HCORDIC: A High-Performance Cordic Algorithm

by

Ahmad Nour Al-Islam Al-Sawi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

January, 1997

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

HCORDIC: A HIGH-PERFORMANCE CORDIC ALGORITHM

BY

Ahmad Nour Al-Islam Al-Sawi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
COMPUTER ENGINEERING

JANUARY, 1997

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by **Ahmad Nour Al-Islam Al-Sawi** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** in **COMPUTER ENGINEERING**.

THESIS COMMITTEE

Dr. Fayez El-Guibaly (Chairman)

Dr. Alaaeldin Amin (Co-Chairman)

Dr. Mohamed Osman (Member)

Dr. Mostafa Abd-El-Barr (Member)

Department Chairman

Dean, College of Graduate Studies

Date: 5 Apr. 1997

كلية الدراسات العليا
COLLEGE OF GRADUATE STUDIES

To My Beloved Mother and Father
and to Those Who Share Their Care and Concern

بسم الله الرحمن الرحيم

الرَّحْمَنُ (١) عَلَّمَ الْقُرْآنَ (٢) خَلَقَ الْإِنسَانَ (٣) عَلَّمَهُ الْبَيَانَ (٤) الشَّمْسُ وَالْقَمَرُ بِحُسْبَانٍ (٥)
وَالنَّجْمُ وَالشَّجَرُ يَسْجُدَانِ (٦) وَالسَّمَاءَ رَفَعَهَا وَوَضَعَ الْمِيزَانَ (٧) أَلَّا تَطْغَوْا فِي الْمِيزَانِ (٨)
وَأَقِيمُوا الْوَزْنَ بِالْقِسْطِ وَلَا تُخْسِرُوا الْمِيزَانَ (٩) وَالْأَرْضَ وَضَعَهَا لِلْأَنَامِ (١٠) فِيهَا فَاكِهَةٌ وَالنَّخْلُ
ذَاتُ الْأَكْمَامِ (١١) وَالْحَبُّ ذُو الْعَصْفِ وَالرَّيْحَانُ (١٢) فَبِأَيِّ آلَاءِ رَبِّكُمَا تُكَذِّبَانِ (١٣)

سورة الرحمن (١-١٣)

In the name of Allah, the Compassionate, the Merciful

The Most Gracious! It is He Who has taught the Qur'an. He has created man: He has taught him an intelligent speech. The sun and the moon follow courses (exactly) computed; And the herbs and the trees-both (alike) bow in adoration. And the Firmament has He raised high, and He has set up the Balance (of Justice), In order that ye may not transgress (due) balance. So establish weight with justice and fall not short in the balance. It is He Who has spread out the earth for (His) creatures: Therein is fruit and date-palms, producing spathes (enclosing dates); Also corn, with (its) leaves and stalk for fodder, and sweet-smelling plants. *Then which of the favours of your Lord will ye deny?*

AR-RAHMAN (1-13)

Acknowledgment

First and foremost, all praise is due to Allah, the Almighty, Who gave me the opportunity, courage, and patience to carry out this work. I seek His mercy, favor, and forgiveness. I ask Him to accept my little effort. May Allah guide us and the whole of humanity to the right path *(Amen)*.

Acknowledgment is due to *King Fahd University of Petroleum & Minerals* for its support of this research.

I would like to express my deep appreciation to *Professor Fayez El-Guibaly*, thesis advisor, for his patient guidance and his generous support for this research.

I would also like to express my gratitude to my other committee members, *Dr. Alaaeldin Amin*, thesis co-advisor, *Professor Mohamed Osman* and *Professor Mostafa Abd-El-Barr* for their valuable advice and helpful remarks.

Special thanks go to *Dr. Khalid Al-Tawil*, Department Chairman, for his kindness and support, to the teaching, research, and administration members of the *College of Computer Science and Engineering*, to *Abdullah Rayhan* for his great assistance during the thesis period, and to *Ramadan Fan, Adel Ahmed*, my roommates, and *Husam Abu-Haimed, Talal Al-Kharoubi, Adel Ibrahim, Abdul-Rahman Ibrahim,* and many others, for the memorable days we shared together.

Finally, I would like to express my sincere gratitude to my parents whose constant encouragement has made the completion of this work possible.

Contents

List of Tables

List of Figures

Thesis Abstract

Name: Ahmad Nour Al-Islam Ayoub Al-Sawi
Title: HCORDIC: A High-Performance CORDIC Algorithm
Major Field: Computer Engineering
Date of Degree: January, 1997

The COordinate Rotation DIgital Computer (CORDIC) was introduced in 1959. It is a single unified algorithm for calculating many elementary functions including trigonometric, hyperbolic, logarithmic and exponential functions, multiplication, division and square root. It is also useful in many other applications, such as digital signal processing, robotics, and matrix arithmetic. CORDIC is inherently plagued with many problems that have prevented its widespread acceptance. Thus, for the last 40 years, many researchers have attempted unsuccessfully to remedy these problems.

In this thesis, the CORDIC algorithm is reviewed, including its theory, applications, problems, and earlier solutions. A new High-Performance CORDIC algorithm, HCORDIC, is proposed. The new algorithm has none of the drawbacks of the original CORDIC. The modifications introduced in HCORDIC allow it to make use of new advances in VLSI and computer architecture. Extensive numeric simulations are performed that corroborate the superiority of HCORDIC when compared with the standard CORDIC. Finally, hardware implementation issues of HCORDIC are discussed and VHDL simulation is conducted.

Keywords: CORDIC algorithm, elementary functions evaluation, fast division, computer arithmetic, VLSI design, parallel multipliers, digital signal processing

Master of Science Degree
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
January, 1997

خلاصة الرسالة

الاسم: أحمد نورالإسلام أيوب السوي

عنوان الرسالة: حاسب رقمي متقدم لدوران الإحداثيات **(HCORDIC)**

التخصص: هندسة الحاسب الآلي

تاريخ التخرج: شعبان ١٤١٧هـ

ظهرت خوارزمية **الحاسب الرقمي لدوران الإحداثيات** *CORDIC* في عام ١٩٥٩، والتي بإمكانها القيام بكل من عملية الضرب والقسمة وحل الدوال المثلثية والقطوع الزائدة واللوغاريتمات والجذور التربيعية. ولذلك فإن CORDIC يستخدم في العديد من التطبيقات العملية مثل الأذرعة المفصلية الآلية ومعالجة الإشارات الرقمية وحساب المصفوفات. وبالرغم من إيجابيات CORDIC الكثيرة، هناك سلبيات متأصلة، ولهذا حاول الباحثون خلال العقود الأربعة السابقة إيجاد حلول لهذه السلبيات، لكن جميعها كانت حلولاً هامشية وليست جذرية. ومن ثم جاءت هذه الأطروحة بمثابة الجواب الكافي لمن سأل عن الدواء الشافي.

تبدأ هذه الأطروحة بمراجعة الجانب النظري والتطبيقي لخوارزمية CORDIC، مع نقد وتحليل لسلبياتها المتأصلة. ثم تعرض جانباً من الحلول السابقة وما لها وما عليها. وبعد ذلك تقدم التعديلات المقترحة لتحسين CORDIC والتي تقتلع المشاكل المتأصلة من جذورها. وتتميز هذه التعديلات بالحفاظ على عراقة الخوارزمية القديمة مع ملاءمة التقدم الحاصل في صناعة الدوائر ذات التكاملية الفائقة VLSI. وقد سميت الخوارزمية المقترحة **بالحاسب الرقمي المتقدم لدوران الإحداثيات** *HCORDIC*، وهذه التسمية جاءت بناءً على نتائج محاكاة الخوارزمية الجديدة والتي تؤكد تفوقها على نظيرتها القديمة. وقد تضمنت الأطروحة مناقشة بعض المسائل المتعلقة بتصميم معالج HCORDIC على الدوائر ووضع نموذج له باستخدام لغة وصف الدوائر VHDL.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران – المملكة العربية السعودية
شعبان ١٤١٧هـ

Chapter 1

Introduction

The COordinate Rotation DIgital Computer (CORDIC) algorithm was introduced by Volder in 1959 [1], and later generalized and unified by Walther in 1971 [2]. The unified algorithm computes trigonometric, hyperbolic, exponential and logarithmic functions, as well as, multiplication, division and square root. CORDIC is an attractive algorithm because it can compute most mathematical functions using basic operations of the form $a \pm b \times 2^{-i}$, using simple hardware.

At the time of CORDIC introduction, multipliers were very expensive. Hence, CORDIC was an attractive way to evaluate elementary functions instead of using polynomial techniques. It has been implemented in pocket calculators like Hewlett Packard's HP 35, and in arithmetic co-processors like the Intel 8087.

1.1 History

CORDIC was designed to be a special-purpose digital computer for real-time airborne computation. It was proposed by Volder for solving trigonometric relationships involved in plane coordinate rotation and conversion from rectangular to polar coordinates. At that time, compared to the analog devices, the basic operation of CORDIC can be functionally described as the digital equivalent of an analog resolver [1].

