

Article

A Low-Latency CORDIC Algorithm Based on Pre-Rotation and Its Application on Computation of Arctangent Function

Kun Li , Hongji Fang , Zhenguo Ma , Feng Yu , Bo Zhang  and Qianjian Xing *

College of Biomedical Engineering and Instrument Science, Zhejiang University, Hangzhou 310027, China; 12315003@zju.edu.cn (K.L.); 11815025@zju.edu.cn (H.F.); 850501@zju.edu.cn (Z.M.); osfengyu@zju.edu.cn (F.Y.); 11915004@zju.edu.cn (B.Z.)

* Correspondence: xingqianjian@zju.edu.cn

Abstract: This paper presents a low-latency coordinate rotation digital computer (CORDIC) algorithm to accelerate the computation of arctangent functions, and it describes the corresponding iterative and pipelined architecture of this novel algorithm. As compared to the existing methods based on CORDIC, the proposed method can effectively reduce the number of iterations by dedicated pre-rotation and comparison processes. Moreover, the proposed CORDIC algorithm supports all vectors with arbitrary angles while maintaining convergence. By error analysis, the proposed algorithm can achieve the same accuracy as the conventional CORDIC algorithm during floating-point arctangent function computation and reduce the number of iterations by approximately 50%. This paper presents two new architectures—the iterative architecture, which can be more resource efficient, and the pipelined architecture, which can achieve a throughput rate of one datum per clock. Finally, the experimental comparison results indicate that the proposed method outperforms extant methods as it exhibits low latency, requires fewer resources to compute the arctangent function for floating-point inputs, and necessitates no digital signal processing (DSP) and memory for fixed-point inputs.

Keywords: CORDIC; arctangent; floating point; pre-rotation; low latency; iterative architecture; pipeline architecture



Citation: Li, K.; Fang, H.; Ma, Z.; Yu, F.; Zhang, B.; Xing, Q. A Low-Latency CORDIC Algorithm Based on Pre-Rotation and Its Application on Computation of Arctangent Function. *Electronics* **2024**, *13*, 2338. <https://doi.org/10.3390/electronics13122338>

Academic Editor: Alexander Barkalov

Received: 13 May 2024

Revised: 12 June 2024

Accepted: 12 June 2024

Published: 14 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The inverse tangent, commonly termed the arctangent or atan, is a fundamental mathematical function with diverse applications in many fields, ranging from engineering and physics to computer science and signal processing. In early hardware algorithms, the primary methods for computing trigonometric functions such as arctangents involved techniques based on look-up tables (LUTs) [1] and the approximate transformation of trigonometric functions [2,3]. With the continuous development of information technology, the demand for computing arctangent functions suitable for high-precision or floating-point numbers concurrently increases. However, these methods often consume significant storage and computational resources or may lack sufficient precision when computing high-precision numbers. To address the aforementioned challenges, the coordinate rotation digital computer (CORDIC) algorithm has emerged as a solution.

CORDIC was first described by Volder in 1959 for the computation of trigonometric functions, multiplication, and division [4] and later generalized [5] to calculate a broader range of equations involving logarithms, exponentials, and square roots, among others. Owing to its hardware implementation being highly efficient and not resource-intensive (one iteration can be performed with basic shift-add operations), CORDIC has undergone rapid development [6,7] and extensive application [8–11].

When calculating trigonometric functions, the conventional CORDIC algorithm employs an iterative method to incrementally approach the theoretical values. Owing to the sequential nature of the iterations, each iteration must wait for the previous one to complete,

and the number of iterations depends on the data precision required. Consequently, latency has become a major drawback of the CORDIC algorithm when performing calculations on high-bit-width data.

Throughout the developmental trajectory of CORDIC, many researchers have invested substantial efforts to enhance the algorithm. These efforts have focused on improving computational accuracy, expanding the range of angles covered, reducing the number of iterations, and minimizing resource consumption. Hu and Naganathan proposed an angle recoding (AR) method based on the greedy algorithm [12], which achieves optimality at each iteration by encoding the theoretical results as a set of linear combinations of steps, which was then developed as a branch of CORDIC [13]. However, the necessity of encoding in advance restricts the applicability to when the result is known, such as in discrete Fourier transform. In addition, the method of reducing the total number of iterations by adding specific comparisons and calculations into a single iteration, such as by using the radix-4 [14] and higher radix [15] CORDIC algorithms, the double-step branching algorithm [16], the hybrid CORDIC [17], CORDIC II [18], and binary CORDIC algorithms [19], can effectively reduce latency. Moreover, in the case of fixed-point inputs, the specific calculation process can be simplified by analyzing the binary bits. However, the specific computation in these algorithms consumes copious resources when calculating a single floating-point number. The TCORDIC algorithm, combining a low-latency CORDIC algorithm with the Taylor expansion, has been proposed, utilizing sign prediction, compressed iteration, and parallel iteration techniques to reduce latency and errors at the boundary [20]. Nevertheless, it simultaneously incurs a substantial resource overhead and has not been applied to the computation of arctangents. In [21], the decision-based CORDIC also reduced latency by skipping some redundant iterations, but it is only used for fixed-point numbers, and its method of skipping iterations cannot achieve optimality.

To realize floating-point calculations of the arctangent function with fewer resources and low latency, the pre-rotation-based CORDIC (PR-CORDIC) algorithm proposed herein applies a more appropriate rotation angle at each iteration by pre-rotating the input vector and comparing the results. Moreover, the PR-CORDIC algorithm selects the micro-rotation angle from a limited number of pre-rotation angles rather than finding an optimal solution for all angles to avoid overusing the resources. The PR-CORDIC algorithm applies the comparison result of the pre-rotations to select the better angle and determines the angle for the subsequent pre-rotation to maximize the utilization of the results. Therefore, it utilizes fewer resources more efficiently to select a better rotation angle at each iteration. Another benefit of the flexible selection of the rotation angle is the expansion of the input angle range.

The remainder of this paper is organized as follows. Section 2 presents an overview of the CORDIC algorithm. The PR-CORDIC algorithm is detailed in Section 3. Section 4 primarily discusses the performance of the proposed CORDIC algorithm regarding error rates. Section 5 presents selected corresponding architectures for the proposed algorithm. In Section 6, the proposed approach is compared with other methods, and its implementation is demonstrated regarding area and performance. Section 7 concludes the paper.

2. Overview of CORDIC

2.1. CORDIC Algorithm

The CORDIC algorithm comprises three categories: the circular, linear, and hyperbolic coordinate systems, as well as the rotation and vector modes. To compute the arctangent function, we focus on the vector mode of the CORDIC operation by using a circular trajectory. On a computer, when a 2D vector V_0 must be rotated to another vector V_n by an angle α , the process is that the coordinate matrix of V_0 is multiplied by another matrix comprising $\sin \alpha$ and $\cos \alpha$. To simplify the hardware implementation for computing this process, the circular CORDIC algorithm decomposes the rotation into a sequence of elementary rotations through predefined angles. Then, the angle between V_0 and V_n can be obtained on the basis of the angle sequence.

In vector mode, as shown in Figure 1, the initial vector $V_0 = (x_0, y_0)$ is rotated to the final vector $V_n = (x_n, y_n)$; then, $y_n = 0$ and the phase is zero. Once the rotation is completed, the phase information of the initial vector can be obtained.

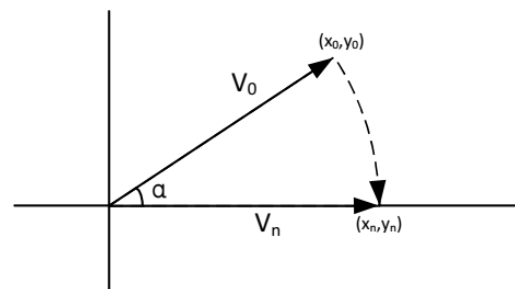


Figure 1. CORDIC rotation in vector mode.

