A Novel Implementation of CORDIC Algorithm Based on Dynamic Microrotation Generation

Aidin Shiri*

Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

Article Info

Abstract

Mathematical functions are generally classified into two groups. Functions such as logarithmic or trigonometric functions are called elementary functions, and functions such as sin(1/x), which are a combination of two or more elementary functions, are called combinatorial functions. Mathematical functions have many applications in digital devices, such as digital signal processing, image processing, and telecommunication systems. Although software computation of mathematical functions in digital systems has flexibility and convenience advantage, sometimes it does not keep up with the real-time requirements of modern digital systems. To solve this problem, various algorithms have been proposed to implement mathematical functions on hardware. Hardware implementations tend to have higher throughput compared to software implementations, but usually, they suffer in terms of accuracy. In this paper, we propose a novel method for calculating the elementary trigonometric functions using the CORDIC algorithm based on the dynamic microrotation generation technique. We implement our design on Spartan-6 FPGA. Results show our method outperforms similar works in terms of throughput and power consumption while exploiting less hardware.

1. Introduction

Elementary functions such as sin(x), cos(x), log(x), exp(x), and … are an important group of mathematical functions and their common point is that they cannot be calculated precisely by basic math operations and require infinite series of algebraic polynomials to estimate the final result with the desired accuracy. These functions have many applications in engineering fields such as in robotics [1], 3D graphics [2], Singular Value Decomposition [3], Digital signal processing [4], telecommunication systems [5], Data compression [6], adaptive filters [7], and so forth. While software computation of the elementary functions usually yields higher accuracy, because processors are engaged in instruction fetch-decode-execute cycle, they tend to be slower compared to the hardware implementations. For example, the calculation of trigonometric functions has the worst delay, and to estimate them, all of the mathematical operations must be engaged. Therefore, it is essential to be able to calculate the elementary functions real-time to prevent system performance degradation [8]. Hardware implementation of the mathematical functions increases the speed of computation, but since achieving higher accuracy results in exponential growth in resource utilization, a careful tradeoff between the resource usage, throughput, and accuracy should be taken into account.

2. Background

Generally, there are three conventional methods for approximating the elementary functions in hardware:

- Table lookup method
- Polynomial estimation
- CORDIC algorithm
2.1. Table lookup method

The table lookup method [9] is the most straightforward method of calculating the elementary functions. In this method, the function domain is divided into several successive intervals, and the value along with the function's gradient in the breakpoints are stored in the memory. If the input argument was not one of the stored points, the value can be calculated by interpolation between the two breakpoints. This method is suitable for lower precisions where there are few breaking points. In higher precisions, the size of the table increases dramatically, which requires large memory for calculating the final result.

2.2. Polynomial estimation method

A polynomial in the below form could be used to estimate every arbitrary continuous function within a given interval:

\[ f(x) = P_n(x) = P_n x^n + P_{n-1} x^{n-1} + \ldots + P_1 x + P_0 \]  

Where \( n \) is the degree of the polynomial, \( p_i \) is the coefficient, and \( x \) is the domain of the argument. These factors also determine the accuracy and speed of the computation. The accuracy of the estimation could be increased by using a higher-order polynomial, but the number of multipliers, adders, and coefficients stored in the memory grows exponentially as well. If the range of the argument’s domain was large, estimation requires a polynomial with a higher degree, which means more resource utilization and slower computation. Chebyshev Minimax [10] is a method to reduce the maximum error by storing pre-calculated coefficients in a ROM.

2.3. CORDIC algorithm

CORDIC algorithm [11] is one of the most flexible methods among the algorithms for computing the elementary functions, which can calculate trigonometric functions (\( \sin, \cos \), \( \ldots \)), Hyperbolic functions (\( \sinh, \cosh, \ldots \)), inverse trigonometric functions (\( \cos^{-1}, \sin^{-1}, \ldots \)), Inverse Hyperbolic functions (\( \cosh^{-1}, \sinh^{-1}, \ldots \)) and \( \exp \) with the same hardware. CORDIC algorithm only needs two shifters and three adders for calculation, which makes the hardware implementation simple and fast.

3. Related Works

Despite the CORDIC algorithm is straightforward regarding the hardware implementation, it has a drawback; convergence to the final answer with \( N \) bits of accuracy requires \( N \) iterations. The performance of the CORDIC algorithm could be improved by reducing the number of iterations and consequently, the overall delay. Besides the conventional CORDIC algorithm [11], many works have been proposed to improve the various aspects of the CORDIC algorithm.