A prototype digital computer, CORDIC I, was built at Convair[1], Fort Worth. It has a special serial arithmetic unit consisting of three shift registers, three adders, and special interconnections.

Originally, CORDIC was programmed to solve either set of the following equations:

$$y' = K(y\cos\theta + x\sin\theta) \tag{1.1}$$

$$x' = K(x\cos\theta - y\sin\theta) \tag{1.2}$$

or

$$R = K\sqrt{x^2 + y^2} \tag{1.3}$$

$$\theta = \tan^{-1} y/x \tag{1.4}$$

where K is a constant.

[1]A division of General Dynamics Corp. Fort Worth, Texas.

The first set of equations is solved in a CORDIC operation called *Rotation*, and the second set is solved in *Vectoring* operation. In rotation operation, the coordinate components of a two-dimensional vector and a rotation angle are given; and the coordinate components of the vector after rotation are computed. In vectoring operation, the coordinate components of a vector are given; and the magnitude and the angle of the original vector with the x-axis are computed.

Essentially, the basic computing method used in both rotation and vectoring operations is a step-by-step sequence of micro rotations which result in an overall rotation by a given angle as in the rotation operation, or result in zeroing the final angle of the vector as in the vectoring operation. These micro rotation angles are chosen such that the computations needed are only shift and add operations. The micro rotations are performed in iterations, where the number of iterations required is equal to the number of significant bits that represent the rotated components.

In 1972, Walther generalized CORDIC to be used in other coordinate systems. This means that instead of rotating a vector along a circular curve, the vector can be rotated along a line or a hyperbola, as shown in Figure 1.1, where ϕ is the "angle" and R_1, R_0, and R_{-1} are the "radii" of the vector $P = (x, y)$ in circular, linear, and hyperbolic coordinate systems, respectively. The generalized radius can be geometrically described as the distance from the origin to the intersection point of the coordinate curve (circle, line, or hyperbola) with the x-axis, while the generalized angle is equal to twice the area enclosed by the vector, the x-axis, and the coordinate

curve, divided by the radius squared, as in the following equation:

$$\phi = \frac{2 \text{ Enclosed Area}}{R^2} \tag{1.5}$$

Walther came up with a set of unified equations that describes the coordinate components of the rotated vector. These equations are parameterized in terms of the coordinate system. Hence, more elementary functions can be computed using CORDIC, such as division, multiplication, square root, and hyperbolic functions.

$$R = (x^2 + my^2)^{1/2} \tag{1.6}$$

$$\phi = m^{-1/2} \tan^{-1}(m^{1/2} y/x) \tag{1.7}$$

$$m = \begin{cases} 1 & \text{a circle} \\ 0 & \text{a line} \\ -1 & \text{a hyperbola} \end{cases} \tag{1.8}$$

1.2 Advantages of CORDIC

CORDIC is a single algorithm that is capable of computing a wide range of elementary functions using simple shift and add operations. Beside being used for computing elementary mathematical functions, it has been applied in many digital signal processing applications, such as speech synthesis, fast Fourier transform (FFT), and digital filtering. It is also used in matrix arithmetic, singular-value decomposition (SVD), matrix triangularization and QR decomposition. CORDIC has been extensively used in robotics applications, such as inverse kinematics.

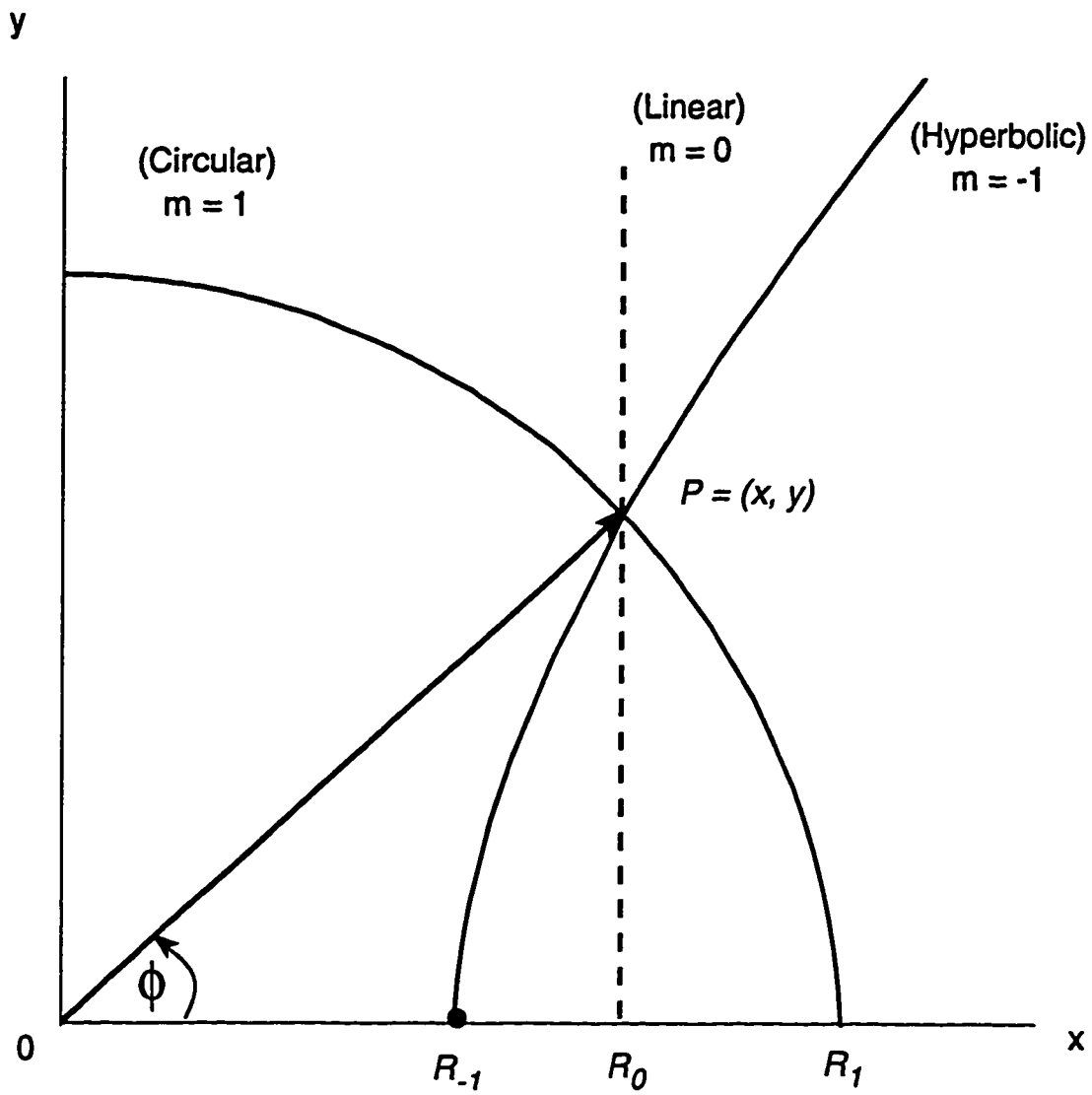

Figure 1.1: Coordinate systems of CORDIC.

Because CORDIC can compute many useful functions, a number of recent applications requires CORDIC as a basic processor. Some of those applications are video compression, video conferencing [3], [4], fast cable modems [5], [6], and co-processor of super computers.

We can summarize the main advantages of the CORDIC algorithm in two points. CORDIC is a single algorithm capable for computing a wide range of arithmetic functions. It can perform the required computations without using a multiplier, which was very expensive at that time.

1.3 Overview of Previous Research

We can notice that CORDIC is a very versatile algorithm that has been used in many applications since its introduction. It is a very elegant algorithm that has a complicated theory from the inside but appears to users as a handy tool for various applications. However, the algorithm has some drawbacks and constraints that can be improved or eliminated. Therefore, a lot of research work has been done on that.

The major research directions of CORDIC can be classified into the following.

- Eliminating constraints: CORDIC algorithm has some inherited constraints. For example, its input arguments should satisfy some conditions, which is known as convergence range. Hence, many researchers worked on expanding this limited range [2], [7], [8], [9].

- Speeding up: CORDIC is a slow iterative algorithm. Hence, researchers attempted to reduce the number of iterations required, propose new designs by adding more hardware to achieve speed-up [10], [11], resort to redundant number systems [12], [13], [14], or exploit parallel or pipelined architectures [15], [16].

- Investigating new applications: As mentioned earlier, CORDIC has many applications. Researchers are continuously looking for new areas for applying CORDIC [16], [17], [18], [19]. Similarly, researchers of other fields often discover that CORDIC is their solution [20]. Hence, many new applications are being added in recent years.