According to [22], the generalized circular CORDIC algorithm is specified by the following iterative equation:

$$\begin{cases} x_{i+1} = x_i - \sigma_i \cdot 2^{-i} \cdot y_i, \\ y_{i+1} = y_i + \sigma_i \cdot 2^{-i} \cdot x_i, \\ \omega_{i+1} = \omega_i - \sigma_i \cdot \alpha_i, \end{cases} \quad (1)$$

where $\alpha_i = \tan^{-1}(2^{-i})$ and $\sigma_i = -\text{sign}(y_i)$ in vector mode and $\sigma_i = \text{sign}(\omega_i)$ in rotation mode.

The outputs following m iterations of the CORDIC algorithm are displayed in Table 1. K_c is the scale factor, which is defined as

$$K_c = \prod_{k=0}^m \sqrt{1 + 2^{-2 \cdot k}}. \quad (2)$$

Table 1 shows that circular CORDIC in vector mode (VC-CORDIC) can evaluate an arctangent when the initial phase $\omega_0 = 0$.

Table 1. Generalized circular CORDIC algorithm.

	Rotation	Vector
x_m	$K_c(x_0 \cos \omega_0 - y_0 \sin \omega_0)$	$K_c \sqrt{x_0^2 + y_0^2}$
y_m	$K_c(y_0 \cos \omega_0 + x_0 \sin \omega_0)$	0
ω_m	0	$\omega_0 + \tan^{-1}(y_0/x_0)$

2.2. Pipeline Architecture for Arctangent Computation

Figure 2 illustrates the conventional architecture used to compute the arctangent function, where n determines the precision of the result. Theoretically, after n iterations, the final result has an accuracy of 2^{-n+1} and a maximum value of

$$\tan^{-1}(2^0) + \tan^{-1}(2^{-1}) + \dots + \tan^{-1}(2^{-n}), \quad (3)$$

which is 99.88° in terms of angle. In other words, when the angle of the input vector (x_0, y_0) falls outside the range of $[-99.88, 99.88]$ x_0 and y_0 must first be preprocessed (by rotating by 90 or 180 degrees, which are trivial transformations), allowing for the architecture to adapt to random inputs in all quadrants.

During the iteration of the CORDIC algorithm, although the modulus of the vector scales gradually, it does not affect the correct operation of the iteration, since the direction of the micro-rotation is derived from the sign bit of y_i . Meanwhile, as shown in Table 1, the

result of the arctangent function can be derived directly from the final ω_n , independently of the scale-factor K_c . In summary, the scale factor can be ignored throughout the computation of the arctangent.

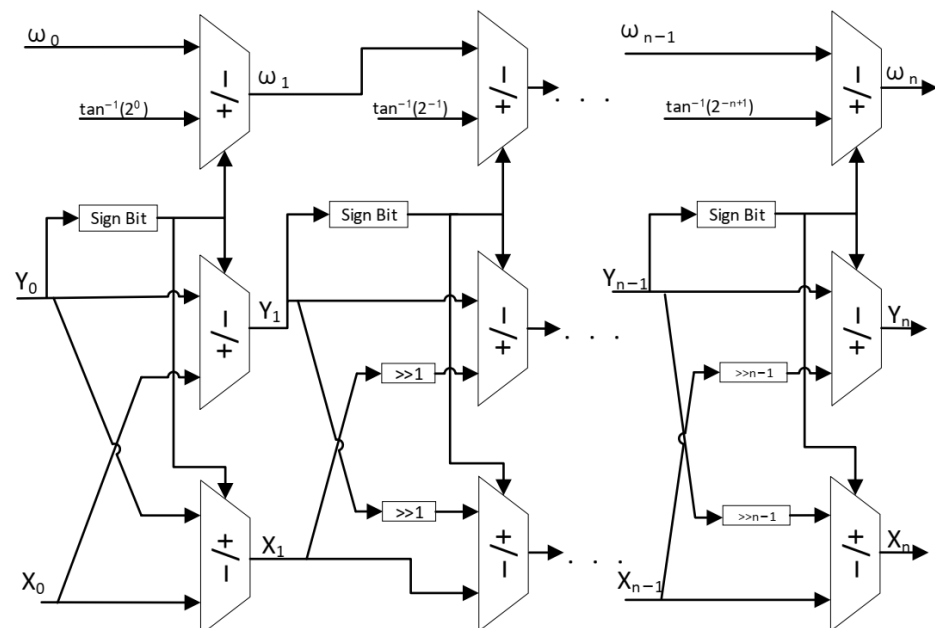


Figure 2. The conventional CORDIC pipelined architecture in vector mode. It consists of n stages, with each stage requiring three adders, and the rotation angle completed in each stage is fixed.

3. Proposed Algorithm

To accelerate the iteration of the CORDIC algorithm, the proposed algorithm skips some unnecessary iterations by pre-rotation. In Figure 3, when the initial vector is V_0 , it arrives at positions V_1 and V_2 after rotations, which, in radians, are $\tan^{-1}(2^0)$ and $\tan^{-1}(2^{-1})$, respectively. Evidently, the value of the vertical coordinate of V_2 is closer to zero; this implies that the rotation from V_0 to V_1 can potentially be skipped. Therefore, the proposed algorithm performs pre-rotation before each iteration and compares the obtained results to select the most suitable rotation angle.

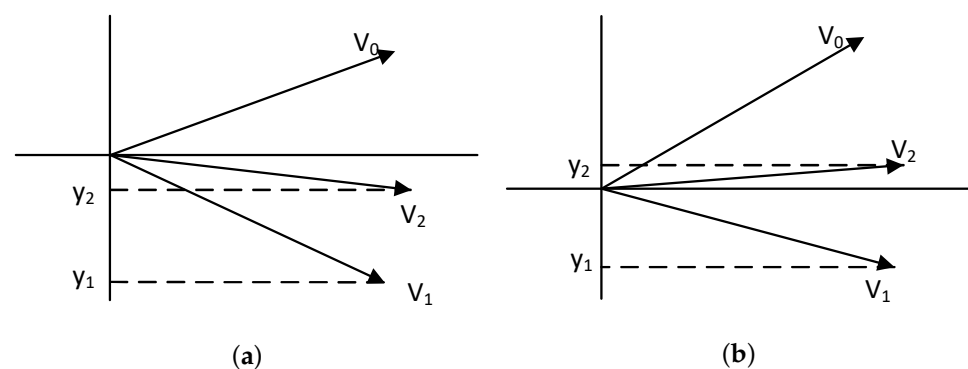


Figure 3. First rotation of the initial vector: (a) Angle of the initial vector is 20.00° . (b) Angle of the initial vector is 30.00° .

3.1. Single Iteration Process

The iterative equation for the proposed PR-CORDIC algorithm in vector mode is presented in (4):

$$\begin{cases} x_n &= x_c + \sigma_c \cdot y_c \cdot 2^{-k}, \\ y_n &= y_c - \sigma_c \cdot x_c \cdot 2^{-k}, \\ \omega_n &= \omega_c + \sigma_c \cdot \alpha_k, \end{cases} \quad (4)$$

where the subscripts c and n represent the current and next values, respectively, k is an integer determined by the pre-rotations, and the value of σ_c is either $\text{sign}(y_c)$ or 0 as determined by the pre-rotations. The r pre-rotations in one iteration are described in (5):

$$\begin{cases} y_1 &= y_c - \sigma_{y_c} \cdot x_c \cdot 2^{-i}, \\ y_2 &= y_c - \sigma_{y_c} \cdot x_c \cdot 2^{-i-1}, \\ &\vdots \\ y_r &= y_c - \sigma_{y_c} \cdot x_c \cdot 2^{-i-r+1}, \end{cases} \quad (5)$$

where i is determined by the previous iteration and $\sigma_{y_c} = \text{sign}(y_c)$.

