in $(-1.7433, 1.7433)$. Such ROC may not satisfy the across-all-range requirement of FP input values.

In addition, when i is 4, 13, 40, 121, $\cdots$, $(3^{u+2} - 1)/2$, $\cdots$ where integer u starts from 0, repeated iterations are necessary in order to ensure the convergence of basic CORDIC in hyperbolic coordinates. Thus, actual iteration sequence of CORDIC is i = 1, 2, 3, 4, 4, 5, $\cdots$, 12, 13, 13, $\cdots$.

2.4. Another Computation of Functions Sinhx and Coshx

Restricted to limited ROC, rough implementation of functions sinh$x$ and cosh$x$ with basic CORDIC seems inappropriate. To realize the across-all-range computation of functions sinh$x$ and cosh$x$, this paper proposes another methodology.

Hyperbolic functions sinh$x$ and cosh$x$ can be defined in terms of exponential function e^x,

$$\text{sinh}x = \frac{e^x - e^{-x}}{2} \tag{5}$$

$$\cosh x = \frac{e^x + e^{-x}}{2} \tag{6}$$

where $e^{-x} = 1/e^x$. It can be seen from (5) and (6) that computation of sinh$x$ and cosh$x$ consists of function e^x, division (to compute e^{-x}), addition/subtraction operation, and shift operation (right shift). When it comes to the computation of function e^x, several studies [25,26] address this problem using an approximation method. In addition to the approximation approach, iterative methods are also widely exploited. Iterative methods include digit-recurrence method [27–29] and hyperbolic CORDIC [30,31].

To enhance computational precision of function e^x as high as possible with less complex hardware, hyperbolic CORDIC was chosen for this study. However, hyperbolic CORDIC brings about high-precision computation at the cost of high latency, which cannot be tolerated by modern hardware. To eliminate the high-latency flaw from the hyperbolic CORDIC algorithm, this paper proposes a novel QH-CORDIC architecture.

3. Quadruple-Step-Ahead Hyperbolic CORDIC Architecture

3.1. Improvement of Basic CORDIC Algorithm

Inspired by the double-step CORDIC algorithm [32], this paper proposes a QH-CORDIC architecture, which combines four sequential iterations into one single iteration step. Recurrent equations of the proposed QH-CORDIC are shown in (7)–(9).

$$\begin{aligned}X_{i+4} = X_i * \{&1 + 2^{-(4i+6)} * [\sigma_{i+3}\,\sigma_{i+2}\,\sigma_{i+1}\,\sigma_i\,]\\&+ 2^{-(2i+5)} * [16\,\sigma_{i+1}\,\sigma_i + 8\,\sigma_{i+2}\,\sigma_i + 4\,\sigma_{i+2}\,\sigma_{i+1} + 4\,\sigma_{i+3}\,\sigma_i + 2\,\sigma_{i+3}\,\sigma_{i+1} + \sigma_{i+3}\,\sigma_{i+2}]\}\\&+ Y_i * \{\,2^{-(i+3)} * [8\,\sigma_i + 4\,\sigma_{i+1} + 2\sigma_{i+2} + \sigma_{i+3}]\\&+ 2^{-(3i+6)} * [8\,\sigma_{i+2}\,\sigma_{i+1}\,\sigma_i + 4\,\sigma_{i+3}\sigma_{i+1}\sigma_i + 2\,\sigma_{i+3}\,\sigma_{i+2}\,\sigma_i + \sigma_{i+3}\,\sigma_{i+2}\,\sigma_{i+1}]\}\end{aligned} \tag{7}$$

$$\begin{aligned}Y_{i+4} = Y_i * \{&1 + 2^{-(4i+6)} * [\sigma_{i+3}\,\sigma_{i+2}\,\sigma_{i+1}\,\sigma_i\,]\\&+ 2^{-(2i+5)} * [16\,\sigma_{i+1}\,\sigma_i + 8\,\sigma_{i+2}\,\sigma_i + 4\,\sigma_{i+2}\,\sigma_{i+1} + 4\,\sigma_{i+3}\,\sigma_i + 2\,\sigma_{i+3}\,\sigma_{i+1} + \sigma_{i+3}\,\sigma_{i+2}]\}\\&+ X_i * \{\,2^{-(i+3)} * [8\,\sigma_i + 4\,\sigma_{i+1} + 2\sigma_{i+2} + \sigma_{i+3}]\\&+ 2^{-(3i+6)} * [8\,\sigma_{i+2}\,\sigma_{i+1}\,\sigma_i + 4\,\sigma_{i+3}\,\sigma_{i+1}\,\sigma_i + 2\,\sigma_{i+3}\,\sigma_{i+2}\,\sigma_i + \sigma_{i+3}\,\sigma_{i+2}\,\sigma_{i+1}]\}\end{aligned} \tag{8}$$

$$Z_{i+4} = Z_i - \sigma_{i+3}\,\alpha_{i+3} - \sigma_{i+2}\,\alpha_{i+2} - \sigma_{i+1}\,\alpha_{i+1} - \sigma_i\,\alpha_i \tag{9}$$

where σ_i, σ_{i+1}, σ_{i+2}, σ_{i+3} designate rotation directions of the i-th, (i+1)-th, (i+2)-th, (i+3)-th rotations, $\alpha_i = \tanh^{-1}(2^{-i})$, $\alpha_{i+1} = \tanh^{-1}[2^{-(i+1)}]$, $\alpha_{i+2} = \tanh^{-1}[2^{-(i+2)}]$, $\alpha_{i+3} = \tanh^{-1}[2^{-(i+3)}]$, and $i = 1, 2, \cdots, n$.

The necklace of the QH-CORDIC lies in the simultaneous prediction of σ_i for four sequential iterations. The value of σ_i is either -1 (rotating in a clockwise direction) or 1 (rotating in an anticlockwise direction). A combination of $\{\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}\}$ corresponding to four sequential iterations has 16 possible cases as for their values, ranging from $\{-1, -1, -1, -1\}$ to $\{1, 1, 1, 1\}$.

Substitute the 16 possible cases of $\{\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}\}$ into (8) and obtain the 16 simplified expressions for Y_{i+4}. Table 2 details the corresponding recurrent equations of Y_{i+4} when $\{\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}\}$ ranges from $\{-1, -1, -1, -1\}$ to $\{1, 1, 1, 1\}$. Since recurrent equations of X_{i+4} are almost the same as those of Y_{i+4}, table listing recurrent equations of X_{i+4} is omitted.

Table 2. Recurrent equations of Y_{i+4} in QH-CORDIC.