- Proposing new hardware designs: With the advances of VLSI technology, more advanced and economical hardware options are available to designers. Hence, several alternate implementations for the standard CORDIC algorithm were reported [7], [21], [22]. In addition, several CORDIC processors have been geared toward a specific application, such as in digital signal processing [16], [23].

In this thesis, we present a new High-Performance CORDIC algorithm that improves CORDIC but preserves its general framework. The original standard CORDIC was a bit-serial algorithm, as opposed to the new proposed algorithm which is bit-parallel. The standard CORDIC operates on one bit of the argument

per iteration and performs a prespecified micro rotation; while the new algorithm operates on several bits per iteration and performs a micro rotation which is adapted to the argument at each iteration. Hence, the fixed set of predefined micro rotations is replaced by an adaptive one. The size of the adaptive set depends on the number of bits the algorithm operates on in each iteration.

1.4 Thesis Motivations and Objectives

Using shift and add operations, the main advantage of CORDIC in 1959, has turned into a disadvantage after the great advances in digital technologies. This factor is the major motivation for this thesis.

The standard CORDIC is originally a sequential, non-adaptive algorithm which does not take advantage of the advances in computer architecture and VLSI technology. In this thesis, we would like to prove that without compromising the elegance of the standard CORDIC, its theory and concepts, we can achieve considerable improvements by taking advantage of new technologies.

The objective of this thesis is to:

1. Study the standard CORDIC algorithm, its theory and advantages.

2. Analyze its drawbacks and review previous proposed solutions.

3. Master the proposed High-Performance CORDIC (HCORDIC) algorithm.

4. Simulate the HCORDIC algorithm.

5. Perform system design for an HCORDIC processor and model it in VHDL.

1.5 Thesis Organization

This thesis is organized as follows. In Chapter 2, the standard CORDIC algorithm is reviewed, and its main drawbacks are highlighted. In Chapter 3, earlier solutions for those drawbacks are reviewed. In Chapter 4, the newly proposed CORDIC algorithm, HCORDIC, is introduced. The main concepts of HCORDIC are explained as well as its advantages. In Chapter 5, numerical software simulation results of HCORDIC are presented and compared to the standard CORDIC. In Chapter 6, hardware implementation issues are discussed, and a VHDL model is presented. Finally, in Chapter 7, the main contributions of this thesis are summarized and possible future research directions are outlined.

Chapter 2

The Standard CORDIC

CORDIC works on two-dimensional vectors by applying either a rotation or vectoring operation. In the rotation operation, the inputs are the x and y components of the vector and the desired rotation angle. The outputs are the resulting vector components after rotation through the desired rotation angle. In the vectoring operation, the inputs are the x and y components of the vector. The outputs are the magnitude of the vector and its angle with the positive x-axis. These two operations can be performed in one of three modes, namely, circular, linear, and hyperbolic. By performing these two basic operations in various coordinate systems, CORDIC is able to generate several elementary functions.

Figure 2.1 shows a block diagram of CORDIC which summarizes the input, output and the internal variables used. The CORDIC algorithm accepts three data inputs (x, y, z) which correspond to the x and y components of a vector and its

Table 2.1: Variables used in CORDIC.

Variable	Comments
i	iteration index
x_i	x component of the vector at iteration i
y_i	y component of the vector at iteration i
z_i	accumulated angle at iteration i
op	operation control
m	mode control
μ_i	sign of the rotation angle at iteration i
θ_i	rotation angle at iteration i
δ_i	a variable that depends on θ_i

angle with the positive x-axis. CORDIC also requires two control inputs for the type of operation (op) and mode (m). CORDIC is essentially an iterative algorithm that deals with other internal variables within each iteration. The internal variables are listed in Table 2.1. CORDIC computations are based on the decomposition of the rotation angle into the sum of a set of predefined elementary angles, such that rotation through each of these angles can be done with simple shift and add operations. The accuracy of CORDIC is determined by several factors, such as machine word length, the number of iterations, and the step size in each iteration. In n iterations CORDIC can achieve an accuracy of n bits [24].

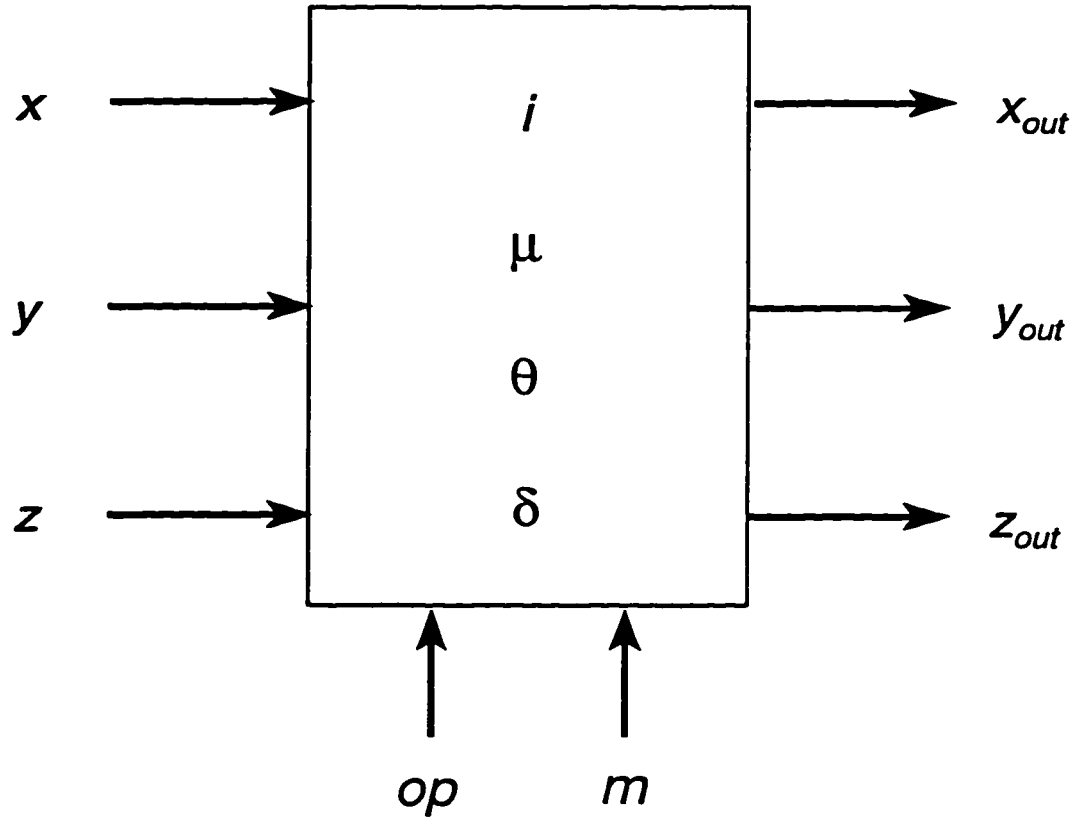

Figure 2.1: Block diagram description of CORDIC.

2.1 The Unified CORDIC Algorithm

Walther [2] unified the different modes of CORDIC into a single set of equations by generalizing the concept of the radius R of a vector, and its angle ϕ with the positive x-axis according to the following equations:

$$R = (x^2 + my^2)^{1/2} \tag{2.1}$$

$$\phi = m^{-1/2} \tan^{-1}(m^{1/2} y/x) \tag{2.2}$$

$$m = \begin{cases} 1 & \text{a circle} \\ 0 & \text{a line} \\ -1 & \text{a hyperbola} \end{cases} \tag{2.3}$$

The generalized radius can be geometrically described as the distance from the origin to the intersection point of the coordinate curve (circle, line, or hyperbola) with the x-axis, while the generalized angle is equal to twice the area enclosed by the vector, the x-axis, and the coordinate curve, divided by the radius squared, as in the following equation:

$$\phi = \frac{2 \text{ Enclosed Area}}{R^2} \tag{2.4}$$

2.1.1 Circular Mode ($m = 1$)

In the circular mode, the end point of the vector moves on a circle described by

$$x^2 + y^2 = R^2 \tag{2.5}$$

as shown in Figure 2.2. The x and y components of the vector are given by the parameterized equations:

$$x = R\cos\phi \tag{2.6}$$

$$y = R\sin\phi \tag{2.7}$$

Rotating the vector by an angle θ, the new x and y values become,

$$x' = R\cos(\phi+\theta) = x\cos\theta - y\sin\theta \tag{2.8}$$

$$y' = R\sin(\phi+\theta) = y\cos\theta + x\sin\theta \tag{2.9}$$

To calculate the rotated components, four multiplications are required. If we divide Equations 2.8 and 2.9 by $\cos\theta$ we get,

$$x'' = x - y\tan\theta \tag{2.10}$$

$$y'' = y + x\tan\theta \tag{2.11}$$

where $x'' = x'/\cos\theta$ and $y'' = y'/\cos\theta$.