In contrast with the previous algorithm, wherein the value of k is mechanically related to the number of iteration layers, in the PR-CORDIC algorithm it is related to the pre-rotation result. The pre-rotation and comparison processes are described in the pseudocode of Algorithm 1. The number of pre-rotations is set to $r = 2$ for a simplified description:

Algorithm 1 Pre-rotation and Comparison

Input: i_c, x_c, y_c, ω_c

Output: i_n, x_n, y_n, ω_n

Function: $[i_n, x_n, y_n, \omega_n] = \text{Compare}(i_c, x_c, y_c, \omega_c)$

```

1:  $\sigma_{y_c} = \text{sign}(y_c)$ .
2:  $y_1 = y_c + \sigma_{y_c} \cdot x_c \cdot 2^{-i_c}$ .
3:  $y_2 = y_c + \sigma_{y_c} \cdot x_c \cdot 2^{-i_c-1}$ .
4:  $\sigma_{y_1} = \text{sign}(y_1), \sigma_{y_2} = \text{sign}(y_2)$ .
5: if  $\sigma_{y_c} = \sigma_{y_1}$  then
6:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c}$ 
7:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c})$  ▷ Case 1
8:    $y_n = y_1, i_n = i_c$ 
9: else if  $\sigma_{y_c} \neq \sigma_{y_1}, \sigma_{y_c} = \sigma_{y_2}, |y_1| < |y_2|$  then
10:   $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c}$ 
11:   $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c})$  ▷ Case 2
12:   $y_n = y_1, i_n = i_c + 2$ 
13: else if  $\sigma_{y_c} \neq \sigma_{y_1}, \sigma_{y_c} = \sigma_{y_2}, |y_1| \geq |y_2|$  then
14:   $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-1}$ 
15:   $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-1})$  ▷ Case 3
16:   $y_n = y_2, i_n = i_c + 2$ 
17: else if  $\sigma_{y_c} \neq \sigma_{y_2}, |y_2| < |y_c|$  then
18:   $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-1}$ 
19:   $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-1})$  ▷ Case 4
20:   $y_n = y_2, i_n = i_c + 2$ 
21: else if  $\sigma_{y_c} \neq \sigma_{y_2}, |y_2| \geq |y_c|$  then
22:   $x_n = x_c, \omega_n = \omega_c$  ▷ Case 5
23:   $y_n = y_c, i_n = i_c + 2$ 
24: end if
25: return  $i_n, x_n, y_n, \omega_n$ 

```

For better illustration, we provide examples for each of the five cases in Figure 4. When y_c is negative, all cases are the same as when y_c is positive.

In the proposed algorithm, the entire calculation process can be divided into two major stages. In the first stage, the rotation angle is fixed, and the number of iterations varies based on the angle of the input vector. In this stage, by using the initial angle for multiple iterations, we can rotate the input vector more greatly and expand the algorithm's vector angle range. In Algorithm 1, the comparison results in this stage always correspond to Case 1. In the second stage, the rotation angle is not fixed, and we select the most suitable rotation angle by comparing the comparison results of r pre-rotations.

These two stages do not repeat, meaning there will be no transition from the second stage back to the first stage in the entire calculation process. This will be explained in the convergence proof section later.

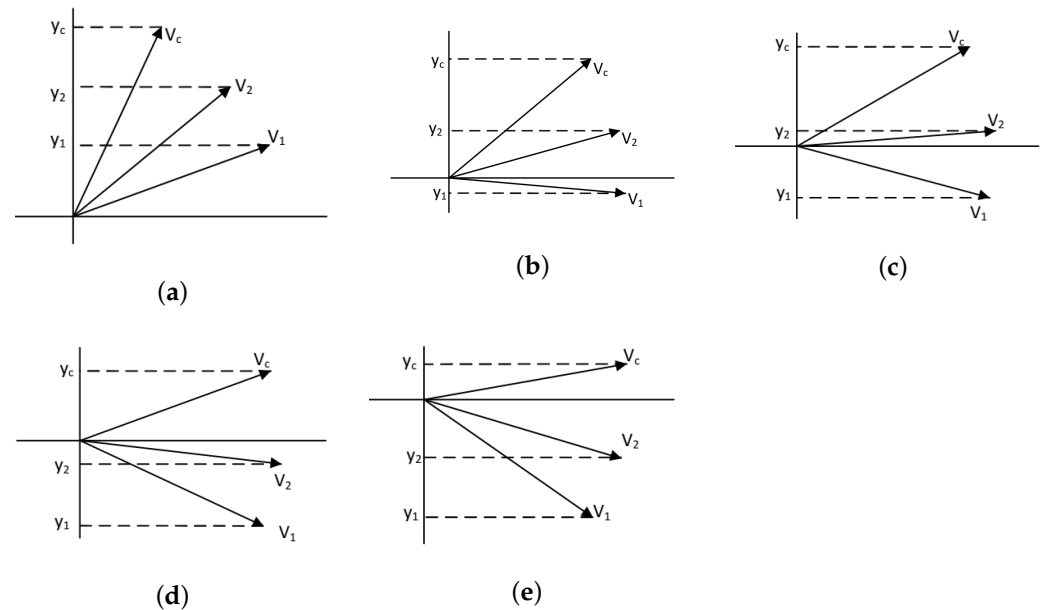


Figure 4. All cases of the first Rotation when the initial vector is in the first quadrant: (a) Case 1. The vector angle is too large and the rotation angle corresponding to y_1 is the best choice. (b) Case 2. The rotation angle corresponding to y_1 is the best choice. (c) Case 3. The rotation angle corresponding to y_2 is the best choice. (d) Case 4. The rotation angle corresponding to y_2 is the best choice. (e) Case 5. No rotation is the best choice.

3.2. Convergence of Proposed Algorithm

To prove the convergence of the proposed PR-CORDIC algorithm, it is necessary to demonstrate that the resultant vector is within the range of convergence for the following iterations. In Algorithm 1, it can be observed that when the angle of the resultant vector is excessively large, potentially exceeding the convergence range of the algorithm, Case 1 in the comparison arises. Therefore, in the convergence proof, we argue that Case 1 (the increase in i_c being zero) only occurs in the initial few iterations.

The proof entails two steps. First, at the beginning of the algorithm, the number of iterations that i_c remains unchanged (first major stage) is no more than three when the initial i_c is 0. As $\tan^{-1}(2^0) = 45.00^\circ$ the angle of the current vector must be within $(-45.00^\circ, 45.00^\circ)$ after no more than three rotations, which angle is 45.00° . Correspondingly, if the initial i_c is 1, the iterations in the first stage will not exceed two times.

Second, when the increase in i in the algorithm is not zero, the increase in i cannot be zero in subsequent iterations. This is described in Theorem 1 and the proof is provided:

Theorem 1. When $\sigma_{y_c} \neq \sigma_{y_1}$, (Cases 2–5), then $\text{sign}(y_n - \text{sign}(y_n) \cdot x_n \cdot 2^{-i_n}) \neq \text{sign}(y_n)$; that is, the condition $\sigma_{y_c} \neq \sigma_{y_1}$ will also be satisfied in the following iteration.