3.1. Higher radix CORDIC

This method uses powers greater than 2, (\( 2^\sigma \) in the conventional CORDIC), to estimate \( \tan(2^{-\sigma}) \). For achieving \( N \) bits of accuracy, the radix-4 CORDIC algorithm requires \( N/2 \) micro-rotation, which is half of the required number for the conventional algorithm. However, the computational time per micro-rotation is higher than the basic Radix-2 version; additionally, the hardware is more complex. The number of microrotations shifts varies with the value of \( \sigma \); moreover, the scale coefficient \( K \) changes according to the \( \sigma \) for each rotation angle [12].

3.2. Pipelined CORDIC architecture

Because CORDIC iterations are similar, it can be implemented it in a pipelined fashion. Pipelined CORDIC is of great importance in many digital signal processing applications because of its higher throughput [13].

3.3. Angle recoding method

The goal of the angle recoding method is to reduce the number of iterations by decomposing the rotation angle from a linear combination of some fixed angles instead of all of them. The scale coefficient \( K \) is not constant in this method, and it must be calculated separately for each angle. The Angle recoding algorithms presented by [14] reduces the number of iterations by 50% while keeping \( N \) bits precision similar to the conventional CORDIC algorithm.

3.4. Scaling free CORDIC

In this method, Taylor's expansion of \( \sin \) and \( \cos \) are used to eliminate the need for scale coefficient. The selection of microrotations for achieving the desired accuracy depends on the degree of the polynomial. Results show that maximum error does not change significantly after the third degree [15].
3.5. Fixed angle of rotation method

In this method, the CORDIC algorithm is optimized for the fixed and predetermined angles (odd angles in the interval of [0; 45]). Because the micro-rotations are predefined, hardwired pre-shifting with barrel shifters are used to reduce the complexity of the hardware and the time of calculation. The algorithm needs four micro-rotations for achieving the desired accuracy, but it also requires three more micro-rotations for scaling. The predefined angles are decomposed offline before the start of the computation, and relevant control bits along with the size of shifts for microrotations are saved in a ROM [16].

Besides the discussed methods, some other works have been proposed, each improves different aspects of the CORDIC algorithm [17-23].

4. CORDIC Algorithm

CORDIC (Coordinate Rotation Digital Computer) algorithm first proposed by J. Volder in 1959. Volder developed this method for improving the performance of the real-time navigation system. He exploited the idea of calculating the \( \sin \) and \( \cos \) of an angle by moving a vector from its initial location on the X-axis to its final location, which is a vector with an angle of \( \theta \) from the X-axis. The vector moves in small angular steps and meanwhile, updates the values of \( X \) and \( Y \) for each step. When the vector reaches its final location, the values of \( X \) and \( Y \) are \( \cos \)ine and \( \sin \) of the angle respectively.

CORDIC algorithm has two vector rotation and vector translation modes. In the vector rotation mode, the initial location of the vector and the rotation angle are determined, and the final location must be calculated. In the vector translation mode, however, the initial and final locations of the vector are determined, and the angle between the two vectors must be calculated. Vector rotation is performed by applying a sequence of angular steps named micro-rotations. For every angle, micro-rotations are the same, and depending on the precision, the number of required micro-rotations varies. For \( N \) bits accuracy, \( N \) fixed angles, and consequently, \( N \) microrotations are required. Fixed angles are chosen that to simplify the process of calculating vector rotation. Figure 1 illustrates the micro-rotations for decomposing 25 angles.

![Figure 1. Decomposing a 25-degree angle after nine micro-rotations](image1)

![Figure 2. Vector rotation with angle \( a_i \)](image2)
Mathematical equations of the CORDIC iterations on the coordinate plane are presented in equations (2-17). Figure 2 shows the unit circle with two unit vectors \( P, Q \), where \( a_i \) is the angle between two vectors. The vector \( P \) is rotated toward the vector \( Q \) with angle \( a_i \). The rotation of the vector \( P \) with angle \( a_i \) to the target vector \( Q \) is described in the following equations:

\[
Q = e^{\phi \cdot a_i} P
\]  
\[
(x_{i+1} + jy_{i+1}) = (\cos a_i + j\sin a_i) \ast (x_i + jy_i) =
\]
\[
(x_i \cos a_i + jy_i \cos a_i + jx_i \sin a_i - y_i \sin a_i)
\]  
\[
(\sigma_i = \pm 1)
\]