Case	σ_i	σ_{i+1}	σ_{i+2}	σ_{i+3}	$\mathbf{Y_{i+4}}$
1	−1	−1	−1	−1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} + 35 \times 2^{-(2n+5)}] + X_i \times [-15 \times 2^{-(n+3)} - 15 \times 2^{-(3n+6)}]$
2	−1	−1	−1	1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} + 21 \times 2^{-(2n+5)}] + X_i \times [-13 \times 2^{-(n+3)} - 2^{-(3n+6)}]$
3	−1	−1	1	−1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} + 9 \times 2^{-(2n+5)}] + X_i \times [-11 \times 2^{-(n+3)} + 7 \times 2^{-(3n+6)}]$
4	−1	1	−1	−1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} - 9 \times 2^{-(2n+5)}] + X_i \times [-7 \times 2^{-(n+3)} + 11 \times 2^{-(3n+6)}]$
5	1	−1	−1	−1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} - 21 \times 2^{-(2n+5)}] + X_i \times [2^{-(n+3)} + 13 \times 2^{-(3n+6)}]$
6	−1	−1	1	1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 2^{-(2n+5)}] + X_i \times [-9 \times 2^{-(n+3)} + 9 \times 2^{-(3n+6)}]$
7	−1	1	−1	1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 15 \times 2^{-(2n+5)}] + X_i \times [-5 \times 2^{-(n+3)} + 5 \times 2^{-(3n+6)}]$
8	−1	1	1	−1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 19 \times 2^{-(2n+5)}] + X_i \times [-3 \times 2^{-(n+3)} - 3 \times 2^{-(3n+6)}]$
9	1	−1	−1	1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 19 \times 2^{-(2n+5)}] + X_i \times [3 \times 2^{-(n+3)} + 3 \times 2^{-(3n+6)}]$
10	1	−1	1	−1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 15 \times 2^{-(2n+5)}] + X_i \times [5 \times 2^{-(n+3)} - 5 \times 2^{-(3n+6)}]$
11	1	1	−1	−1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} - 2^{-(2n+5)}] + X_i \times [9 \times 2^{-(n+3)} - 9 \times 2^{-(3n+6)}]$
12	−1	1	1	1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} - 21 \times 2^{-(2n+5)}] + X_i \times [-2^{-(n+3)} - 13 \times 2^{-(3n+6)}]$
13	1	−1	1	1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} - 9 \times 2^{-(2n+5)}] + X_i \times [7 \times 2^{-(n+3)} - 11 \times 2^{-(3n+6)}]$
14	1	1	−1	1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} + 9 \times 2^{-(2n+5)}] + X_i \times [11 \times 2^{-(n+3)} - 7 \times 2^{-(3n+6)}]$
15	1	1	1	−1	$Y_{i+4} = Y_i \times [1 - 2^{-(4n+6)} + 21 \times 2^{-(2n+5)}] + X_i \times [13 \times 2^{-(n+3)} + 2^{-(3n+6)}]$
16	1	1	1	1	$Y_{i+4} = Y_i \times [1 + 2^{-(4n+6)} + 35 \times 2^{-(2n+5)}] + X_i \times [15 \times 2^{-(n+3)} + 15 \times 2^{-(3n+6)}]$

Table 3 lists recurrent equations of Z_{i+4} when $\{\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}\}$ ranges from $\{-1, -1, -1, -1\}$ to $\{1, 1, 1, 1\}$.

Table 3. Recurrent equations of Z_{i+4} in QH-CORDIC.

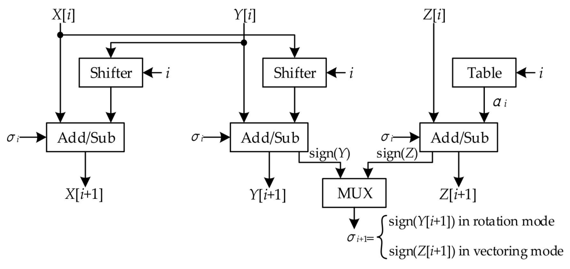
Case	σ_i	σ_{i+1}	σ_{i+2}	σ_{i+3}	$\mathbf{Z_{i+4}}$
1	−1	−1	−1	−1	$Z_{i+4} = Z_i + \alpha_i + \alpha_{i+1} + \alpha_{i+2} + \alpha_{i+3}$
2	−1	−1	−1	1	$Z_{i+4} = Z_i + \alpha_i + \alpha_{i+1} + \alpha_{i+2} - \alpha_{i+3}$
3	−1	−1	1	−1	$Z_{i+4} = Z_i + \alpha_i + \alpha_{i+1} - \alpha_{i+2} + \alpha_{i+3}$
4	−1	1	−1	−1	$Z_{i+4} = Z_i + \alpha_i - \alpha_{i+1} + \alpha_{i+2} + \alpha_{i+3}$
5	1	−1	−1	−1	$Z_{i+4} = Z_i - \alpha_i + \alpha_{i+1} + \alpha_{i+2} + \alpha_{i+3}$
6	−1	−1	1	1	$Z_{i+4} = Z_i + \alpha_i + \alpha_{i+1} - \alpha_{i+2} - \alpha_{i+3}$

Table 3. *Cont.*

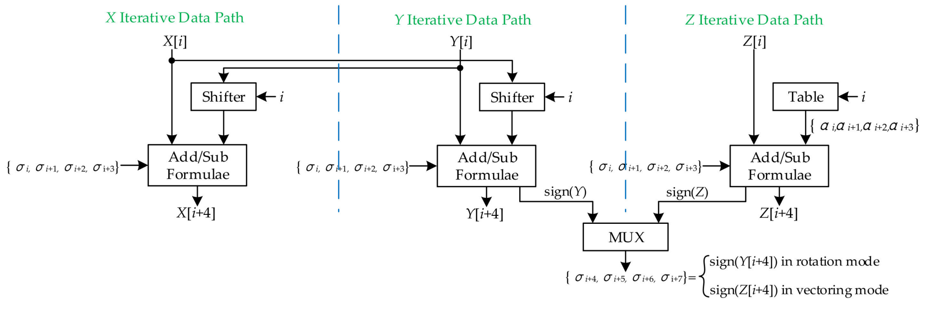
Case	σ_i	σ_{i+1}	σ_{i+2}	σ_{i+3}	Z_{i+4}
7	−1	1	−1	1	$Z_{i+4} = Z_i + \alpha_i - \alpha_{i+1} + \alpha_{i+2} - \alpha_{i+3}$
8	−1	1	1	−1	$Z_{i+4} = Z_i + \alpha_i - \alpha_{i+1} - \alpha_{i+2} + \alpha_{i+3}$
9	1	−1	−1	1	$Z_{i+4} = Z_i - \alpha_i + \alpha_{i+1} + \alpha_{i+2} - \alpha_{i+3}$
10	1	−1	1	−1	$Z_{i+4}= Z_i- \alpha_i + \alpha_{i+1} - \alpha_{i+2} + \alpha_{i+3}$
11	1	1	−1	−1	$Z_{i+4} = Z_i - \alpha_i - \alpha_{i+1} + \alpha_{i+2} + \alpha_{i+3}$
12	−1	1	1	1	$Z_{i+4} = Z_i + \alpha_i - \alpha_{i+1} - \alpha_{i+2} - \alpha_{i+3}$
13	1	−1	1	1	$Z_{i+4} = Z_i - \alpha_i + \alpha_{i+1} - \alpha_{i+2} - \alpha_{i+3}$
14	1	1	−1	1	$Z_{i+4} = Z_i - \alpha_i - \alpha_{i+1} + \alpha_{i+2} - \alpha_{i+3}$
15	1	1	1	−1	$Z_{i+4} = Z_i - \alpha_i - \alpha_{i+1} - \alpha_{i+2} + \alpha_{i+3}$
16	1	1	1	1	$Z_{i+4} = Z_i - \alpha_i - \alpha_{i+1} - \alpha_{i+2} - \alpha_{i+3}$

3.2. General Architecture of QH-CORDIC

The hardware architecture of basic CORDIC and QH-CORDIC is presented in Figure 1a,b, respectively. Figure 1a bears a close resemblance to Figure 1b because they both have three major data paths (X data path, Y data path, and Z data path). Their differences mainly lie in the signal(s) that determines the rotation direction of the next iteration.



(**a**)



(**b**)

Figure 1. (**a**) Hardware architecture of basic CORDIC; (**b**) hardware architecture of QH-CORDIC.

The hardware iteration of QH-CORDIC in two modes, vectoring mode and rotating mode, is briefly demonstrated in Figure 2a,b, respectively. As explained in Section 2.3, in order to ensure ROC of hyperbolic CORDIC, when $i = 5, 13, 41, 121, \cdots, (3^{u+2} - 1)/2, \cdots$ where u starts from 0, repeated iterations are needed. Therefore, except $X/Y/Z$ iterative data path that performs iterative formulae of $X_{i+4}/Y_{i+4}/Z_{i+4}$, $X/Y/Z$ repetitive data path when $i = 5, 13, 41, \cdots$ is also listed.

It should be noted that QH-CORDIC in rotating mode can be used to compute exponential function e^x. According to (5) and (6), this paper only employs QH-CORDIC in rotating mode to implement hyperbolic functions sinh$x$ and cosh$x$. As for QH-CORDIC in vectoring mode, it can be used to compute the logarithmic function ln$x$.