Proof of Theorem 1. Substituting (4) into the proof leads to

$$\text{sign}(y_c - \sigma_c \cdot x_c \cdot 2^{-i_c-q} - \text{sign}(y_n) \cdot (x_c + \sigma_c \cdot y_c \cdot 2^{-i_c-q}) \cdot 2^{-i_c-2}) \neq \text{sign}(y_n), \quad (6)$$

where q denotes the increase in i_c . Furthermore, since x_c is always positive in Case 2–5, simplifying the left side of (6) yields

$$\text{sign}(\sigma_{y_c} \cdot |\tan(\omega_c)| - \sigma_c \cdot 2^{-i_c-q} - \text{sign}(y_n) \cdot (1 + \sigma_c \cdot \sigma_{y_c} \cdot |\tan(\omega_c)| \cdot 2^{-i_c-q}) \cdot 2^{-i_c-2}). \quad (7)$$

In Case 2, $\sigma_c = \sigma_{y_c}$, $q = 0$, $\sigma_{y_n} = -\sigma_{y_c}$, $3 \cdot 2^{-i_c-2} < |\tan(\omega_c)| < 2^{-i_c}$. Therefore, (7) in Case 2 is equal to:

$$\sigma_{y_c} \cdot \text{sign}(|\tan(\omega_c)| - 3 \cdot 2^{-i_c-2} + |\tan(\omega_c)| \cdot 2^{-i_c} \cdot 2^{-i_c-2}). \quad (8)$$

It is evident that (8) is equal to σ_{y_c} and is opposite to σ_{y_n} . This indicates that Theorem 1 holds in Case 2. All possible scenarios are listed in Table 2. The sign of y_n always differs from (7). Therefore, Case 1 only occurs in the initial few iterations in the entire calculation process. $\square$

Table 2. Possible cases in proof of Theorem 1.

	σ_c	q	σ_{y_n}	$ \tan(\omega_c) $	Sign of (7)
Case 2	σ_{y_c}	0	$-\sigma_{y_c}$	$(3 \cdot R^1, 4 \cdot R)$	σ_{y_c}
Case 3	σ_{y_c}	1	σ_{y_c}	$[2 \cdot R, 3 \cdot R]$	$-\sigma_{y_c}$
Case 4	σ_{y_c}	1	$-\sigma_{y_c}$	$(R, 2 \cdot R)$	σ_{y_c}
Case 5	0	—	σ_{y_c}	$[0, R]$	$-\sigma_{y_c}$

¹ $R = 2^{-i_c-2}$

On the basis of the aforescribed proof, it can be concluded that the increase in i_c would not always be zero, implying that the algorithm converges in all quadrants.

3.3. Other Pre-Rotation Counts

To illustrate the algorithm for different pre-rotation counts, we provide details of the algorithm for $r = 3$ in Algorithm 2. Compared to Algorithm 1, Case 6 and Case 7 are additional in Algorithm 2. This is because there are three pre-rotation angles to be compared. After a more detailed comparison (Case 4–7), a more appropriate rotation angle can be obtained. The corresponding examples are shown in Figure 5, with Cases 1–3 not displayed as they are similar to those in Algorithm 1:

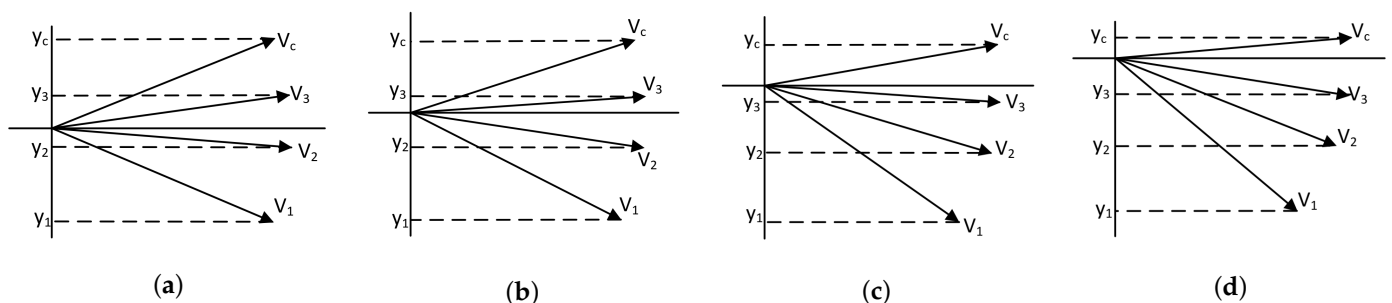


Figure 5. Examples of different cases in Algorithm 2: (a) Case 4. The rotation angle corresponding to y_2 is the best choice. (b) Case 5. The rotation angle corresponding to y_3 is the best choice. (c) Case 6. The rotation angle corresponding to y_3 is the best choice. (d) Case 7. No rotation is the best choice.

When the pre-rotation count r changes, the algorithm will involve more angle comparisons. However, simply increasing the pre-rotation count may not always effectively accelerate the iteration process. This will be explained in the next section.

Furthermore, in terms of convergence proof, although more pre-rotation counts lead to more cases, the proof process for convergence is similar to the one presented in the previous section for the algorithm of $r = 2$. Therefore, we will no longer provide additional proofs.

Algorithm 2 Pre-rotation and Comparison for $r = 3$

Input: i_c, x_c, y_c, ω_c

Output: i_n, x_n, y_n, ω_n

Function: $[i_n, x_n, y_n, \omega_n] = \text{Compare}(i_c, x_c, y_c, \omega_c)$

```

1:  $\sigma_{y_c} = \text{sign}(y_c)$ .
2:  $y_1 = y_c + \sigma_{y_c} \cdot x_c \cdot 2^{-i_c}$ .
3:  $y_2 = y_c + \sigma_{y_c} \cdot x_c \cdot 2^{-i_c-1}$ .
4:  $y_3 = y_c + \sigma_{y_c} \cdot x_c \cdot 2^{-i_c-2}$ .
5:  $\sigma_{y_1} = \text{sign}(y_1), \sigma_{y_2} = \text{sign}(y_2), \sigma_{y_3} = \text{sign}(y_3)$ .
6: if  $\sigma_{y_c} = \sigma_{y_1}$  then
7:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c}$ 
8:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c})$  ▷ Case 1
9:    $y_n = y_1, i_n = i_c$ 
10: else if  $\sigma_{y_c} \neq \sigma_{y_1}, \sigma_{y_c} = \sigma_{y_2}, |y_1| < |y_2|$  then
11:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c}$ 
12:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c})$  ▷ Case 2
13:    $y_n = y_1, i_n = i_c + 2$ 
14: else if  $\sigma_{y_c} \neq \sigma_{y_1}, \sigma_{y_c} = \sigma_{y_2}, |y_1| > |y_2|$  then
15:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-1}$ 
16:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-1})$  ▷ Case 3
17:    $y_n = y_2, i_n = i_c + 2$ 
18: else if  $\sigma_{y_c} \neq \sigma_{y_2}, \sigma_{y_c} = \sigma_{y_3}, |y_2| < |y_3|$  then
19:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-1}$ 
20:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-1})$  ▷ Case 4
21:    $y_n = y_2, i_n = i_c + 3$ 
22: else if  $\sigma_{y_c} \neq \sigma_{y_2}, \sigma_{y_c} = \sigma_{y_3}, |y_2| > |y_3|$  then
23:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-2}$ 
24:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-2})$  ▷ Case 5
25:    $y_n = y_3, i_n = i_c + 3$ 
26: else if  $\sigma_{y_c} \neq \sigma_{y_3}, |y_3| < |y_c|$  then
27:    $x_n = x_c + \sigma_{y_c} \cdot y_c \cdot 2^{-i_c-2}$ 
28:    $\omega_n = \omega_c + \sigma_{y_c} \cdot \tan^{-1}(2^{-i_c-2})$  ▷ Case 6
29:    $y_n = y_3, i_n = i_c + 3$ 
30: else if  $\sigma_{y_c} \neq \sigma_{y_3}, |y_3| > |y_c|$  then
31:    $x_n = x_c, \omega_n = \omega_c$  ▷ Case 7
32:    $y_n = y_c, i_n = i_c + 3$ 
33: end if
34: return  $i_n, x_n, y_n, \omega_n$ 

```

3.4. Number of Iterations

In this subsection, the number of iterations required by the PR-CORDIC algorithm is demonstrated. There are n_1 iterations from the first stage (where i_c does not change) and n_2 iterations from the second stage.