By separating real and imaginary parts, equation (4) becomes:

\[
x_{i+1} = x_i \cos a_i - y_i \sin a_i
\]
\[
y_{i+1} = x_i \sin a_i + y_i \cos a_i
\]  
\[
\text{Equation (5&6) can be written in the matrix form:}
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = 
\begin{bmatrix}
\cos a_i & -\sin a_i \\
\sin a_i & \cos a_i
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\text{Factorization of } \cos a_i \text{ from equation (7) results in:}
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = \cos a_i \begin{bmatrix}
1 & \tan a_i \\
\tan a_i & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\text{Also, by replacing } \tan a_i \text{ with the approximation of } 2^{-i} \text{ we will have:}
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = \cos a_i \begin{bmatrix}
1 & -2^{-i} \\
2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\text{Equation (9) can be implemented by shifters, and the need for time-consuming and expensive multiplication operation is eliminated. In general, } a_i \text{ can be positive or negative, corresponding to the clockwise or counterclockwise rotation.}
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = \cos a_i \begin{bmatrix}
1 & -\sigma_i 2^{-i} \\
\sigma_i 2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\text{Where } \sigma_i = \pm 1
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = \frac{1}{\sec a_i} \begin{bmatrix}
1 & -\sigma_i 2^{-i} \\
\sigma_i 2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\frac{1}{\sqrt{1 + \tan^2 a_i}} \begin{bmatrix}
1 & -\sigma_i 2^{-i} \\
\sigma_i 2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\frac{1}{\sqrt{1 + \sigma_i^2 2^{-2i}}} \begin{bmatrix}
1 & -\sigma_i 2^{-i} \\
\sigma_i 2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
\[
\begin{bmatrix}
[x_{i+1}] \\
[y_{i+1}]
\end{bmatrix} = \frac{1}{k_i} \begin{bmatrix}
1 & -\sigma_i 2^{-i} \\
\sigma_i 2^{-i} & 1
\end{bmatrix} 
\begin{bmatrix}
x_i \\
y_i
\end{bmatrix}
\]
In Fig. 3, a sequence of micro-rotations \((a_0,a_1,a_2,...,a_n)\), rotates \(P[x_i,y_i]\) towards \(Q[x_f,y_f]\).

\[
x_{i+1} = \frac{1}{k_i} (x_i - \sigma_i 2^{-i} y_i)
\]
\[
y_{i+1} = \frac{1}{k_i} (y_i + \sigma_i 2^{-i} x_i)
\]
If all of the micro-rotations are used for decomposing an angle, a fixed and constant-coefficient can be calculated by multiplying all of the scaling factors. This general scaling factor can be applied to the final results either at the beginning or the end of the process.

\[
\begin{bmatrix}
  x_f \\
  y_f 
\end{bmatrix} = \frac{1}{k_0 k_1 k_2 \ldots k_n} \begin{bmatrix}
  1 & -\sigma_0 2^{-n} \\
  \sigma_0 2^{-n} & 1
\end{bmatrix} \ldots \begin{bmatrix}
  1 & -\sigma_0 2^{-0} \\
  \sigma_0 2^{-0} & 1
\end{bmatrix} \begin{bmatrix}
  x_0 \\
  y_0
\end{bmatrix}
\]

Figure 3. Chain of micro-rotations

5. Proposed Method

As discussed in the previous section, the selection of microrotations has a massive impact on the accuracy, hardware usage, and overall performance of the CORDIC algorithm. Accordingly, if an angle is decomposed with fewer microrotations, the computation’s speed increases. Moreover, if microrotations are selected carefully (more precision in decomposing the given angle), the accuracy of the calculations also increases. In this section, a new method for decomposing an angle to the required microrotations of the CORDIC algorithm is proposed.