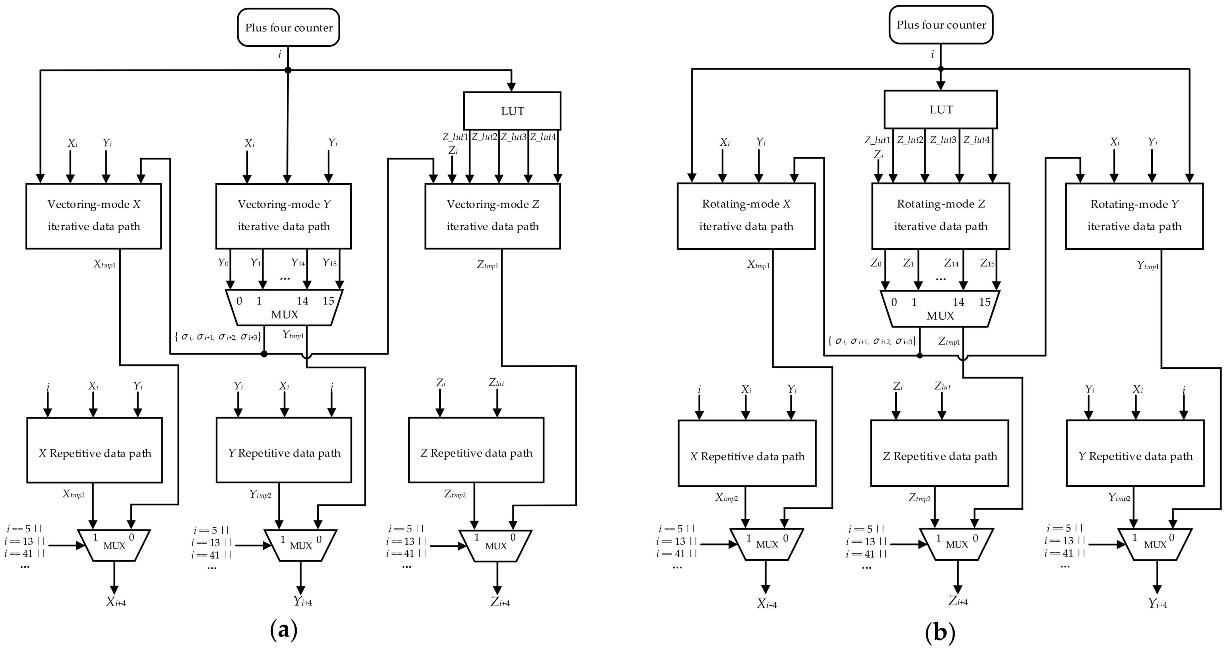


Figure 2. (**a**) QH-CORDIC in vectoring mode; (**b**) QH-CORDIC in rotating mode.

3.3. ROC of QH-CORDIC for Exponential Function

The computation of exponential function e^v is performed through (10).

$$e^v = \frac{e^v + e^{-v}}{2} + \frac{e^v - e^{-v}}{2} = \cosh v + \sinh v. \tag{10}$$

Initial conditions and terminated statuses for QH-CORDIC-based computation of e^v are listed in (11) and (12), respectively.

$$\begin{cases} x_1 = K_\infty \\ y_1 = 0 \\ z_1 = v \\ K_\infty = \prod_{i=1}^{\infty} \frac{1}{\sqrt{1-2^{-2i}}} \end{cases} \tag{11}$$

$$\begin{cases} x_\infty = \cosh v \\ y_\infty = \sinh v \\ z_\infty = 0 \end{cases} \tag{12}$$

According to (4), ROC of input v for function e^v is $(-1.7433, 1.7433)$.

3.4. Validity of Computing Exponential Function with QH-CORDIC

To study the validity of computation of exponential function e^x in FP format using QH-CORDIC, suppose input FP number x as $(-1)^S \times M \times 2^E$ where S is the sign of x, E is the exponent of x after correcting bias, and M is mantissa of x after complementing the implicit bit. The assumption is made that the output of function e^x is $A \times 2^B$ where $0.5 < A < 1$ and B is an integer.

Suppose S = 0 first. The discussion of sign S = 1 will be involved later. From

$$e^{M\times 2^E} = A \times 2^B, \tag{13}$$

we can obtain

$$0.5 < \frac{e^{M\times 2^E}}{2^B} < 1 \tag{14}$$

$$2^{B-1} < e^{M\times 2^E} < 2^B. \tag{15}$$

Performing the two-based-log operation of both sides to (15), we obtain

$$B-1 < M \times \frac{2^E}{\ln 2} < B. \tag{16}$$

Since B is an integer, and the value of B can be attained with (16).

In order to ensure the value of A, suppose $2^B = e^Z$. Then,

$$Z = B\ln 2 \tag{17}$$

Substitute (17) into (13) and yield

$$A = e^{M\times 2^E - B\ln 2} \tag{18}$$

By (16), the value of B can be computed. A is in the range of (0.5,1). According to the graph of exponential function e^x, $M \times 2^E - B \times \ln 2$ must locate in the ROC of CORDIC, i.e., $(-1.7433, 1.7433)$. Therefore, the value of A can be attained by (18).

When $S = 1$, $e^x = -A \times 2^B$. Following the abovementioned steps, we can obtain

$$B-1 < -M \times \frac{2^E}{\ln 2} < B. \tag{19}$$

$$A = e^{-M\times 2^E - B\ln 2} \tag{20}$$

Similarly, for the condition where $S = 1$, the value of B can be computed by (19) and A is also in the range of (0.5,1). According to the graph of exponential function e^x, $-M \times 2^E - B \times \ln 2$ must locate in the ROC of CORDIC. Therefore, the value of A can be attained by (20).

Thus, the validity of computing exponential function e^x with CORDIC is checked.

3.5. Simplified Computing of B in Formula (16) or (19)

Since the proposed QH-CORDIC architecture is mainly for quadruple precision FP hyperbolic functions sinh$x$ and cosh$x$, it is necessary to reduce the area of circuit design in the context of high-precision FP input. In Section 3.4, if input FP number x is a quadruple precision FP number, M will be a 113-bit fixed-point number. The difficulty of computing B in Formula (16) or (19) lies in the calculation of $M \times 2^{E/}$ln2 where both M and 1/ln2 are 113-bit fixed-point numbers. Multiplying M with 1/ln2 straightforward is theoretically feasible. However, in practice, such operation will take an extremely large circuit design area.

It can be observed that in the context of the above situation, B will be a 15-bit fixed-point number, which means that the complex multiplication of M and 1/ln2 can be simplified. The challenge is to reduce effective digits of M and 1/ln2 in the actual calculation. Denote M and 1/ln2 as (21) and (22),

$$\begin{aligned} M &= \overbrace{1.x_{-1}x_{-2}\cdots x_{-(p-2)}x_{-(p-1)}}^{p}x_{-p}x_{-(p+1)}\cdots x_{-111}x_{-112} \\ &= \overbrace{1.x_{-1}x_{-2}\cdots x_{-(p-2)}x_{-(p-1)}}^{p}00\cdots 00 + \overbrace{0.00\cdots 00}^{p}x_{-p}x_{-(p+1)}\cdots x_{-111}x_{-112} \\ &= P + \Delta P \end{aligned} \tag{21}$$

$$\begin{aligned}
1/\ln 2 &= \overbrace{1.101x_{-4}x_{-5}\cdots x_{-(q-2)}x_{-(q-1)}}^{p}x_{-q}x_{-(q+1)}\cdots x_{-111}x_{-112} \\
&= \overbrace{1.101x_{-4}x_{-5}\cdots x_{-(q-2)}x_{-(q-1)}}^{p}00\cdots 00+\overbrace{0.00\cdots 00}^{p}x_{-q}x_{-(q+1)}\cdots x_{-111}x_{-112} \\
&= Q+\Delta Q
\end{aligned} \tag{22}$$

where p and q are two positive integers. P is defined as the high-order p bits of M extended with 0s to obtain a 113-bit number, while Q is defined as the high-order q bits of 1/ln2 extended with 0s to obtain a 113-bit number. Let $\Delta P = M - P$ and $\Delta Q = 1/\ln 2 - Q$. Hence, $P < 2$ and $Q < 2$; $|\Delta P| < 2^{-p}$, and $|\Delta Q| < 2^{-q}$.