The angle of the input vector and the fixed initial angle in the iteration determine the value of n_1 . As shown in Figure 6, as the angle of the input vector changes, the value of n_1 could be 0, 1, 2 when the maximum rotation angle and also the fixed initial angle are set to $\tan^{-1}(2^1) \approx 63.43^\circ$, while the value of n_1 could be 0, 1, 2, 3 when the maximum rotation angle is set to $\tan^{-1}(2^0) = 45.00^\circ$:

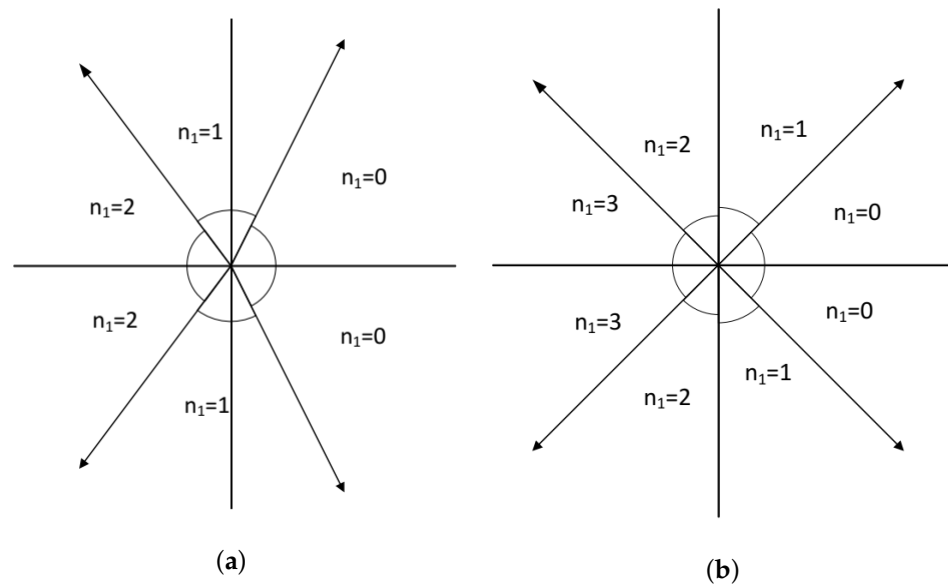


Figure 6. Classification of initial vectors according to angle: (a) Maximum rotation angle is 63.43°. (b) Maximum rotation angle is 45.00°.

The value of n_2 is related to the growth rate of i_c and the required accuracy of the result. From Algorithm 1, it is known that the greater the number of pre-rotations in each iteration, the greater would be the likely increase in i_c . If we approximate $\arctan(2^{-i})$ as twice $\arctan(2^{-1-i})$ then the average number of conventional CORDIC iterations that are equivalent to each PR-CORDIC iteration is denoted by

$$2 \cdot \frac{1}{2} + 3 \cdot \frac{1}{4} + \dots + r \cdot \frac{1}{2^{r-1}} + r \cdot \frac{1}{2^{r-1}} = 3 - 2^{2-r}. \quad (9)$$

Therefore, when the accuracy of the result is required to be 2^{-t} ,

$$n_2 = \frac{t+1}{3-2^{2-r}}, \quad (10)$$

where r represents the number of pre-rotations. Therefore, the number of iterations does not decrease linearly when r increases.

In general, the proposed algorithm achieves a flexible selection of rotation angles through pre-rotation, thereby expanding the convergence range and reducing latency. However, since the rotations are not fixed as in traditional CORDIC, the scale factor is no longer constant. Therefore, in computations where the scale factor is needed, such as calculating the magnitude of a vector, the algorithm requires additional calculations, which may lead to a lack of efficiency. On the other hand, for computations that do not require the scale factor, such as arctangent, the proposed algorithm is more suitable.

4. Error Analysis

Errors in the CORDIC algorithm arise from two main sources: the word length used in the implementation and the number of iterations. For a more intuitive error analysis, a software simulation was used to analyze and compare the conventional algorithm against the proposed PR-CORDIC algorithm.

During the software simulation, the algorithmic error was characterized by the angle error defined as

$$\mu_r = |\alpha_t - \alpha_e|, \quad (11)$$

where α_t stands for the true values of $\tan^{-1}(\frac{y_0}{x_0})$ and α_e represents the experimental values computed by CORDIC. All the arithmetic data were single floating-point numbers, except for α_t and μ_r , which were double floating-point numbers. All the data and figures were generated using MATLAB R2016b software.

For a more reasoned analysis of the errors in PR-CORDIC, we initially simulated the computation of the arctangent function using conventional methods. During these simulations, the input range for x_0 was defined as $[0, 10,000]$ and for y_0 as $[-10,000, 10,000]$, and then 10,000 sets of data were randomly selected as inputs. The variation of maximum error and average error with the number of iterations is shown in Figure 7. Once the iteration count exceeded 23, further increases did not discernibly reduce either the maximum or average error. This is primarily attributable to the mantissa in the single-precision floating-point representation comprising only 23 bits.

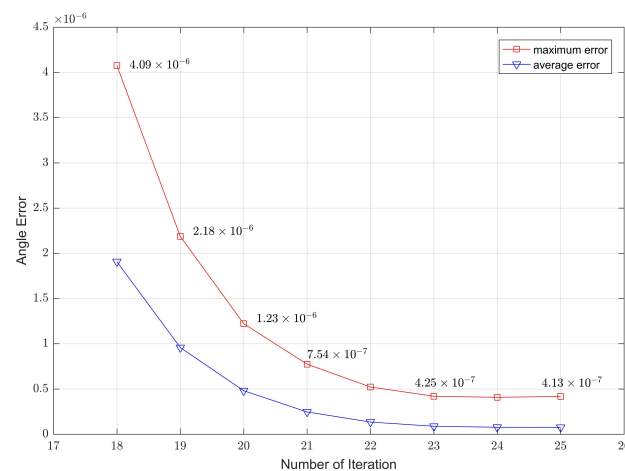


Figure 7. Errors in conventional CORDIC algorithm.

In simulating the PR-CORDIC algorithm, the input range for x_0 was modified to $[-10,000, 10,000]$, again with 10,000 data sets being input randomly. According to the discussion on the number of iterations in the preceding section, both the pre-rotation count r and the maximum rotation angle α_0 are pivotal in shaping the relationship between iteration counts and precision. The experimental results depicted in Figure 8a,b indicate that while an increase in the pre-rotation count leads to a reduction in average error its impact on the maximum error is evidently negligible. Moreover, setting the maximum rotation angle to $\tan^{-1}(2)$ allows the algorithm to attain optimal error values comparable to conventional algorithms with a minimal number of iterations.

Figure 9 illustrates the relationship between the angle of the input vector and the associated error after 14 iterations of the proposed algorithm. Owing to the limited precision of single-precision floating-point numbers, with only 23 significant bits, the error increases with the angle of the vector. However, it does not exceed 5×10^{-7} .

In summary, the proposed PR-CORDIC algorithm demonstrates computational precision comparable to conventional algorithms while concurrently achieving a notable reduction in the number of iterations. Furthermore, a pre-rotation count of 2 exerts the most substantial influence on the maximum error. When the maximum rotation angle is set to $\tan^{-1}(2)$, the algorithm achieves the minimum number of iterations required to attain the theoretically optimal precision.