By choosing $\tan\theta = 2^{-i}$, the rotated vector components are obtained through simple shift and add operations. The above two equations describe the basic CORDIC equations in the circular mode.

2.1.2 Linear Mode ($m = 0$)

In the linear mode, the end point of the vector moves on a line described by

$$x = R \tag{2.12}$$

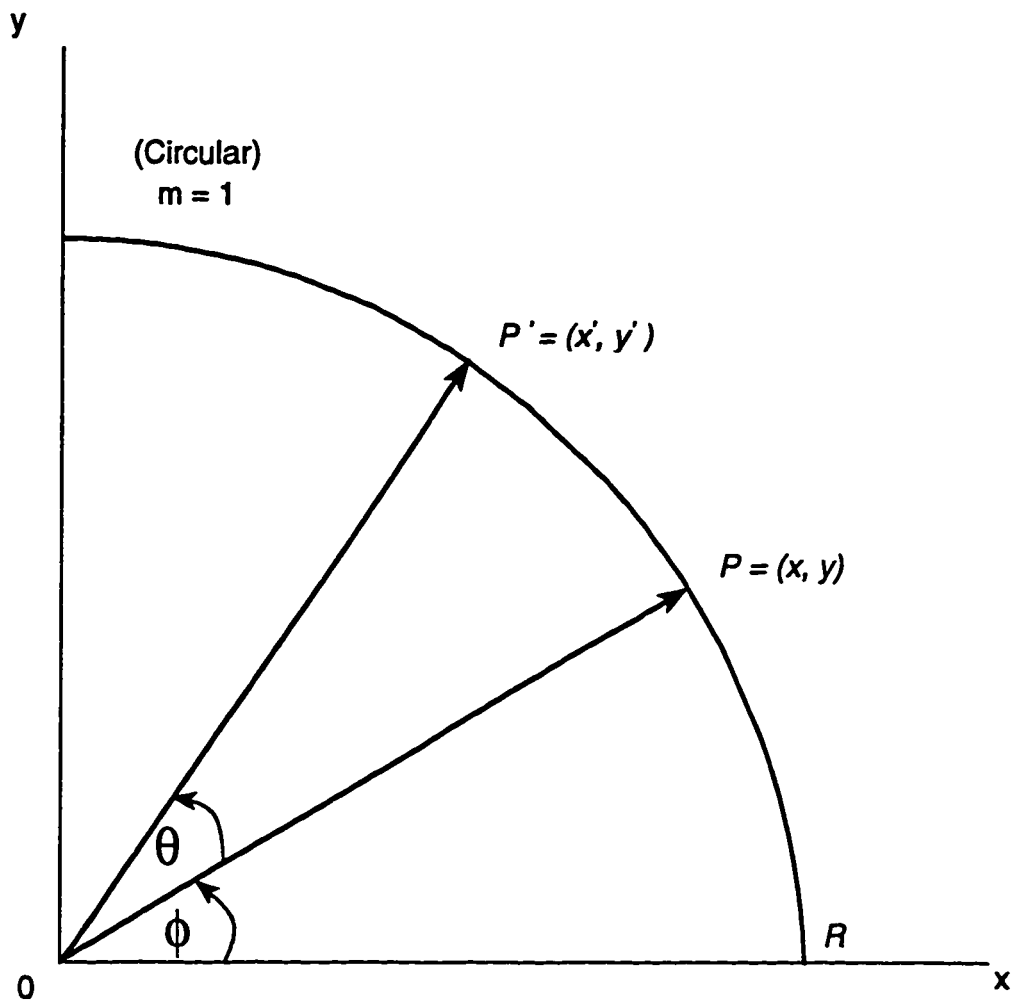

Figure 2.2: Rotating a vector by θ in the circular mode, where ϕ is the initial angle and R is the radius of the vector $P = (x, y)$.

as shown in Figure 2.3. The x and y components of the vector are given by the parameterized equations:

$$x = R \tag{2.13}$$

$$y = \phi R \tag{2.14}$$

where $\phi = y/x$, notice that the definition of ϕ in the linear mode is the ratio of y/x. Some detailed derivations are available in Appendix A. Rotating the vector by an "angle" θ in the linear mode is equivalent to changing the value of ϕ into $\phi+\theta$. The new "rotated" x and y values now become,

$$x' = R \tag{2.15}$$

$$y' = (\phi+\theta)R = y + \theta x \tag{2.16}$$

By choosing $\theta = 2^{-i}$, the rotated vector components are obtained through simple shift and add operations. The above two equations describe the basic CORDIC equations in the linear mode.

2.1.3 Hyperbolic Mode ($m = -1$)

In this mode, the end point of the vector moves on a hyperbola described by

$$x^2 - y^2 = R^2 \tag{2.17}$$

as shown in Figure 2.4. Similar to the circular mode, the x and y components of

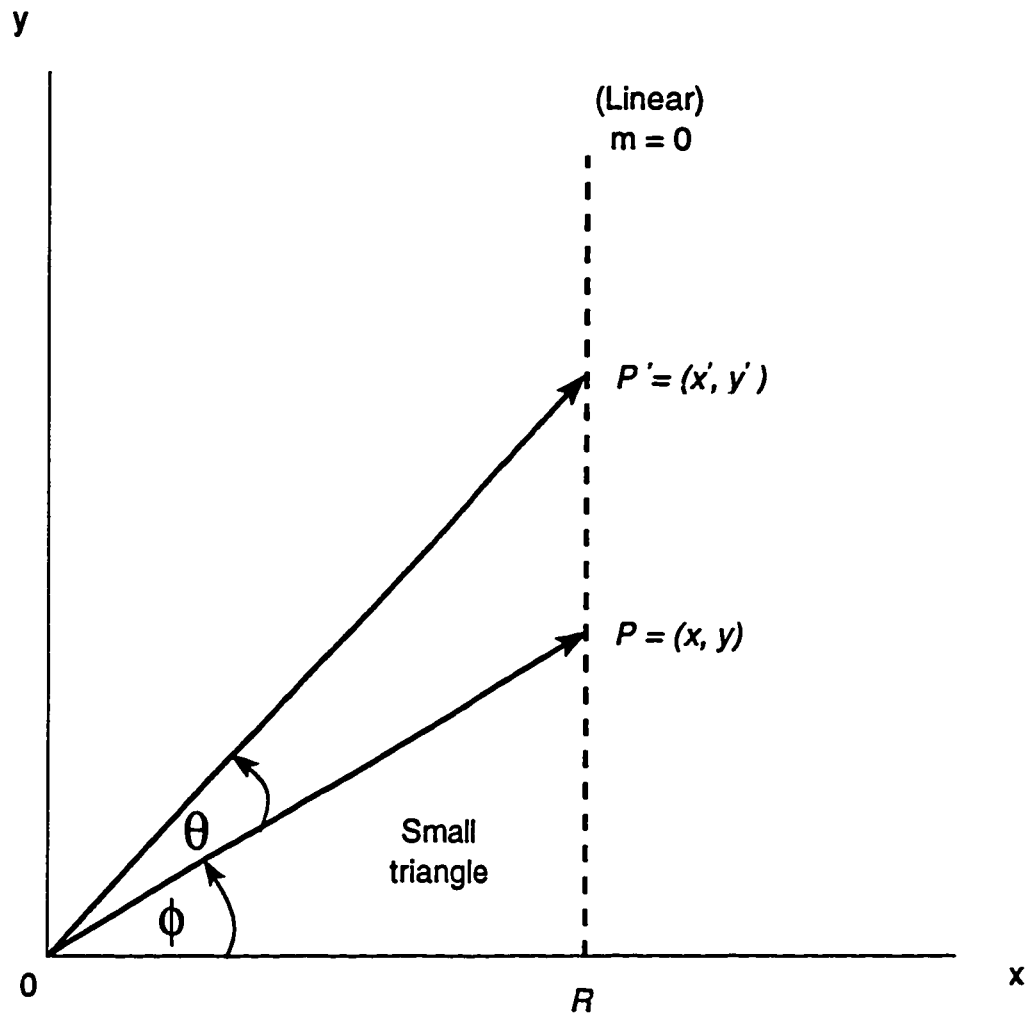

Figure 2.3: Rotating a vector by θ in the linear mode, where ϕ is the initial "angle" and R is the "radius" of the vector $P = (x, y)$.