According to (21) and (22), B must be $x_1x_{0.}\ x_{-1}x_{-2}\cdots x_{-13}$ where x_1x_0 may be 01, 10, or 11. Finding appropriate values for integers p and q to ensure $|P \times Q - M \times 1/\ln 2| < 2^{-13}$ is the key for simplified computing of B. Since $M = P + \Delta P$ and $1/\ln 2 = Q + \Delta Q$, then

$$|P \times Q - M \times 1/\ln 2| = |P\Delta Q + Q\Delta P + \Delta P\Delta Q|. \tag{23}$$

As $P < 2$, $Q < 2$, $|\Delta P| < 2^{-p}$, $|\Delta Q| < 2^{-q}$, $p \gg 1$ and $q \gg 1$, (23) can be approximated as (24),

$$|P\Delta Q + Q\Delta P + \Delta P\Delta Q| \approx |P\Delta Q + Q\Delta P| \tag{24}$$

and

$$|P\Delta Q + Q\Delta P| < 2^{-p+1} + 2^{-q+1} \le 2\cdot\sqrt{2^{-p-q+2}} = 2^{\frac{-p-q}{2}+2}. \tag{25}$$

Thus, it becomes

$$2^{\frac{-p-q}{2}+2} < 2^{-13}. \tag{26}$$

We can obtain

$$p + q > 30. \tag{27}$$

Let $p = q = 16$, and (23) becomes

$$\begin{aligned}
&|P\Delta Q + Q\Delta P + \Delta P\Delta Q| \\
&< 2\cdot 2^{-16} + 2\cdot 2^{-16} + 2^{-16-16} \\
&< 2^{-14} + 2^{-32} \\
&< 2^{-14} + 2^{-14} \\
&< 2^{-13}
\end{aligned} \tag{28}$$

From the check of (28), we can derive that setting p and q to 16 can ensure $|P \times Q - M \times 1/\ln 2| < 2^{-13}$. That is to say, effective digits of M and 1/ln2 only need to be 16 rather than 113. This helps to simplify the calculation of B in formula (16) or (19), which is also reflected in the architecture of state PRE_B in Section 4.

4. Hardware Implementation of Hyperbolic Functions Sinh*x* and Cosh*x* with QH-CORDIC

The proposed QH-CORDIC architecture can apply to both fixed-point and FP operations. Meanwhile, the QH-CORDIC architecture is appropriate for configurable precision. In this paper, based on the QH-CORDIC architecture, a quadruple precision FP hardware implementation of hyperbolic functions sinh*x* and cosh*x* is presented.

The overall architecture of the quadruple precision FP hyperbolic functions sinh*x* and cosh*x* is illustrated in Figure 3. The proposed architecture is divided into three parts: Module Pre_deal, Module Cordic_core, and Module exp_divide_sinh_cosh. Inputs are an FP number, *input_num*, and two signals—*clk* and *rst_n*. Outputs are *sinh_result*, *cosh_result*, *sinh_cosh_done*, and *sinh_cosh_exception*, which are a 128-bit calculated FP result of function sinh*x*, a 128-bit calculated FP result of function cosh*x*, a completion signal, and an exception signal, respectively.

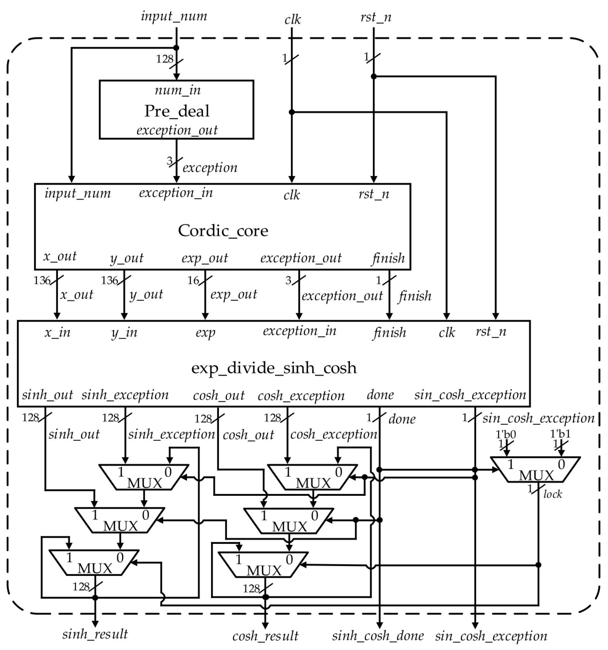


Figure 3. Overall architecture of quadruple precision functions sinh$x$ and cosh$x$ in FP format.

Module Pre_deal is to judge whether exception situations exit after breaking down the FP input *input_num* into three portions: 1-bit sign (*sign*), 15-bit exponent (e), and 112-bit mantissa (m). The output of Module Pre_deal is a 3-bit signal *exception*. There are five possible values of *exception*: 3′b000 (no exception), 3′b001 (*input_num* is not a number), 3′b010 (*input_num* is negative infinite), 3′b011 (*input_num* is positive infinite), and 3′b100 (*input_num* is small enough to be seen as zero when 15-bit exponent of *input_num* is smaller than 15′h3f8c).

After Module Pre_deal, Module Cordic_core computes function e^{input_num} with the proposed QH-CORDIC algorithm under the circumstance of no exception. If any exception exists, signal *exception_out* will be outputted and completion signal *finish* turns to be 1. Module Cordic_core is mainly composed of a finite state machine (FSM), which has six states in total. The state transition diagram of the FSM is shown in Figure 4.

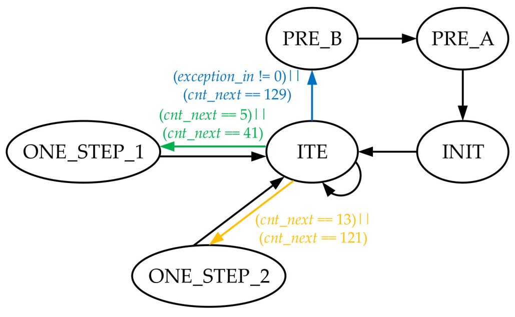


Figure 4. State transition diagram of finite state machine.

Among these six states, state PRE_B and state PRE_A are to calculate the value of B and A, respectively, in (16) and (18). The architectures of state PRE_B and state PRE_A are shown in Figure 5a,b, respectively.

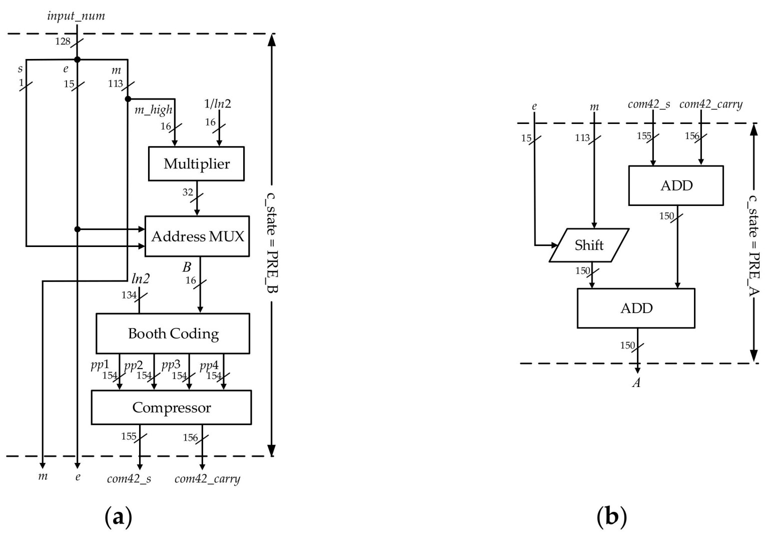


Figure 5. (**a**) Architecture of state PRE_B; (**b**) architecture of state PRE_A.