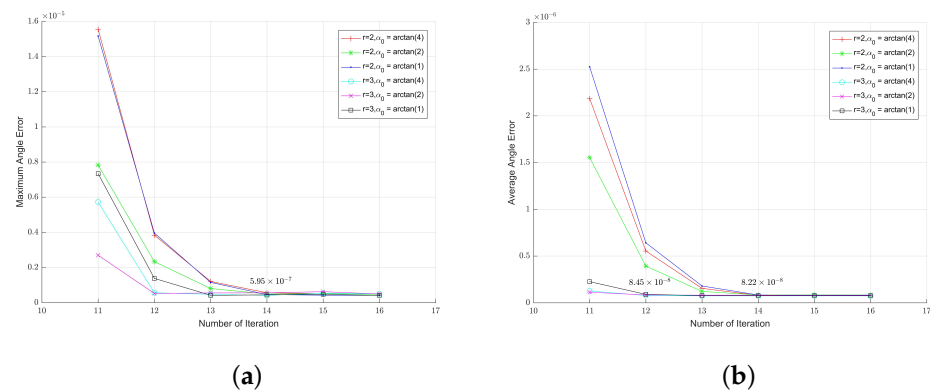


Figure 8. Error of PR-CORDIC algorithm: (a) Maximum error of PR-CORDIC algorithm under different parameters. (b) Average error of PR-CORDIC algorithm under different parameters.

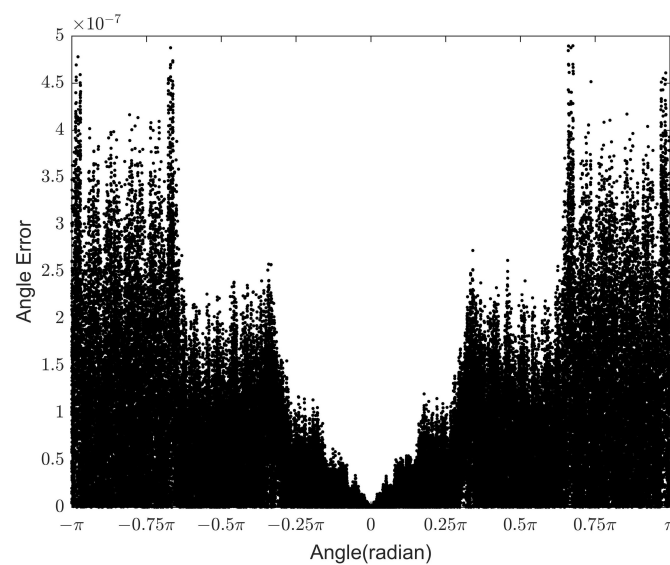


Figure 9. The correlation between error and the input vector angle.

5. Proposed Architecture

In this section, we present two architectures based on the PR-CORDIC algorithm to compute the arctangent function. The first is an iterative architecture with low throughput and low resource requirements, while the other is a pipeline architecture with high throughput. The pipeline architecture optimizes a portion of the original algorithm to reduce the utilization of hardware.

5.1. Iterative Architecture

According to the discussion on errors in the previous section, when the pre-rotation count is set to $r = 2$ and the initial rotation angles $\alpha_0 = \tan^{-1}(2)$, the algorithm reaches its minimum achievable error after the 13th iteration, and no further reduction is observed. The architecture we present employs single-precision floating-point arithmetic. Figure 10a shows the architecture of the pre-rotation process element. Figure 10b depicts the architecture of the process element for micro-rotation following comparison. The iterative architecture of the entire calculation process is illustrated in Figure 11.

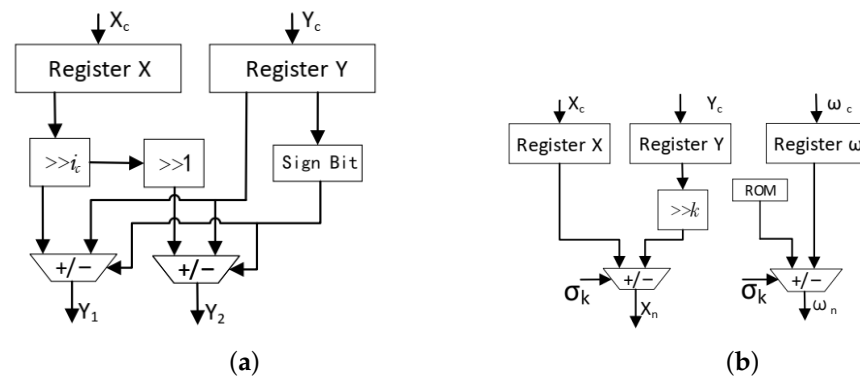


Figure 10. Details of the entire proposed iterative architecture. Both PE1 and PE2 require two adders each: (a) PE1 for pre-rotation. (b) PE2 for rotation.

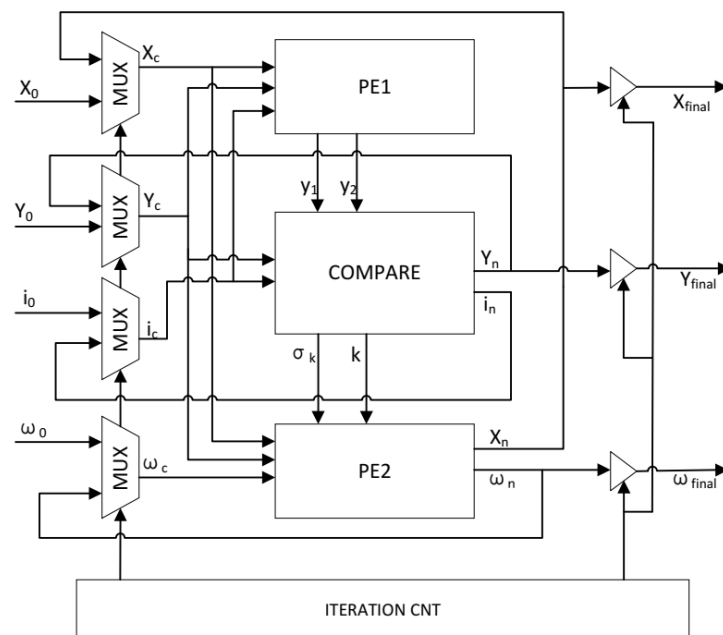


Figure 11. Proposed iterative architecture. The architectures of PE1 and PE2 are shown in Figure 10. COMPARE completes the logic of Algorithm 1, while ITERATION CNT is used to control the start and end of the entire calculation process.

The pre-rotation process element is used to obtain the results of (5). The comparator module is used to implement the comparison logic in Algorithm 1, which obtains the optimal rotation angle according to the input pre-rotation value. The ITERATION CNT serves as an iteration counter, controlling the initiation and termination of the entire computation process. The input data are updated and the output data ω_{final} , x_{final} , y_{final} , K_{final} are valid when all iterations are completed.

5.2. Pipeline Architecture

To enhance the data throughput, we propose a pipeline architecture for PR-CORDIC. Simply breaking down the iterative process into various stages within a pipeline architecture may unnecessarily increase resource usage. Therefore, in this architecture, the proposed algorithm is modified to simplify the pre-rotation process implementation.

Based on the error analysis from the previous section, when operating with single-precision floating-point data, the minimum rotational precision in the architecture is $\tan^{-1}(2^{-22})$ and the rotation angle can be either one of two fixed angles or zero in each

stage. Therefore, this pipelined architecture has 12 stages in total. Notably, owing to the fixed rotation angle per iteration in this architecture, its convergence domain is no longer arbitrary for any vector.