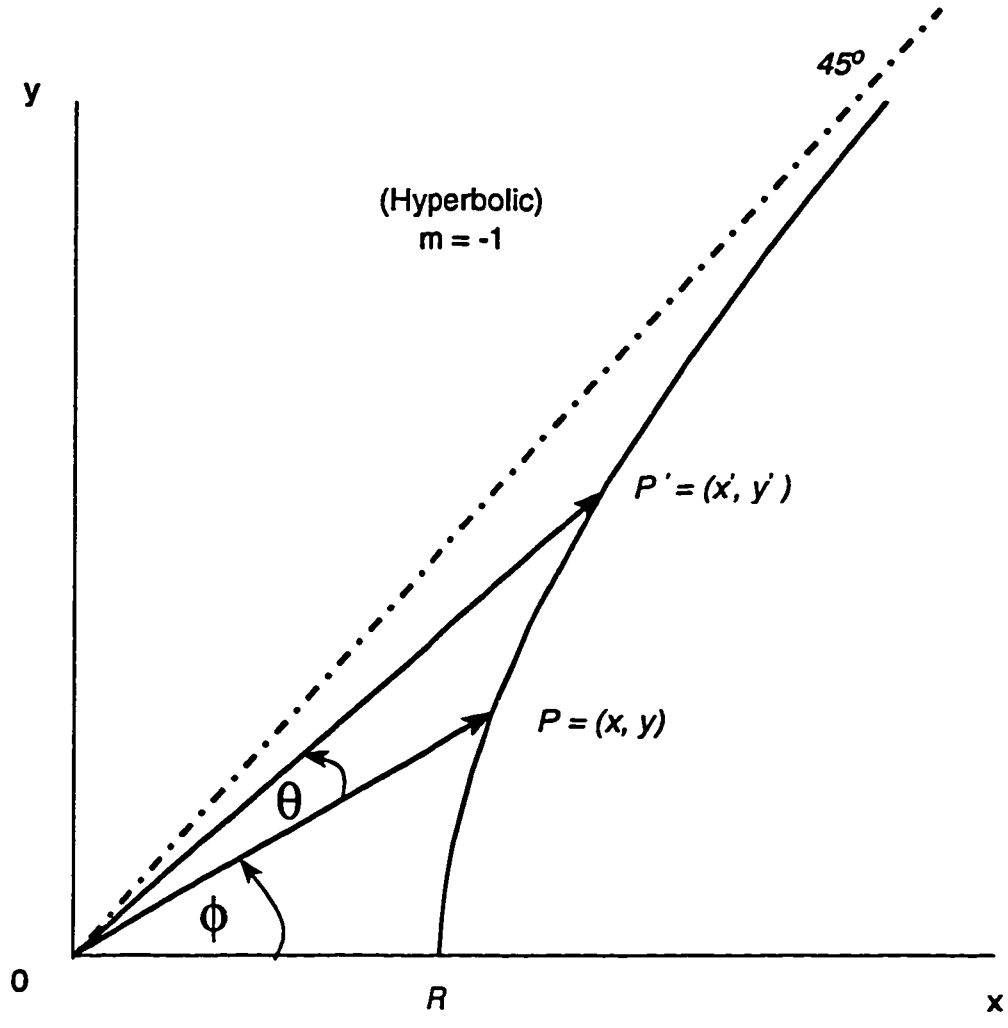

Figure 2.4: Rotating a vector by θ in the hyperbolic mode, where ϕ is the initial "angle" and R is the "radius" of the vector $P = (x, y)$.

the vector are given by the parameterized equations:

$$x = R\cosh\phi \tag{2.18}$$

$$y = R\sinh\phi \tag{2.19}$$

Rotating the vector by an "angle" θ in the hyperbolic mode is equivalent to changing the value of ϕ into $\phi + \theta$. The new "rotated" x and y values now become,

$$x' = R\cosh(\phi+\theta) = x\cosh\theta + y\sinh\theta \tag{2.20}$$

$$y' = R\sinh(\phi+\theta) = y\cosh\theta + x\sinh\theta \tag{2.21}$$

To calculate the rotated components, four multiplications are required. If we divide Equations 2.20 and 2.21 by $\cosh\theta$ we get,

$$x'' = x + y\tanh\theta \tag{2.22}$$

$$y'' = y + x\tanh\theta \tag{2.23}$$

where $x'' = x'/\cosh\theta$ and $y'' = y'/\cosh\theta$.

By choosing $\tanh\theta = 2^{-i}$, the rotated vector components are obtained through simple shift and add operations. The above two equations describe the basic CORDIC equations in the hyperbolic mode.

2.2 CORDIC Iterations

Independent of the CORDIC mode, the rotation angle can be decomposed into a set of elementary angles, such that the CORDIC iterations consist of shift and

Table 2.2: Guidelines for choosing the direction of rotation.

μ_i value	Rotation	Vectoring
+1	$z_i < 0$	$y_i \geq 0$
−1	$z_i \geq 0$	$y_i < 0$

add operations only. Thus at iteration i, the components x and y will be updated according to the following equations:

$$x_{i+1} = x_i + m\mu_i y_i \delta_i \tag{2.24}$$

$$y_{i+1} = y_i - \mu_i x_i \delta_i \tag{2.25}$$

$$z_{i+1} = z_i + \mu_i \theta_i \tag{2.26}$$

where,

$$i = 0, 1, \cdots, n-1 \qquad n \text{ is the number of bits in the mantissa.} \tag{2.27}$$

$$\delta_i = 2^{-i} \tag{2.28}$$

$$\theta_i = \begin{cases} \tan^{-1} \delta_i & m = 1 \\ \delta_i & m = 0 \\ \tanh^{-1} \delta_i & m = -1 \end{cases} \tag{2.29}$$

$$\mu_i = \begin{cases} 1 & \text{clockwise vector rotation} \\ -1 & \text{counterclockwise vector rotation} \end{cases} \tag{2.30}$$

The direction of rotation is determined by μ_i, where the choice of μ_i depends on the operation of the CORDIC according to Table 2.2. After n iterations, the total

change in angle is the sum of the incremental changes:

$$\theta = \sum_{i=0}^{n-1} \mu_i \theta_i \tag{2.31}$$

where[1]

$$\theta_i = m^{-1/2} \tan^{-1}(m^{1/2}\delta_i) \tag{2.32}$$

The variable z_i is introduced to accumulate the angle changes during iteration i as shown in Equation 2.26.

2.3 Scale Factor

We notice that in both the circular and hyperbolic modes, the resulting x and y values are scaled by $k_1 = 1/\cos\theta$ and $k_{-1} = 1/\cosh\theta$, respectively. To get the true final values of x and y, we need to descale the rotated values by the scale factor K_m as follows:

$$x_{true} = x'/K_m \tag{2.33}$$

$$y_{true} = y'/K_m \tag{2.34}$$

where $m = \pm 1$, and K_m is described by Equation 2.35

The descaling operation can be done using CORDIC as well. For n iterations, the resulting scale factor is the product of the scale factor in each iteration i, as

[1]Derivation of θ in each mode can be obtained using Identities A.9 and A.11 in Appendix A.

Table 2.3: Scale factor K_m.

m	K_m
1	1.646760
0	1
−1	0.828159

follows:

$$K_m = \prod_{i=0}^{n-1} k_i \tag{2.35}$$

where

$$k_i = (1 + m\delta_i^2)^{1/2} \tag{2.36}$$

K_m depends on the mode m, and it can be precalculated since the values of δ_i are known and fixed. The values of K_m for the three modes are shown in Table 2.3. The number of decimal places to represent K_m should be at least $\lceil \log_2^n \rceil$.

After n iterations, we can summarize all three modes equations in the following parameterized equations [2]:

$$x_n = K_m[x_0 \cos(\theta m^{1/2}) + y_0 m^{1/2} \sin(\theta m^{1/2})] \tag{2.37}$$

$$y_n = K_m[y_0 \cos(\theta m^{1/2}) - x_0 m^{-1/2} \sin(\theta m^{1/2})] \tag{2.38}$$

$$z_n = z_0 + \theta \tag{2.39}$$

where x_0, y_0 and z_0 are the initial input values.

Table 2.4: CORDIC processor output in the rotation operation ($z \rightarrow 0$).

m	x_{out}	y_{out}
1	$K_1 (x_0 \cos z_0 - y_0 \sin z_0)$	$K_1 (y_0 \cos z_0 + x_0 \sin z_0)$
0	x_0	$y_0 + x_0 z_0$
-1	$K_{-1} (x_0 \cosh z_0 + y_0 \sinh z_0)$	$K_{-1} (y_0 \cosh z_0 + x_0 \sinh z_0)$

The block diagram in Figure 2.5 shows the input-output functions of CORDIC. In the left column of the blocks, z is forced to zero, making $z_n \rightarrow 0$. While in the right column, y is forced to zero, making $y_n \rightarrow 0$. These functions are also summarized in Table 2.4 and Table 2.5.

For example, to evaluate the sine function of an angle θ, we set the inputs as follows:

$$x_{in} = 1, \quad y_{in} = 0, \quad z_{in} = \theta, \quad m = 1, \quad op = \text{rotation}$$

The value that results in y_{out} will be equal to $K_1 \sin\theta$. To descale it, we have to perform another CORDIC operation with the following inputs:

$$x_{in} = K_1, \quad y_{in} = K_1 \sin\theta, \quad z_{in} = 0, \quad m = 0, \quad op = \text{vectoring}$$

Now, the value that results in z_{out} will be equal to $\sin\theta$.