State INIT is to perform the initialization process of exponential function e^{input_num}. Figure 6 shows the data path of state INIT. In Figure 6, input A is the output of state PRE_A. K_inv is a constant, and its value is expressed in (21).

$$K\_inv = 1/K_\infty = 1/(\prod_{i=1}^{\infty} \frac{1}{\sqrt{1-2^{-2i}}}) \tag{29}$$

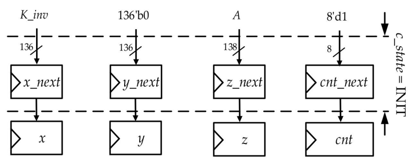


Figure 6. Architecture of state INIT.

State ITE is to perform QH-CORDIC computation of exponential function e^{input_num}. Figure 7 shows the data path of state ITE. The red box in Figure 7 corresponds to the rotating-mode $X/Y/Z$ iterative data path in Figure 2b. The three important modules x_pre, y_pre, and z_pre, respectively, perform iterative data path of X_{i+4} in (7), Y_{i+4} in (8), and Z_{i+4} in (9). In addition, the signal in the register *cnt_next* and signal *exception_in* determine the next state of state ITE together.

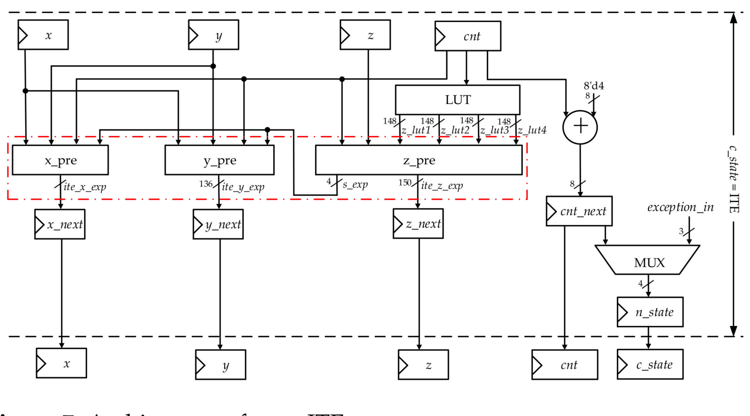


Figure 7. Architecture of state ITE.

Figure 8a,b demonstrates architectures of state ONE_STEP_1 and state ONE_STEP_2, respectively. Two blue boxes in Figure 8a,b make up $X/Y/Z$ repetitive iterative data path and in Figure 2b for exponential function jointly.

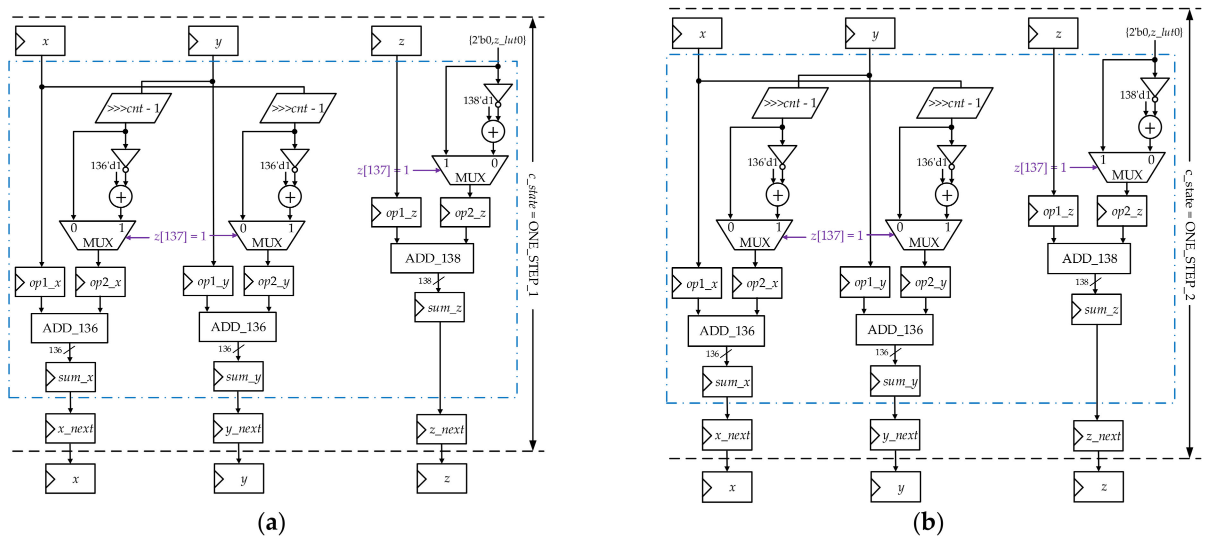


Figure 8. (**a**) Architecture of state ONE_STEP_1; (**b**) architecture of state ONE_STEP_2.

After Module Cordic_core, output signals *x_out*, *y_out*, *exp_out*, *exception_out* and *finish* are generated. Receiving the above-mentioned five signals and two control signals *clk* and *rst_n*, Module exp_divide_sinh_cosh firstly calculates exponential function e^{-input_num} with *x_in* and *y_in*. As Figure 9 demonstrates, $e^{input\_num} = x\_in + y\_in$. Furthermore, $e^{-input\_num} = 1/e^{input\_num}$. The computation of e^{-input_num} is implemented through the Predict-Correct algorithm in [33] with $p = 113$, $q = 113$, $m = 11$, $n = 3$ and $t = 3$. After obtaining e^{-input_num} and e^{input_num}, the two desired hyperbolic functions sinh(*input_num*) and cosh(*input_num*) can be attained with (21) and (22).

$$\sinh(input\_num) = \frac{e^{input\_num} - e^{-input\_num}}{2} \tag{30}$$

$$\cosh(input\_num) = \frac{e^{input\_num} + e^{-input\_num}}{2} \tag{31}$$

It can be inferred from (21) and (22) that computation of sinh(*input_num*) and cosh (*input_num*) is made up with a 128-bit FP addition/subtraction operation and a right-shift operation, which is also demonstrated in Figure 9.

There also exists exception handling in Module exp_divide_sinh_cosh. If no exception conditions exist, Module exp_divide_sinh_cosh outputs the result of hyperbolic functions sinh(*input_num*) and cosh(*input_num*), respectively, *sinh_out* and *cosh_out*. Otherwise, Module exp_divide_sinh_cosh outputs an exception flag signal *sinh_cosh_exception* and the corresponding exceptional result of sinh(*input_num*) and cosh(*input_num*), respectively, *sinh_out* and *cosh_out*.