5.1. Micro-rotation generation unit

In the Scale-Free CORDIC algorithm [15], after identifying the location of the most significant one (MSO) of the input angle, two 16-bits numbers are generated; the first number is “1” only in the location of the MSO-bit and the rest of its bits are zero, and the second number is “1” only in one higher value location and the rest of its bits are zero. If the location of MSO-bit in these two numbers was 14 and 15, \(\text{arctan}2^0\) and \(\text{arctan}2^{-1}\) replace the intended numbers, since the estimation error of using \(\tan2^{-i}\) with \(2^{-i}\) is relatively large in \(2^0\) and \(2^{-1}\) case. Afterward, generated numbers subtract from the input angle, and the smallest value is chosen. MSO-Locatin of this number is subtracted from the 15-degree angle, and the result is the desired shift amount for micro rotation. To find the least differences, a scheme similar to the Parallel Angle Recoding method [24] is used. However, in [24] all of the angles are stored in a ROM and subtracted from the input angle in parallel, and then all of the results are compared, but in the proposed scheme two nearest microrotations to the input angle are calculated dynamically, and then a comparison is performed. Figure 5 shows the microrotations generation unit. The input angle must be positive before entering the MSO location finder unit. Therefore, if the input angle was negative, a multiplexer delivers two’s complement of the input angle. After determining the MSO, generated numbers are added to and subtracted from the input angle. Results are compared, and the smallest number and the corresponding shift is determined. The sign bit of the input angle determines the desired number. The positive sign bit selects the result of the subtractors’ comparator, and the negative sign selects the result of the adders’ comparator. The final result is the desired shift.
Figure 5. Micro-rotation generation hardware

Figure 6-(a) illustrates the remaining angle after four iterations of the algorithm and Figure 6-(b) shows the error percentage in the range of [0, 45]. The remaining angles after the fourth iteration are insignificant, which indicates the accuracy of the proposed scheme.

(a) Remaining angles after four iterations of the proposed method
(b) Error percentage of decomposing the input angle with the proposed method

Figure 6. Accuracy of the proposed method

In general, the proposed method for finding the appropriate micro-rotation is similar to the method of the Fixed Angle [16]. However, the Fix Angle method is used for known odd angles in [0, 45], but the proposed method is applicable for any arbitrary angle in the range [0, 45]. Moreover, in [16] microrotations are predefined and stored in a ROM for each angle, but in the proposed method micro-rotations are generated dynamically for each angle.

5.2. Determining the sign of micro-rotations

The ability to use micro-rotation larger than the input angles increases the speed of decomposition. This is achieved by the possibility of input angle being negative after decomposition, a feature that is used to control adders/subtracters of the CORDIC unit at the next stage. The sign of micro-rotations which is determined by the sign bit of the input angle, use for selecting the add/subtract operation in adders/subtractors. Pseudocode 1 presents the algorithm for finding proper micro rotations and determining the sign for the adders.
In the Pseudocode, **MSOL** is the most significant one’s location, **mic1** and **mic2** are two generated numbers from **MSOL**, **Sign** is the sign bit sent to adders, and **r1** and **r2** are the differences between the input and micro-rotations.

### 5.3. Scale Factor Unit

The final result of the CORDIC module must be scaled. There are different techniques for applying the scale factor to the CORDIC module. It can be applied in another iteration in the algorithm, via bit by bit method in parallel compensation technique, or by exploiting iterations that are used for calculating the scale factor. In the conventional CORDIC algorithm, because all of the micro-rotations are predefined and used in the decomposition of the input angle, a constant coefficient can be calculated based on equation (18), which can be stored in the memory at the beginning of the process, or multiplied to the final result.

\[
k = \prod_{i} \frac{1}{\sqrt{1 + a_i^2 2^{-2i}}}
\]

(18)

In the proposed algorithm, the micro-rotations for the decomposition of an arbitrary angle are unknown; moreover, all of the micro-rotations are not used in the calculation. As a result, the scale factor would not be constant and must be calculated for each angle separately. The scale factor is equal to:

\[
\cos(\arctan(a_i)) = \frac{1}{\sqrt{1 + a_i^2}}
\]

(19)

\[
= 1 - \frac{a_i^2}{2} + \frac{3a_i^4}{8}
\]

(20)

Equation (19) can be estimated by the polynomial of equation (20). Due to hardware limitations, the second-order estimation is used in the proposed design. Because the denominators are powers of two, they can be implemented by adders and shifters. Also, contrary to the Fixed Angle [16] that uses different shifts for scale factors and micro-rotations which results in more complexity, the proposed method uses the same shifts for determining micro-rotations and calculating the scale factor. Furthermore, calculating the scale factor for different micro-rotations demonstrates that at large angles only the first two micro-rotations are essential and in micro-rotations with shifts equal or larger than three, the value of the scaling factor is approximately one. This causes the proposed algorithm to result in one clock faster. Figure 8 shows the value of the scale factor for all micro-rotations in the proposed algorithm.
The circuit depicted in Figure 9-(a) generates required shifts for the scale factor from the microrotations shifts, and Figure 9-(b) illustrates the unit of generating the scale factor for compensating the final results.