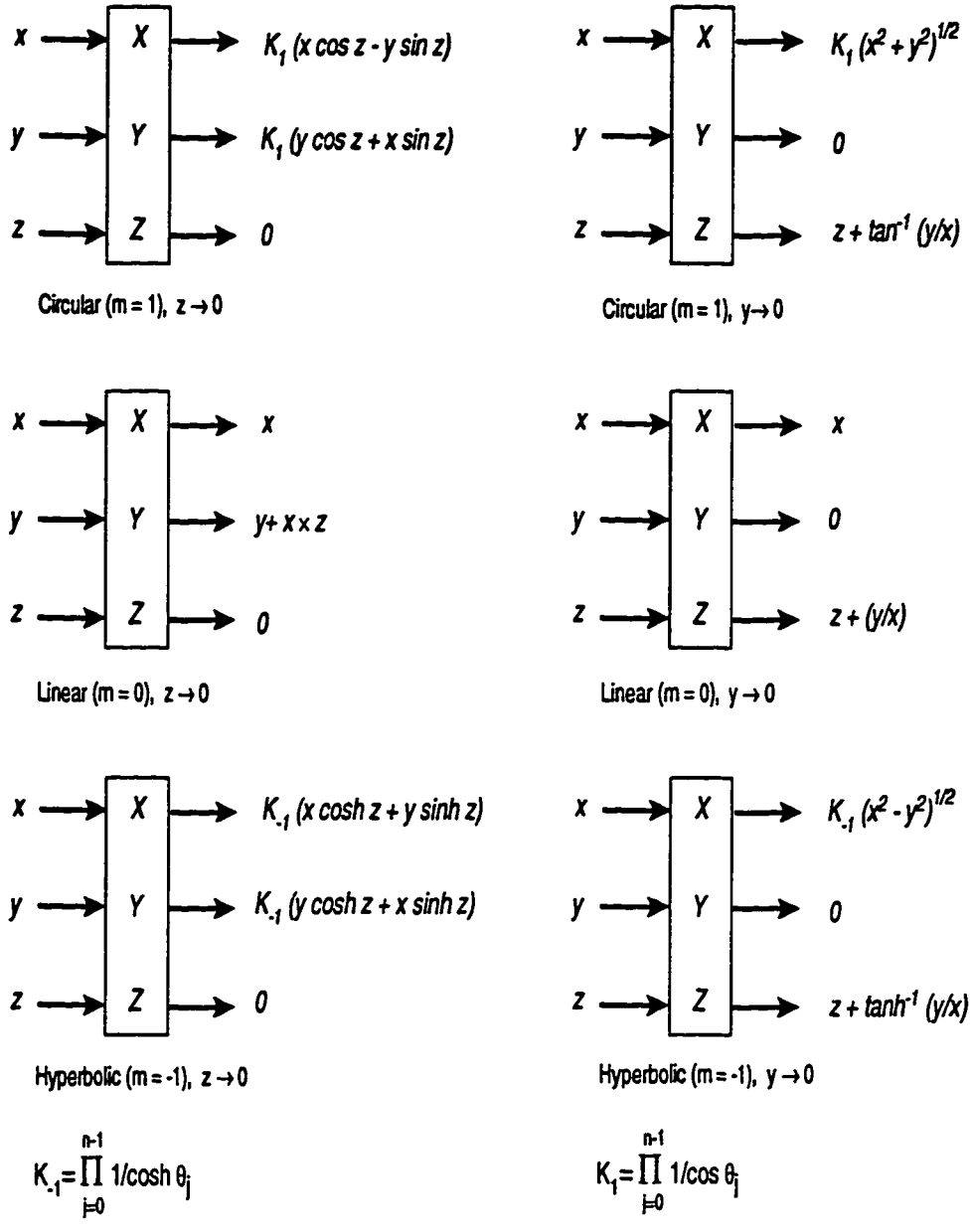

Figure 2.5: Input-output block diagram of CORDIC.

Table 2.5: CORDIC processor output in the vectoring operation ($y \rightarrow 0$).

m	x_{out}	z_{out}
1	$K_1\sqrt{x_0^2 + y_0^2}$	$z_0 + \tan^{-1}(y_0/x_0)$
0	x_0	$z_0 + y_0/x_0$
-1	$K_{-1}\sqrt{x_0^2 - y_0^2}$	$z_0 + \tanh^{-1}(y_0/x_0)$

2.4 Range of Convergence

At each CORDIC iteration, the quantities x, y, z are changed by a small step. The amount of the accumulated change in x, y, z is limited by the step size at each iteration and the number of iterations. To guarantee the convergence of CORDIC, each θ_i must satisfy the condition:

$$\theta_i \leq \sum_{j=i+1}^{n} \theta_j \qquad \forall i = 0, 1, ..., n-1. \tag{2.40}$$

This means that at each step, the sum of the remaining rotations must be enough to bring the angle to at least θ_{n-1}. Hence, the maximum error is within θ_{n-1}. For the circular mode, the range of convergence is defined as

$$\text{Maximum Range} = \sum_{i=0}^{n-1} \theta_i \tag{2.41}$$

The maximum ranges for the linear and hyperbolic modes are similarly defined. Table 2.6 shows the convergence sequence for the different modes. Initial angles greater than the indicated maximum range must be adjusted before applying CORDIC. The hyperbolic case needs the addition of θ_{3i+1} to the sequence to satisfy the convergence

Table 2.6: CORDIC Range of convergence.

m	Convergence Sequence	Maximum Range
1	0, 1, 2, 3, 4, i...	$\simeq 1.74$
0	1, 2, 3, 4, 5, $i+1$...	$1-2^{-n}$
-1	1, 2, 3, 4, 4, 5, ...	$\simeq 1.13$

criterion[2]. From Figure 2.4, we can notice that to move the end point of any vector on a hyperbola, the rotation angle is naturally limited by 45°.

2.5 Basic CORDIC Processor

A CORDIC processor contains three basic modules [15]; one performs the iterations for updating the values of x and y, another updates the angle, and a third module performs the descaling iterations. Figure 2.6 shows the modules that perform the updating operations. Notice that the module that performs iterations can be used in descaling iterations also. Figure 2.7 shows how to multiplex the hardware in the descaling stage. The iteration module shown is for realizing a single CORDIC iteration. It consists of dual barrel shifters and dual adders for updating both x_i and y_i simultaneously. The shift sequence is controlled by the convergence sequence, which can be stored in ROM or RAM, or generated on chip using a counter with some control units. The add or subtract operation is controlled by the value of

[2]Repeat {4, 13, 40, 121,..., k, $3k$+1, ...}

μ_i. The angle update module accumulates the values of θ_i for each iteration. The descaling module is similar to the iteration module. Hence, they can use the same data path by multiplexing.

2.6 Parallel and Pipelined CORDIC

The CORDIC algorithm is inherently iterative. Thus, the iterations cannot be performed in parallel even if n units are available because the input to each unit depends on the output of the previous one. Only updating of the x, y and z variables in each iteration can be parallelized. Thus, parallel CORDIC means updating all variables in parallel using three hardware units. However, some researchers have different definition for parallel CORDIC.

According to Hu [15], parallel CORDIC can be achieved by performing each iteration of CORDIC by a separate dedicated processor as shown in Figure 2.8. For n iterations, n stages of basic CORDIC processors are required, and similarly for the descaling iterations. By cascading these parallel stages of CORDIC processors, the entire CORDIC rotations, including the descaling iterations, can be performed in a single clock cycle.