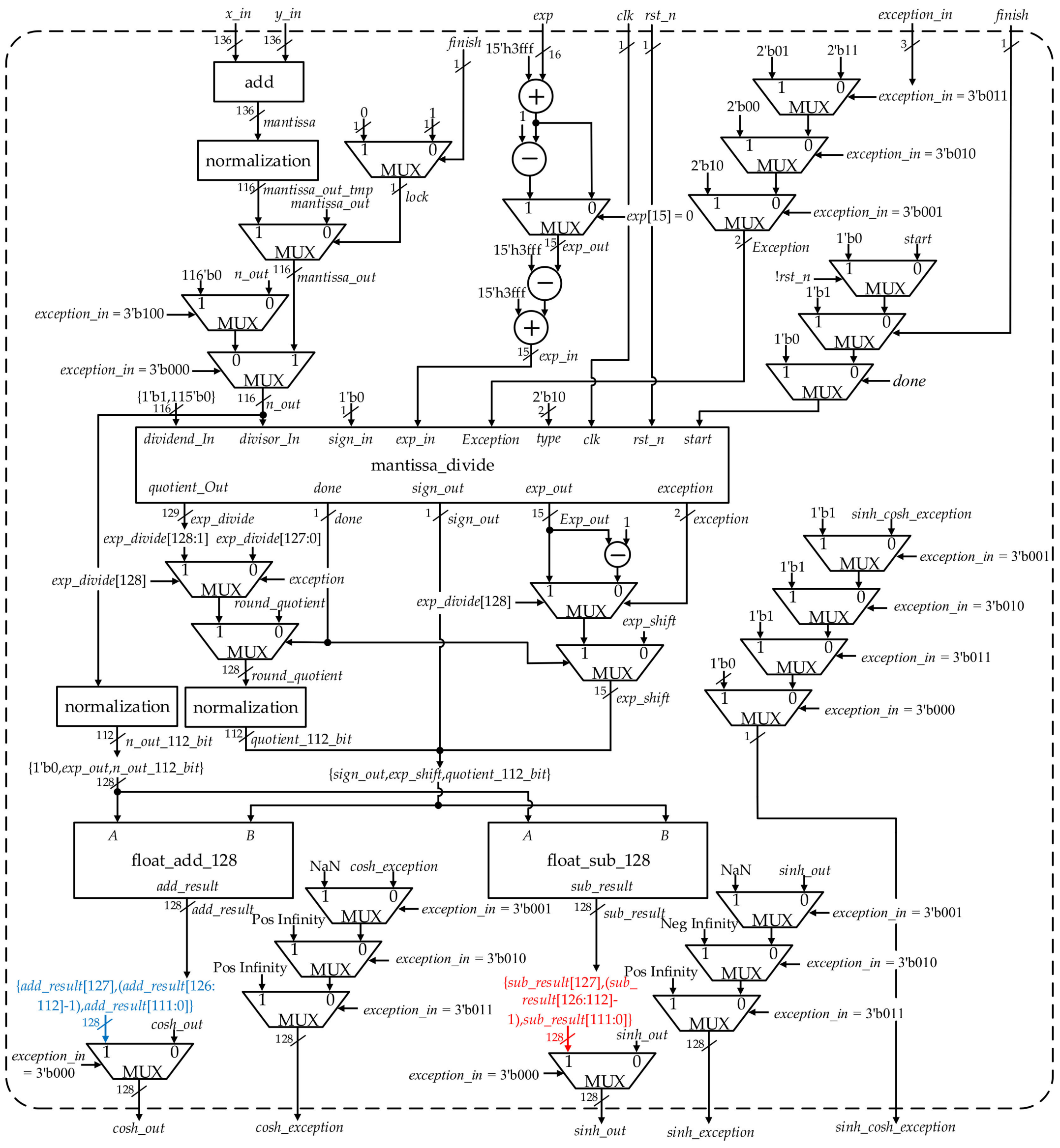


Figure 9. General architecture of Module exp_divide_sinh_cosh.

5. Implementation and Comparisons

The proposed architecture was coded in Verilog Hardware Description Language. Verification of hardware implementation of the two functions sinh$x$ and cosh$x$ is presented in Section 5.1. After that, it was synthesized in the Xilinx ISE Design Suite and mapped to an FPGA device (xc7vx485). Comparisons in terms of timing analysis and device utilization are discussed in Section 5.2. The proposed architecture was also synthesized with TSMC 65 nm standard cell library, using Synopsys Design Compiler. The ASIC implementation details are shown in Section 5.3. Section 5.4 compares the proposed architecture with the LUT method, stochastic computing, and other CORDIC algorithms to show its characteristics of high accuracy, low error, and vast ROC when performing high-precision computing.

5.1. Functional Verification

The functional verification of the proposed architecture was carried out using 1-million random test cases for normal, sub-normal, and other exceptional input numbers with IEEE's 128-bit FP mode. This paper compares the hardware simulation results of the proposed architecture with software results using the bigfloat package. The bigfloat package is a

Python wrapper for the GNU MPFR library for arbitrary-precision FP reliable arithmetic. It provides precise control over precisions and gives correctly rounded reproducible platform-independent results.

In the case of the 1-million random 128-bit FP tests, the statistical correct rate of the proposed architecture is 99.6%, compared with bigfloat data results using Python. Among the 0.4% not-matched 128-bit FP tests, the proposed architecture produces a maximum of 2-ULP (unit at last place) precision loss.

5.2. FPGA Implementation Analysis

Timing analysis and device utilization are discussed in this subsection. This paper mainly implements a 128-bit (i.e., quadruple precision) FP hyperbolic functions architecture, where the number of internal iterations is up to 128.

The methods developed by [3,32] and the proposed architecture in this study are three variants of the CORDIC algorithm. For an equal comparison, set N to 128 in [3,32]. Meanwhile, the study by [3] only focuses on hyperbolic functions with fixed-point inputs that are convergent to ROC of basic CORDIC. Hence, for equal comparison, only Module Cordic_core (without states PRE_B and PRE_A) of the proposed architecture is synthesized.

The designed hardware was simulated with a clock of period 10 ns. Table 4 provides the timing analysis and device utilization of [3] (N = 128), [32] (N = 128), and the proposed architecture. According to Table 4, the method by [3] takes three times more clock cycles than the proposed architecture, while the method by [32] takes one time more clock cycles than the proposed architecture. It can be inferred that for [3], the number of clock cycles absolutely depends on the value of N; for [32] and the proposed architecture, the number of clock cycles equals $N/2$ and $N/4$, respectively. The reason why clock cycles of the proposed architecture are so few lies in the fact that the proposed architecture performs four-bits computation every iteration.

Table 4. Timing waveform and device utilization comparison.

	Paper [3]	**Paper [32]**	**Proposed**
Clock cycles	128 (100%)	64 (50%)	32 (25%)
Time taken (ns)	1280 (100%)	640 (50%)	320 (25%)
Slice	1106 (100%)	7624 (689.3%)	9430 (852.6%)
Slice flip flops	337 (100%)	462 (137.1%)	512 (151.9%)
Four-input LUTs	3403 (100%)	24168 (710.2%)	29172 (857.2%)
Bonded IOBs	403 (100%)	425 (105.5%)	403(100%)

Table 4 also shows that the number of device resources consumed by [3] (N = 128), [32] (N = 128), and the proposed architecture (only Module Cordic_core). According to Table 4, the number of bonded IOBs consumed by the proposed architecture is the same as or even smaller than that consumed by [3] or [32]. The number of slice flip flops consumed by the proposed architecture is half-time more than or slightly larger than that consumed by [3] or [32], respectively.

However, the number of slices and four-input LUTs consumed by the proposed architecture is about 7.5 times more than those consumed by [3]. The reason why the proposed architecture consumes so many slices and LUTs lies in the calculation of X, Y, and Z's 16 predictive formulae. Considering the amount of calculation magnified by a factor of 16 in theory, the practical utilization of the device resources of the proposed architecture seems to be acceptable.

5.3. ASIC Implementation Performance

The proposed architecture is synthesized with the best achievable timing constraints, with a constraint of the max-area set to zero and a global operating voltage of 0.9 V.

Section 5.3 compares the performance of ASIC implementation of the proposed architecture with [3] (N = 128) and [32] (N = 128). This paper retrieves studies [3,32] after enlarging the ROC of [3,32] to $(-2^{15}, 2^{15})$ and reducing their error to be below 2^{-113}.

Table 5 lists nine parameters of ASIC implementation of the three variants of the CORDIC algorithm. Since the clock period is set to be 3.3 ns for [3,32] and the proposed architecture, the clock frequency of ASIC implementation is 300 MHz. Keeping the same clock frequency, the latency parameter of [3,32] and the proposed architecture is 137, 73, and 41, respectively, for 128-bit FP input numbers. The downward trend of parameter latency from [3], to [32], to the proposed architecture, is steeper, showing that the proposed architecture can dramatically cut down on latency. Therefore, it is with the total time parameter.

Table 5. Comparison of ASIC implementation details @ TSMC 65 nm.