Its maximum input vector angle becomes

$$\tan^{-1}(2^1) + \tan^{-1}(2^{-1}) + \dots + \tan^{-1}(2^{-21}), \quad (12)$$

which is approximated as 99.51° . To maintain the arbitrary nature of the input vectors, preprocessing and postprocessing of (13) are required before the data enter the iteration and after the iteration ends:

$$\begin{cases} x_0 = x_{in}, \omega_{out} = \omega_n & \text{if } x_{in} \geq 0 \\ x_0 = -x_{in}, \omega_{out} = \pi - \omega_n & \text{if } x_{in} < 0, y_{in} \geq 0 \\ x_0 = -x_{in}, \omega_{out} = -\pi - \omega_n & \text{if } x_{in} < 0, y_{in} < 0 \end{cases} \quad (13)$$

where x_{in} and x_{in} represent the input to the entire architecture, ω_{out} denotes the output, x_0 stands for the data entering the first iteration, and ω_n denotes the data output at the end of the iteration. After processing, the architecture can handle arbitrary vector inputs, producing output values within the range of $(-\pi, \pi]$.

Figure 12 illustrates the iterative process within the pipeline architecture when performing floating-point calculations. In contrast to the conventional CORDIC algorithm's architecture depicted in Figure 2, each stage in Figure 12 incorporates additional adders, three multiplexers (MUXs) (with an extra one employed during the selection of y_i in COMPARE), and two comparators (COMPs). Notwithstanding this observation, the total number of stages is reduced from 23 to 12. Figure 13 depicts the data preprocessing and postprocessing architecture designed to achieve (13). In this design, the sign bit of x_{in} serves as the input controlling the iteration, and it also determines the final output after a delay of 12 cycles. Meanwhile, the sign bit of y_{in} only influences the result after the delay. The primary resource consumption in this architecture involves three multiplexers and one adder.

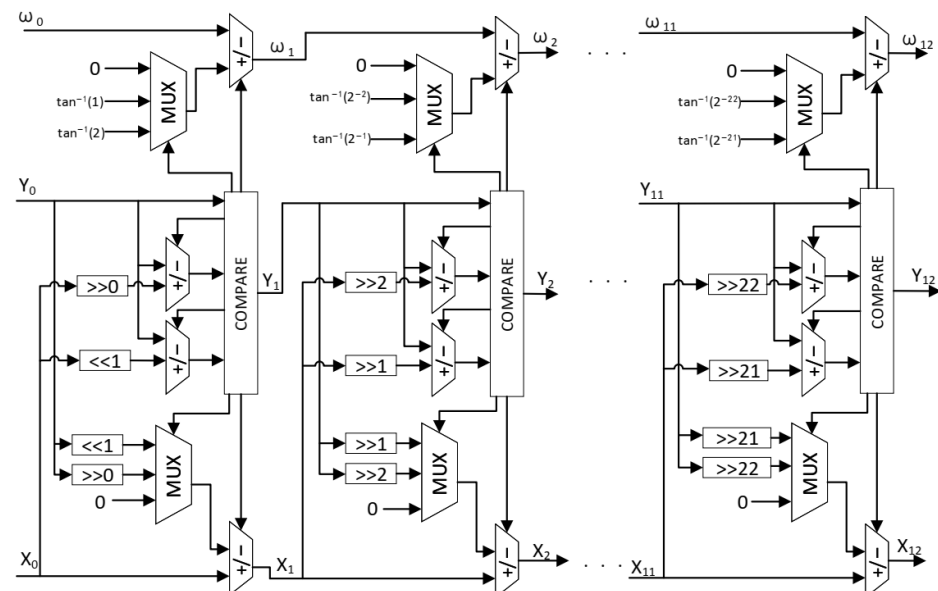


Figure 12. The iteration process pipeline architecture. It consists of 12 stages, with each stage requiring four adders, and there are two possible rotation angles.

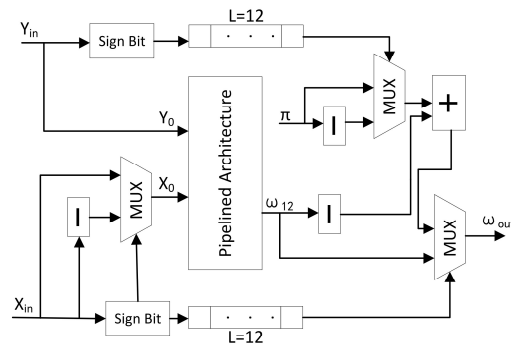


Figure 13. The preprocessing and postprocessing architecture. Through preprocessing and postprocessing, the convergence range of the rotation angles can be achieved within $[-\pi, \pi]$.

Table 3 analyzes the comparative resource utilization of the proposed iterative/pipeline architecture and the conventional counterpart. The proposed architecture is shown to reduce markedly the reliance on adders, albeit by introducing a surplus of multiplexers and comparators. This distinctive feature substantially reduces resource requirements, particularly when employing single-precision floating-point calculations.

Table 3. Comparison of hardware resource.

	Iterative		Pipeline	
	Conventional	Proposed	Conventional	Proposed
adders	3	4	69	49
MUXs	4	5	0	39
COMPs	0	2	0	24
shifters	2	3	46	48

6. Implementation and Comparison

The proposed architecture was coded in the Verilog Hardware Description language (HDL), simulated using Xilinx Vivado 2017.4 software by applying 16-bit fixed-point inputs and float-point inputs, and implemented on an FPGA platform. To adequately compare the performance of PR-CORDIC with other algorithms in the computation of the arctangent function, we implemented the iterative and pipeline architectures for floating-point inputs, as well as the iterative architecture for 16-bit fixed-point inputs. The details of the estimated performance are discussed in this section.

As shown in Table 4, we implemented iterative and pipeline architectures based on the PR-CORDIC and conventional CORDIC algorithms and compared them. The resource comparisons included LUTs, flip-flops (FFs), the iteration clock cycles, throughput rates, and other characteristics. It can be seen from the table that the proposed methods have a relatively lower maximum frequency, which is due to the addition of pre-rotation and COMPARE steps in each iteration. In the iterative architecture, the latency and throughput in computing the arctangent function are improved because PR-CORDIC can skip some unnecessary rotation angles compared to the conventional algorithm. In addition, the PR-CORDIC-based pipeline architecture reduces the resource usage and latency as the number of iterations is reduced from 23 to 12. Moreover, the input vector angle range of the proposed architecture is $(-\pi, \pi)$, which is more consistent with the requirements of real-world usage. In summary, the proposed architecture guarantees accuracy in the results and reduces the resource usage and latency for standard single floating-point inputs.

Table 4. Comparison of Implementation.

	Iterative		Pipeline	
	Conventional	Proposed	Conventional	Proposed
Data Format	SFP ¹	SFP	SFP	SFP
FPGA	Kintex-7	Kintex-7	Kintex-7	Kintex-7
LUTs	1837	2461	37,001	26,509
FFs	103	104	2087	1155
Throughput ²	1/23	1/13	1	1
Latency ³	23	13	23	12
Max Frequency	376 MHz	308 MHz	342 MHz	291 MHz
Range	$[-\pi/2, \pi/2]$	$[-\pi, \pi]$	$[-\pi/2, \pi/2]$	$[-\pi, \pi]$

¹ SFP: Single floating-point. ² The unit is the amount of data output per clock cycle. ³ The unit is clock cycles.

To verify that the proposed method is state-of-the-art, we used several previous methods to compute arctangent functions for comparison. We again implemented the fixed-point PR-CORDIC based iterative architectures on FPGAs to better compare with the algorithm in [3].

As shown in Table 5, the first three rows present the performance of other work in computing the arctangent, and the last three rows present the performance of this paper. The main performance comparisons are LUTs, FFs, memory, DSP, throughput rate, latency. As observed, the proposed architectures can be flexibly used to compute arctangent functions for floating-point and fixed-point inputs. When fixed-point inputs are used, although the number of LUTs is increased in the proposed architecture compared to [3], the feature that does not require DSP and memory is maintained. In addition, the iterative architecture evidences less latency for floating-point inputs compared to the previous methods [23]. The data throughput rate of the pipeline architecture can process one data element per clock cycle, which facilitates its application in real-time systems.