By adding a buffer stage between successive stages of this parallel CORDIC processor, a pipelined CORDIC array can be obtained as shown in Figure 2.9. The buffer stage is to store the intermediate results after each CORDIC iteration, includ-

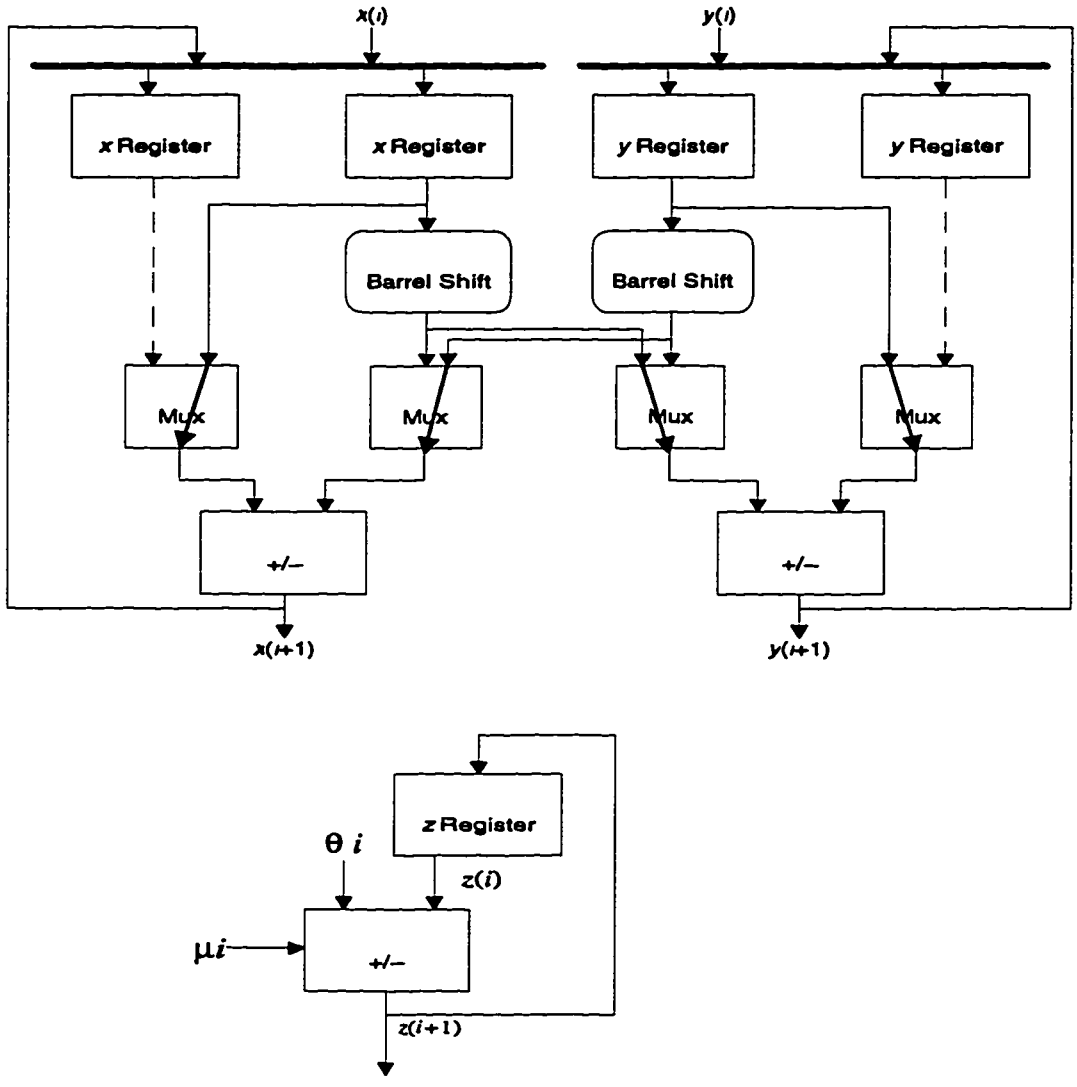

Figure 2.6: Basic CORDIC processor.

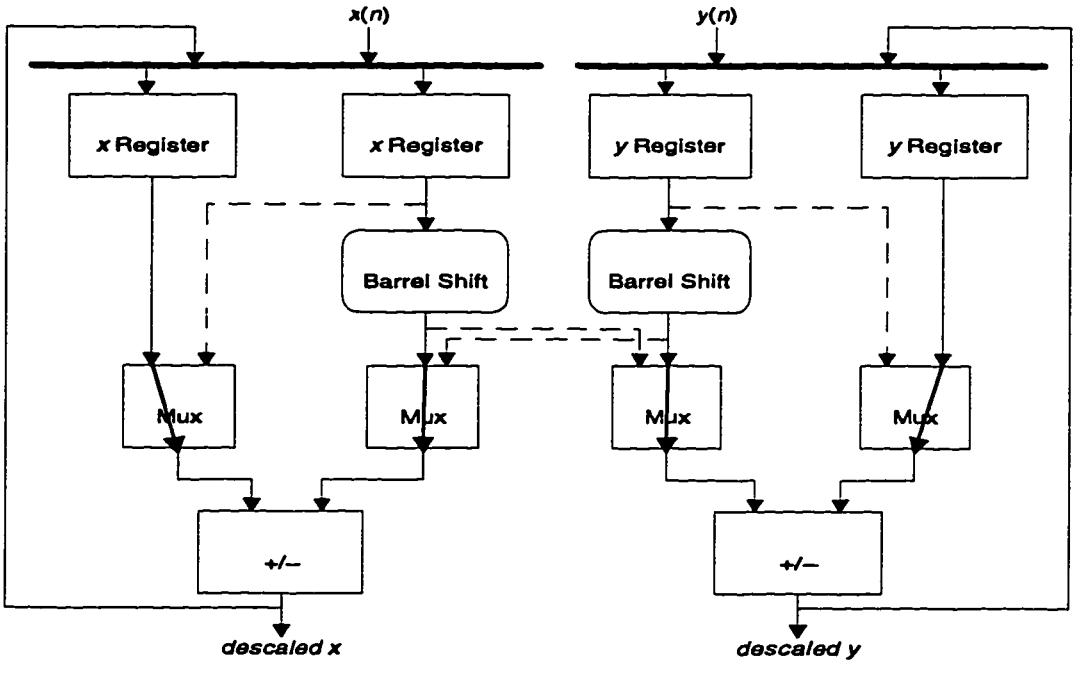

Figure 2.7: CORDIC descaling unit.

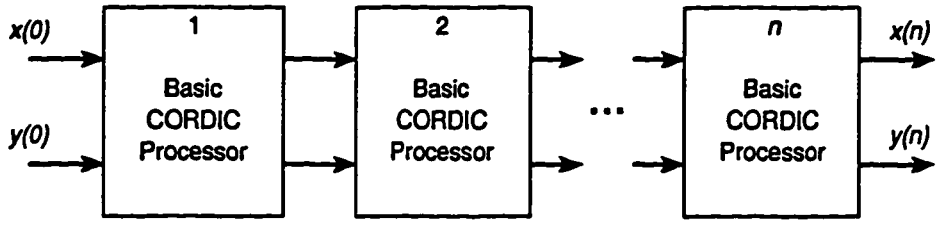

Figure 2.8: Parallel CORDIC processors.

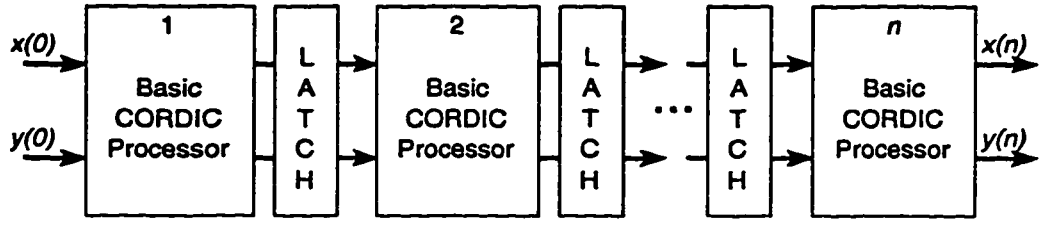

Figure 2.9: Pipelined CORDIC processors.

ing the descaling iterations. As a result, the computations of adjacent processors are isolated.

CORDIC has been implemented as a basic processor in many pipelined or parallel systems, as a special purpose DSP processor [7], [21], [25], and as an ASIC hardware for controlling robots [26].

2.7 CORDIC Applications

Beside being used for computing elementary mathematical functions (Appendix B), CORDIC has been used in several other areas. It has been used for speech synthesis [25], fast Fourier transform (FFT) [15], [23], [27], digital filtering [28], singular-value

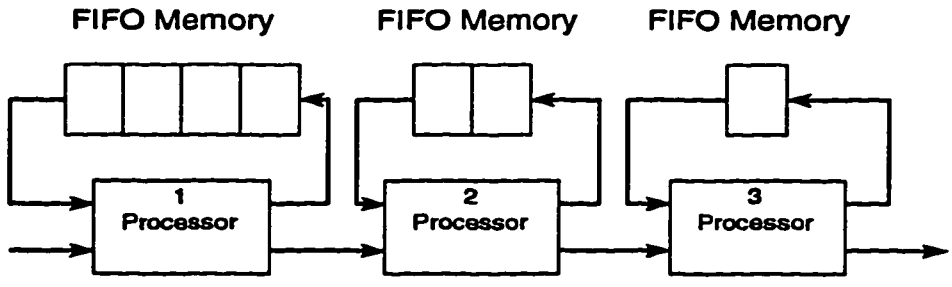

Figure 2.10: Radix 2 eight-point FFT with decimation in time.

decomposition (SVD) [17], and matrix arithmetic, such as Givens rotations and QR decomposition [29]. CORDIC has been extensively used in robotics applications, such as inverse kinematics [19], [26]. It has also been used in biomedical applications, e.g. blood flow velocity measurement [20]. This is a novel application that can be applied to the general problem of measuring fluid flow using non-contact means.