(a) Scale factor’s shift generator circuit

(b) Scale factor generating hardware

Figure 9. Shift generator and scale factor microarchitecture

(a) Proposed design’s block diagram

(b) Proposed design’s hardware view

Figure 10. Hardware Architecture of the proposed method


6. Results and Discussion

For evaluating the performance of the proposed method, factors of merits such as power consumption, utilized resources, latency, number of iterations, and maximum frequency must be compared with similar works. We used Xilinx ISE Design Suite and Matlab for our simulations. Results are compared to the previous designs that aim to reduce the number of iterations. Conventional CORDIC algorithm needs sixteen clocks to produce the desired results with sixteen bits of precision. Scale Free [15] and Fixed Angle [16] need seven clocks to calculate the answer with similar precision. The Fixed Angle method [16] needs four clocks for the calculation of the CORDIC and three more clocks for scaling the result. Our method computes the result in four clocks similar to [16] and scales it in two clocks. Consequently, in the case of the need for more accuracy, one extra clock can be used to increase the accuracy.

Table 1 summarizes the number of occupied slices, maximum frequency, and worst-case iteration of the proposed method, [15], and [16]. Results show that considering the implemented platform and other design restrictions, the proposed algorithm outperforms previous works in terms of resource utilization, frequency, and the number of iteration. The Fixed Angle method [16] is proposed for a limited range of angles stored in a ROM. Therefore the need for complex calculations is eliminated. On the other hand, because in the proposed method micro-rotations generate dynamically, it is capable of calculating arbitrary angles in the range of [0, 45]. Table 2 shows the dynamic, static, and total power derived from the simulation results of the proposed scheme in Xpower software on the Xilinx Spartan 6 FPGA (xc6slx9).

<table>
<thead>
<tr>
<th>Design</th>
<th>Device</th>
<th># of Slices (A)</th>
<th>Max Freq (B)</th>
<th># of Iterations (C)</th>
<th>(A*C/B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>Spartan 2E</td>
<td>231</td>
<td>58.4</td>
<td>7</td>
<td>27.7</td>
</tr>
<tr>
<td>[16]</td>
<td>Virtex-4</td>
<td>279</td>
<td>196.8</td>
<td>7</td>
<td>9.92</td>
</tr>
<tr>
<td>Proposed method</td>
<td>Spartan 6</td>
<td>124</td>
<td>125.6</td>
<td>6</td>
<td>5.92</td>
</tr>
</tbody>
</table>

Table 2. Power consumption of [15], [16] and the proposed design

<table>
<thead>
<tr>
<th>Design</th>
<th>Dynamic (mW)</th>
<th>Static (mW)</th>
<th>Total (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>85</td>
<td>46</td>
<td>131</td>
</tr>
<tr>
<td>[16]</td>
<td>53</td>
<td>45</td>
<td>98</td>
</tr>
<tr>
<td>Proposed Design</td>
<td>102</td>
<td>46</td>
<td>148</td>
</tr>
</tbody>
</table>

Figure 11 depicts the consumed power at different frequencies, from zero to the maximum frequency. Also, Figure 12 illustrates Sin and Cosine calculation’s error in the interval [0 : 45], with the maximum error of 2.949% for Sin and 0.17% for Cosine.
7. Conclusions

In this paper, a new method for reducing the iterations of the CORDIC algorithm is proposed which results in a reduction of overall delay and improvement of the speed of calculation. This method reduces the number of required iterations for converging the calculation to the final result by determining the most significant one’s location (MSO-location). Dynamic generation of microrotations eliminates the need for the memory, and unlike previous works, it can be used for any angle in the range of [0, 45], and it is not limited to certain angles. Moreover, the proposed algorithm uses the same microrotations generated during decomposing the input angel, for producing scale factor, which eliminates the need for an extra ROM. In our design, the scale factor changes according to the input angle. Therefore, a new method for calculating the scale factor in two iterations is presented, which only uses shift and add operations. Compared to similar works that attempt to reduce the number of iterations, this work shows improvement regarding resource utilization and throughput while keeping the accuracy reasonable.

References