Table 5. Comparison of architecture for arctangent computation.

	[23,24]	[3]	[25]	Proposed	Proposed	Proposed
Algorithm	HA-CORDIC	Approximation	Preprocessing	PR-CORDIC	PR-CORDIC	PR-CORDIC
Precision (bits)	23	16	113	16	23	23
Data Format	FLP ¹	FIP	QFLP	FIP	SFLP	SFLP
Architecture	Iterative	Table-based	Iterative	Iterative	Iterative	Pipeline
LUTs	1139	250	37,697	390	2461	26,490
FFs	498	128	-	56	104	1091
Memory	11	2	-	0	0	0
DSP	8	1	-	0	0	0
Throughput (data/clock)	1/26	1/8	1/32	1/8	1/13	1
Latency (clock cycles)	26	8	32	8	13	12

¹ FLP: floating-point, FIP: fixed-point, QFLP: quadruple-precision floating-point, SFLP: single floating-point.

7. Conclusions

This paper presents a CORDIC acceleration algorithm based on pre-rotation. The PR-CORDIC algorithm efficiently reduces the number of iterations, does not place restrictions on the input vector angle, and ensures the accuracy of the result. In addition, the corresponding iterative and pipeline architecture for the computation of the arctangent is detailed. The experimental results demonstrate that, in single-floating calculations, the proposed methods can effectively lower latency, reduce resource usage, and widen the input angle range compared to previous methods. And the proposed pipelined architecture can provide a higher throughput rate.

Author Contributions: Conceptualization, K.L.; methodology, K.L. and H.F.; software, K.L.; validation, K.L., B.Z. and Z.M.; writing—original draft preparation, K.L.; writing—review and editing, F.Y., B.Z. and Q.X.; project administration, F.Y. and Q.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

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

References

- de Lassus Saint-Geniès, H.; Defour, D.; Revy, G. Exact Lookup Tables for the Evaluation of Trigonometric and Hyperbolic Functions. *IEEE Trans. Comput.* **2017**, *66*, 2058–2071. [\[CrossRef\]](#)
- Rajan, S.; Wang, S.; Inkol, R.; Joyal, A. Efficient approximations for the arctangent function. *IEEE Signal Process. Mag.* **2006**, *23*, 108–111. [\[CrossRef\]](#)
- Torres, V.; Valls, J. A Fast and Low-Complexity Operator for the Computation of the Arctangent of a Complex Number. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 2663–2667. [\[CrossRef\]](#)
- Volder, J.E. The CORDIC Trigonometric Computing Technique. *IRE Trans. Electron. Comput.* **1959**, *EC-8*, 330–334. [\[CrossRef\]](#)
- Walther, J.S. A unified algorithm for elementary functions. In Proceedings of the Spring Joint Computer Conference on - AFIPS '71 (Spring), Atlantic City, NJ, USA, 18–20 May 1971; pp. 379–385. [\[CrossRef\]](#)
- Verma, A.; Kiyawat, K.; Das, B.P.; Meher, P.K. An Efficient Scaling-Free Folded Hyperbolic CORDIC Design Using a Novel Low-Complexity Power-of-2 Taylor Series Approximation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2023**, *31*, 1167–1177. [\[CrossRef\]](#)
- Paz, P.; Garrido, M. CORDIC-Based Computation of Arcsine and Arccosine Functions on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2023**, *70*, 3684–3688. [\[CrossRef\]](#)
- Heidarpur, M.; Ahmadi, A.; Ahmadi, M.; Rahimi Azghadi, M. CORDIC-SNN: On-FPGA STDP Learning with Izhikevich Neurons. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2019**, *66*, 2651–2661. [\[CrossRef\]](#)
- Leigh, A.J.; Heidarpur, M.; Mirhassani, M. A Resource-Efficient and High-Accuracy CORDIC-Based Digital Implementation of the Hodgkin–Huxley Neuron. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2023**, *31*, 1377–1388. [\[CrossRef\]](#)
- Delosme, J.M. A Signal Flow Graph Approach to the Resolution of Spherical Triangles Using CORDIC. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2022**, *69*, 5159–5170. [\[CrossRef\]](#)
- Mohamed, N.A.; Cavallaro, J.R. A Unified Parallel CORDIC-Based Hardware Architecture for LSTM Network Acceleration. *IEEE Trans. Comput.* **2023**, *72*, 2752–2766. [\[CrossRef\]](#)
- Hu, Y.; Naganathan, S. An angle recoding method for CORDIC algorithm implementation. *IEEE Trans. Comput.* **1993**, *42*, 99–102. [\[CrossRef\]](#)
- Meher, P.K.; Park, S.Y. CORDIC Designs for Fixed Angle of Rotation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2013**, *21*, 217–228. [\[CrossRef\]](#)
- Antelo, E.; Villalba, J.; Bruguera, J.; Zapata, E. High performance rotation architectures based on the radix-4 CORDIC algorithm. *IEEE Trans. Comput.* **1997**, *46*, 855–870. [\[CrossRef\]](#)
- Antelo, E.; Lang, T.; Bruguera, J. Very-high radix circular CORDIC: Vectoring and unified rotation/vectoring. *IEEE Trans. Comput.* **2000**, *49*, 727–739. [\[CrossRef\]](#)
- Phatak, D. Double step branching CORDIC: A new algorithm for fast sine and cosine generation. *IEEE Trans. Comput.* **1998**, *47*, 587–602. [\[CrossRef\]](#)
- Shukla, R.; Ray, K.C. Low Latency Hybrid CORDIC Algorithm. *IEEE Trans. Comput.* **2014**, *63*, 3066–3078. [\[CrossRef\]](#)
- Garrido, M.; Källström, P.; Kumm, M.; Gustafsson, O. CORDIC II: A New Improved CORDIC Algorithm. *IEEE Trans. Circuits Syst. II Express Briefs* **2016**, *63*, 186–190. [\[CrossRef\]](#)
- Mahdavi, H.; Timarchi, S. Improving Architectures of Binary Signed-Digit CORDIC with Generic/Specific Initial Angles. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2020**, *67*, 2297–2304. [\[CrossRef\]](#)
- Zhu, B.; Lei, Y.; Peng, Y.; He, T. Low Latency and Low Error Floating-Point Sine/Cosine Function Based TCORDIC Algorithm. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *64*, 892–905. [\[CrossRef\]](#)
- Wu, H.; Lin, L.; Sun, H.; Zeng, X.; Chen, Y. A Decision-Based CORDIC Hardware for Arc Tangent Calculation. In Proceedings of the 2023 IEEE 15th International Conference on ASIC (ASICON), Nanjing, China, 24–27 October 2023; pp. 1–4. [\[CrossRef\]](#)
- 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 Regul. Pap.* **2009**, *56*, 1893–1907. [\[CrossRef\]](#)
- Nguyen, H.T.; Nguyen, X.T.; Hoang, T.T.; Le, D.H.; Pham, C.K. Low-resource low-latency hybrid adaptive CORDIC with floating-point precision. *IEICE Electron. Express* **2015**, *12*, 20150258. [\[CrossRef\]](#)

24. Nguyen, H.T.; Nguyen, X.T.; Pham, C.K. A Low-Power Hybrid Adaptive CORDIC. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 496–500. [[CrossRef](#)]
25. He, C.; Yan, B.; Xu, S.; Zhang, Y.; Wang, Z.; Wang, M. Research and Hardware Implementation of a Reduced-Latency Quadruple-Precision Floating-Point Arctangent Algorithm. *Electronics* **2023**, *12*, 3472. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.