	Paper [3]	**Paper [32]**	**Proposed**
Area (μm^2)	451782 (100%)	909540 (201.3%)	1321500 (292.5%)
Power (mW)	4.11 (100%)	8.12 (197.6%)	12.60 (306.6%)
Latency (cycle)	137 (100%)	73 (53.3%)	41 (29.9%)
Period (ns)		3.3	
Total time (ns) [1]	452.1 (100%)	240.9 (53.3%)	135.3 (29.9%)
ATP (mm^2·ns) [2]	204.25 (100%)	219.11 (107.3%)	178.79 (87.5%)
Total energy (fJ) [3]	1858.13 (100%)	1956.11 (105.3%)	1580.04 (85%)
Energy efficiency (fJ/bit) [4]	14.52 (100%)	15.28 (105.2%)	12.34 (84.9%)
Area efficiency (bit/(mm^2·ns)) [5]	0.63 (100%)	0.58 (92.1%)	0.71 (112.7%)

[1] Total time = latency × period. [2] ATP = area × total time. [3] Total energy = power × total time. [4] Energy efficiency = total energy/efficient bits where efficient bits equal to N = 128 in Table 5. [5] Area efficiency = efficient bits/(area × total time) where efficient bits equal to N = 128 in Table 5.

However, the latency and total time of the proposed architecture are reduced at the expense of area and power. In comparison to [3], the area and power of the proposed architecture are approximately three times those of [3]. In comparison to [32], the area and power of the proposed architecture are approximately 1.5 times those of [32].

ATP and total energy parameters are usually used to evaluate ASIC performance more properly and roundly. The smaller ATP and total energy are, the better the ASIC design is. In Table 5, ATP and total energy of the proposed architecture are smaller than those of [3,32]. This can be explained as the advantage of the proposed architecture is low latency at the cost of area and power. To solve the problem of the expanded area and power, the proposed architecture employs module re-using, clock gating, and other techniques. Meanwhile, low latency leads to less computing time, which eventually makes the proposed architecture superior to the first two CORDIC variants in terms of ATP and total energy.

According to the definitions of energy efficiency and area efficiency, the smaller the energy efficiency is and the larger the area efficiency is, the better the ASIC design is. As for the energy efficiency and area efficiency of the two architectures, the proposed architecture also achieves better performance. Due to low latency, less energy is consumed, and more area is utilized per bit in the computing of hyperbolic functions with 128-bit FP inputs using the proposed architecture. Specifically, the proposed architecture has 15.1% energy efficiency and 19.2% energy efficiency overhead over [3,32], respectively. The proposed architecture has 12.7% and 22.4% more area efficiency over, respectively, [3,32].

To summarize, the proposed architecture does not supersede [3] or [32] in terms of parameter area and power. However, it outperforms the other two variants of the CORDIC algorithm in terms of ATP, energy efficiency, and area efficiency parameters since the proposed QH-CORDIC algorithm brings about a low-latency feature.

5.4. Related Works and Comparisons

The proposed architecture also focuses on high-precision computing of the two functions sinh$x$ and cosh$x$ by enhancing accuracy, lowering function error, and enlarging ROC. Table 6 demonstrates the comparisons of the LUT method, stochastic computing, and CORDIC algorithms. It should be noted that the data of the CORDIC algorithm is adopted from original studies [3,9,32], without retrieval.

LUT method is a way to compute hyperbolic functions sinh$x$ and cosh$x$. The study by [5] computes trigonometric and hyperbolic functions using look-up tables whose size is 77 bit $\times$ 14 to achieve the accuracy of 4 bits. In order to improve accuracy, the volume of look-up tables used in this method will increase exponentially; that is, high-precision function values will run out of a huge amount of LUTs. Meanwhile, a larger look-up table brings about the lower searching speed.

Another way to compute hyperbolic functions is stochastic computing, as performed in studies by [20,34]. Stochastic computing applies stochastic bitstreams to compute, and its main features are having a low cost and low power [35]. The accuracy of stochastic computing is related to the length of stochastic numbers. According to [36], the length of stochastic numbers l is related to the precision i, and the number of independent variables n in the calculated function, i.e., $l = 2^{i-n}$. High-precision function values require a larger length of stochastic numbers. For 128-bit FP inputs, the accuracy of 113 for the mantissa part should be guaranteed. In this case, $l = 2^{113-n}$. In practice, l cannot be too large, so n needs to be appropriate. This means that for high-precision computation, a large number of stochastic data will be generated, leading to tremendous latency, area, and power.

From Table 6, the function error of the proposed architecture is less than $2^{-113,}$ and ROC is expanded to $(-2^{15}, 2^{15})$. It is a dramatic improvement, compared with the other structures.

Table 6. Comparisons of LUT, stochastic computing, and CORDIC on high-precision computing.

	LUT Method	**Stochastic Computing**		**CORDIC Algorithms**			
	Paper [5]	**Paper [34]**	**Paper [20]**	**Paper [9]**	**Paper [3]**	**Paper [32]**	**Proposed**
Accuracy (bit)	4	10	7	8	4	10	128
Function Error	-	-	MAE [1] = 0.0043	MRE [2] = 0.45	MAE = 0.043	-	$<2^{-113}$
LUT volume [3]	77 $\times$ 14	No LUTs	20 $\times$ 8	Entry depth = 8	Entry depth = 4	Entry depth = 10	136 $\times$ 128
ROC [4]	[0,10080]	[0,1]	[0,1]	$[-1,1]$	$[-1.207,1.207]$	$[-1.743,1.743]$	$(-2^{15},2^{15})$

[1] MAE stands for mean absolute error. [2] MRE stands for mean relative error. [3] LUT volume = data width (bit) $\times$ entry depth. [4] ROC stands for range of convergence.

To summarize, both the LUT method and stochastic computing are disadvantageous when performing high-precision computation. Among the above four CORDIC algorithms, metrics accuracy (or function error) and ROC are both considered in the proposed architecture.

6. Conclusions

A new method and hardware architecture were proposed to compute hyperbolic functions sinh$x$ and cosh$x$ based on the QH-CORDIC algorithm in this study.

Restricted to limited ROC of basic CORDIC algorithm, hardware implementation of functions sinh$x$ and cosh$x$ with all-floating-point-domain inputs on basis of basic CORDIC seems infeasible. The proposed QH-CORDIC algorithm is based on a basic hyperbolic CORDIC algorithm. Explaining the principle and structure of the QH-CORIDC, this study discussed ROC and validity of the QH-CORDIC when computing exponential function e^x with all-floating-point-domain inputs since function e^x mainly consists of functions sinh$x$ and cosh$x$.

As for the circuit design of functions sinh$x$ and cosh$x$ with QH-CORDIC, the entire logic path was tuned to perform a low-latency computation. The proposed circuit architecture has 75% clock cycles overhead over [3] and 50% clock cycles overhead over [32]. From the trade-off aspect of performance–power–area, in Section 5.3, the proposed architecture was proved to be superior to [3,32] in terms of metrics of total time, ATP, total energy, energy efficiency, and area efficiency. Section 5.4 showed that it is much more favorable for the proposed architecture to perform high-precision computing of hyperbolic functions.

In addition, the proposed architecture can be configured for single-precision, double-precision, quadruple-precision, or other user-defined precisions. Meanwhile, the proposed architecture can also be adapted in computations of hyperbolic functions sinh$x$ and cosh$x$ with fixed-point input numbers after simple adjustment. Moreover, other common hyperbolic functions such as tanh$x$, arcsinh$x$, arccosh$x$, and arctanh$x$ can also be computed using the QH-CORDIC algorithm.

Author Contributions: Conceptualization, M.W. and M.L.; methodology, W.F.; software, W.F.; validation, W.F. and J.X.; writing—original draft preparation, W.F.; writing—review and editing, W.F. and X.L.; funding acquisition, M.W. and M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Science Foundation of Guangdong Province, China (Grant No. 2020B1515120004), Shenzhen Science and Technology Plan-Basic Research (Grant No. JCY20180503182125190), Shenzhen Science and Technology Plan-Basic Research (Grant No. JCYJ20180507182241622), and Scientific research project in school-level (SZIIT2019KJ026).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Muller, J.M. *Elementary Functions: Algorithms and Implementations*, 2nd ed.; Birkhauser: Basel, Switzerland, 2006.
2. Parhami, B. *Computer Arithmetic: Algorithms and Hardware Designs*; Oxford University Press: Oxford, UK, 1999.
3. Saha, A.; Kumar, K.G.; Ghosh, A.; Naskar, M.K. Area efficient architecture of Hyperbolic functions for high frequency applications. In Proceedings of the 2017 International Conference on Circuits, Controls, and Communications (CCUBE), Bangalore, India, 15–16 December 2017; pp. 139–142.
4. Tang, P.T.P. Table-lookup Algorithms for Elementary Functions and Their Error Analysis. In Proceedings of the 10th IEEE Symposium on Computer Arithmetic, Grenoble, France, 26–28 June 1991; pp. 232–236.
5. Saint-Geniès, H.d.L.; Defour, D.; Revy, G. Exact Lookup Tables for the Evaluation of Trigonometric and Hyperbolic Functions. *IEEE Trans. Comput.* **2017**, *66*, 2058–2071. [CrossRef]
6. Koren, I.; Zinaty, O. Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations. *IEEE Trans. Comput.* **1990**, *39*, 1030–1037. [CrossRef]
7. Schulte, M.J.; Swartzlander, E.E. Hardware Design for Exactly Rounded Elementary Functions. *IEEE Trans. Comput.* **1994**, *43*, 964–973. [CrossRef]
8. Volder, J.E. The CORDIC Trigonometric Computing Technique. *IEEE Trans. Electron. Comput.* **1959**, *EC-8*, 330–334. [CrossRef]
9. Boudabous, A.; Ghozzi, F.; Kharrat, M.W.; Masmoudi, N. Implementation of hyperbolic functions using CORDIC algorithm. In Proceedings of the 16th International Conference on Microelectronics, Tunis, Tunisia, 6–8 December 2004; pp. 738–741.
10. Vazquez, Á.; Villalba, J.; Antelo, E. Computation of Decimal Transcendental Functions Using the CORDIC Algorithm. In Proceedings of the 2009 19th IEEE Symposium on Computer Arithmetic, Portland, OR, USA, 8–10 June 2009; pp. 179–186.
11. Ross, D.-M.; Miller, S.; Mihai, S. Exploration of sign precomputation-based CORDIC in reconfigurable systems. In Proceedings of the 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), Pacific Grove, CA, USA, 6–9 November 2011; pp. 2186–2191.
12. Kuhlmann, M.; Parhi, K.K. P-CORDIC: A Precomputation Based Rotation CORDIC Algorithm. *EURASIP J. Adv. Signal Process.* **2002**, *2002*, 936–943. [CrossRef]
13. Srikanthan, T.; Gisuthan, B. A novel technique for eliminating iterative based computation of polarity of micro-rotations in CORDIC based sine-cosine generators. *Microprocess. Microsyst.* **2002**, *26*, 243–252. [CrossRef]
14. Gisuthan, B.; Srikanthan, T. FLAT CORDIC: A Unified Architecture for High-Speed Generation of Trigonometric and Hyperbolic Functions. In Proceedings of the 43rd Midwest Symposium on Circuits and Systems (MWSCAS 2000), Lansing, MI, USA, 8–11 August 2000; pp. 1414–1417.
15. Juang, T.-B.; Hsiao, S.-F.; Tsai, M.-Y. Para-CORDIC: Parallel CORDIC Rotation Algorithm. *Trans. Circuits Syst.—I Regul. Pap.* **2004**, *51*, 1515–1524. [CrossRef]
16. Gaines, B.R. Stochastic Computing. In Proceedings of the American Federation of Information Processing Societies Spring Joint Computer Conf, Atlantic City, NJ, USA, 18–20 April 1967.

17. Parhi, K.; Liu, Y. Computing Arithmetic Functions Using Stochastic Logic by Series Expansion. *IEEE Trans. Emerg. Top. Comput.* **2016**, *7*, 1–13. [CrossRef]
18. Card, B.D.; Brown, H.C. Stochastic Neural Computation I: Computational Elements. *IEEE Trans. Comput.* **2001**, *50*, 891–905.
19. Liu, Y.; Parhi, K.K. Computing hyperbolic tangent and sigmoid functions using stochastic logic. In Proceedings of the 2016 50th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 6–9 November 2016; pp. 1580–1585.
20. Luong, T.; Nguyen, V.; Nguyen, A.; Popovici, E. Efficient Architectures and Implementation of Arithmetic Functions Approximation Based Stochastic Computing. In Proceedings of the 2019 IEEE 30th International Conference on Application-Specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 15–17 July 2019; pp. 281–287.
21. Walther, J.S. A Unified Algorithm for Elementary Functions. In Proceedings of the AFIPS Spring Joint Computer Conference, New York, NY, USA, 18–20 May 1971; pp. 379–385.
22. Kharrat, M.W.; Loulou, M.; Masmoudi, N.; Kamoun, L. A New Method to Implement CORDIC Algorithm. In Proceedings of the International Conference on Electronics, Circuits and Systems ICECS, Malta, Malta, 2–5 September 2001.
23. Eklund, N. CORDIC: Elementary Function Computation Using Recursive Sequences. *Issue Coll. Math. J.* **2001**, *32*, 330–333. [CrossRef]
24. Llamocca-Obregón, R.D.; Agurto-Ríos, P.C. A fixed-point implementation of the expanded hyperbolic CORDIC algorithm. *Lat. Am. Appl. Res.* **2007**, *37*, 83–91.
25. De Dinechin, F.; Pasca, B. Floating-point exponential function-ns for DSP-enabled FPGAs. In Proceedings of the IEEE International Conference on Field-Program Technology, Beijing, China, 8–10 December 2010; pp. 110–117.
26. Langhammer, M.; Pasca, B. Single precision logarithm and exponential architectures for hard floating-point enabled FPGAs. *IEEE Trans. Comput.* **2017**, *66*, 2031–2043. [CrossRef]
27. Pineiro, J.-A.; Ercegovac, M.D.; Bruguera, J.D. Algorithm and architecture for logarithm, exponential, and powering computation. *IEEE Trans. Comput.* **2004**, *53*, 1085–1096. [CrossRef]
28. Chen, D.; Han, L.; Ko, S.B. Decimal floating-point antilogarithmic converter based on selection by rounding: Algorithm and architecture. *IET Comput. Digit. Technol.* **2012**, *6*, 277–289. [CrossRef]
29. Chen, D.; Han, L.; Choi, Y.; Ko, S.-B. Improved decimal floating-point logarithmic converter based on selection by rounding. *IEEE Trans. Comput.* **2012**, *61*, 607–621. [CrossRef]
30. Meher, P.K.; Valls, J.; Juang, T.-B.; Sridharan, K.; Maharatna, K. 50 years of CORDIC: Algorithms, architectures, and applications. *IEEE Trans. Circuits Syst. I Reg. Pap.* **2009**, *56*, 1893–1907. [CrossRef]
31. Wang, Y.; Dinavahi, V. Real-time digital multi-function protection system on reconfigurable hardware. *IET Gen. Transm. Distrib.* **2016**, *10*, 2295–2305. [CrossRef]
32. Phatak, D.S. Double step branching CORDIC: A new algorithm for fast sine and cosine generation. *IEEE Trans. Comput.* **1998**, *47*, 587–602. [CrossRef]
33. Xia, J.; Fu, W.; Liu, M.; Wang, M. Low-Latency Bit-Accurate Architecture for Configurable Precision Floating-Point Division. *Appl. Sci.* **2021**, *11*, 4988. [CrossRef]
34. Huai, L.; Li, P.; Sobelman, G.E.; Lilja, D.J. Stochastic computing implementation of trigonometric and hyperbolic functions. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 553–556.
35. Hayes, J.P. Introduction to stochastic computing and its challenges. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–3.
36. Chen, T.; Ting, P.; Hayes, J.P. Achieving progressive precision in stochastic computing. In Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Montreal, QC, Canada, 14–16 November 2017; pp. 1320–1324.