2.7.1 Using CORDIC in Fast Fourier Transform

For the fast Fourier transform, the CORDIC algorithm has been used for updating the twiddle factor and performing the butterfly operations [15], [23], [27]. Figure 2.10 shows the system used to implement a radix 2 eight-point FFT with decimation in time [27]. The system is based on the pipeline cascade implementation. The number of stages in the system equals $\log_2 8 = 3$ stages. All the processing nodes in the system are identical, where the basic CORDIC processor is used. The FIFO buffer attached to each node has a variable size that depends on its location, for example the first stage has a FIFO buffer of size four.

2.7.2 Using CORDIC in Givens Rotations

Givens rotations are very useful in matrix algebra and many digital signal processing applications. For example, in matrix algebra, Givens rotations are the basic operation for QR decomposition of a matrix, which is used for eigen value evaluation [30]. Solving a system of linear equations through matrix triangularization also relies on Givens rotations. In digital signal processing, Givens rotations are used to implement the butterfly operation for the fast Fourier transform (FFT) algorithm [23], [27].

A Givens matrix G_{pq} is a special square matrix which is similar to the identity matrix except for the elements g_{pq}, g_{pp}, g_{qp}, and g_{qq} as shown in the following example:

$$G_{pq} = \begin{array}{c} \\ p \\ \\ q \\ \end{array} \begin{array}{c} \begin{array}{ccccc} p & & & q & \end{array} \\ \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & \cos\theta & \cdots & \sin\theta & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & -\sin\theta & \cdots & \cos\theta & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \end{array} \tag{2.42}$$

Assume we are given a square matrix $A_{4\times4}$ and the Givens matrix G_{23},

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \tag{2.43}$$

$$G_{23} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.44}$$

Premultiplying A by G_{23} yields the matrix

$$A' = G_{23}A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a'_{21} & a'_{22} & a'_{23} & a'_{24} \\ a'_{31} & a'_{32} & a'_{33} & a'_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \tag{2.45}$$

where,

$$a'_{2i} = a_{2i}\cos\theta + a_{3i}\sin\theta \qquad \text{for } i = 1 \text{ to } 4. \tag{2.46}$$

$$a'_{3i} = -a_{2i}\sin\theta + a_{3i}\cos\theta \qquad \text{for } i = 1 \text{ to } 4. \tag{2.47}$$

Notice that if we consider a_{2i} and a_{3i} as the x and y components of a vector, then premultiplying by a Givens matrix rotates these components by an angle θ, which

can be easily performed using CORDIC. Now, if we want to make the resulting element $a'_{32} = 0$, we must have

$$-a_{22}\sin\theta + a_{32}\cos\theta = 0 \tag{2.48}$$

$$\tan\theta = \frac{a_{32}}{a_{22}} \tag{2.49}$$

The required rotation angle θ can be generated using CORDIC in the vectoring-circular mode. Then to apply the rotation, CORDIC in the rotation-circular mode is used [29].

2.8 Main Problems of the Standard Algorithm

The main problems associated with the standard CORDIC algorithm can be summarized in the following points:

1. CORDIC is iterative and cannot be parallelized. It obtains the desired result one bit at a time.

2. Intrinsically, CORDIC cannot take advantage of recent advances in microelectronics technology or microprocessor architecture, viz. availability of parallel multipliers, on-chip cache, and cheap SRAM memories.

3. Number of iterations is fixed, regardless of the value of inputs.

4. In each iteration, only the sign of the variable to be zeroed is examined, while its magnitude is ignored.

5. The results must be descaled by performing another CORDIC operation because there is no multiplier.

6. As a result of multiple shift operations, rounding or truncation noise is introduced.

7. Limited range of convergence, where the input data cannot exceed the maximum allowable range.

8. The fixed choice for the step size at each iteration creates strange sequencing requirements for the hyperbolic case [25] and limits the input data range as mentioned before.

9. The need for checking the sign of the input data before the start of each iteration introduces extra clock cycles to determine the direction of rotation for the next iteration.

2.9 Concluding Remarks

CORDIC is an algorithm that evaluates elementary arithmetic functions by rotating a two-dimensional vector in any of the three coordinate systems: circular, linear, and hyperbolic. It is useful in many applications such as, digital signal processing, robotics, and matrix arithmetic.

In this Chapter, the basics of CORDIC were introduced, its advantages and

applications were reviewed, and the main drawbacks of this algorithm were pointed out.

Chapter 3

Previous Solutions

Since the introduction of CORDIC by Volder in 1959, many researchers have attempted to improve it. We can classify previous work done to improve CORDIC into the following categories: reducing the number of iterations, removing the scale factor problem, and increasing the range of convergence. Some other attempts to improve CORDIC were mainly geared toward specific applications.

3.1 Reducing the Number of Iterations

A great deal of research work concentrated on reducing the number of iterations of CORDIC and speeding up the algorithm. There are different approaches to achieve this goal. Here we will summarize some of the major proposed solutions.

3.1.1 Using Multiplication with Table Look-up

Ahmed introduced a hybrid CORDIC that uses multiplication and look-up tables [10], [25]. This technique is based on advances in VLSI technology, where multiplier and storage devices are considerably less expensive than before. Two types of hybrid CORDIC were reported. In the first type, to rotate a vector, coarse rotations are performed first using additional look-up tables and a multiplier, followed by CORDIC refined rotations. This approach allows the tradeoff between execution speed and storage size. The second type of hybrid CORDIC is to perform coarse rotations using CORDIC, followed by Taylor series approximation using a multiplier. Hybrid CORDIC tried to improve CORDIC by using other evaluation techniques, e.g. Taylor series, and extra hardware that is used *outside* CORDIC. However, CORDIC itself still remains as a slow bit-serial algorithm.

3.1.2 Using Multiplication or Division

Timmermann et al. used a multiplier to reduce the number of iterations [11]. This method is based on the fact that only the early iterations of CORDIC contribute significantly to the accuracy of the final result. As the iteration index increases the result accuracy decreases. Hence, in this technique, CORDIC iterations are performed up to j iterations, where $j > (n+1)/2$, for n-bit accuracy. Then a multiplication or division operation is performed in the rotation or vectoring operation,

Table 3.1: Additional operations for truncated CORDIC algorithm.

	Operation	Condition
Rotation	$x_n = x_j - mz_jy_j$ $y_n = z_jx_j + y_j$	$j > (n+1)/2$
Vectoring	$x_n = z_j$ $z_n = z_j + y_j/x_j$	$j > (n+1)/2$ $j > n/3 + 0.472$

respectively. Table 3.1 summarizes these additional operations that are performed after j iterations, the limits of j are derived from analyzing the Taylor expansion of

$$z_j = m^{-1/2}\tan^{-1}(m^{1/2}y_j/x_j) \tag{3.1}$$

This approach reduces the number of iterations needed by adding one multiplication in the rotation operation, or one division in the vectoring operation. However, both the multiplication and division operations are expensive. Moreover, the multiplier is utilized only after j iterations. Effectively, CORDIC itself was not improved.

3.1.3 Using Redundant Number Systems

Another approach to speed-up CORDIC is by using redundant number systems. However, this approach has several drawbacks, namely, converting from standard to redundant number system and vice versa, variable scale factor, sign detection problem, complex control, and the resulting CORDIC algorithm is not unified as the original one. For example, Antelo and Bruguera proposed a radix 2-4 redundant

CORDIC that has constant scale factor and can operate in four modes, the vectoring and rotation operations in the circular and hyperbolic modes [12]. However, it is a complex algorithm that has three kinds of special cases during the iterations. It requires prescaling for x and y to confine the x coordinate to an interval around one, so that their selection function can operate properly.

Dawid and Meyr proposed a radix 2 redundant number CORDIC which is called Differential CORDIC or DCORDIC [13]. It transformed the original CORDIC algorithm to a redundant formulation. This results in constant scale factor, new variables, and different sign estimation method. They also derived parallel architectures for the rotation as well as the vectoring operations. This solution solves the variable scale factor problem of redundant CORDIC and avoids additional operations compared to other redundant systems. However, we believe that CORDIC can be improved without resorting to redundant system implementations.

Another method is angle recoding which is similar to redundant number system. Hu and Naganathan proposed the angle recoding technique which encodes the desired rotation angle as a linear combination of few elementary rotation angles [31]. This requires the rotation angle to be known in advance, such that recoding can be performed. The recoding is done by a greedy algorithm which has a complexity of $O(n^2)$. The choice of μ_i is extended to be one of the following values: $\{1, 0, -1\}$ which gives more flexibility than $\mu_i = \pm 1$. They showed that the total number of required elementary rotation angles is reduced by at least 50% while maintaining

the computational accuracy. The number of iterations decreases in proportion to the number of required elementary angles. This approach required an extra recoding step and did not introduce any significant improvement on the algorithm.

Duprat and Muller proposed a branching CORDIC which is also based on redundant numbers [14]. It performs two parallel micro rotations in the same direction if $\mu_i = \pm 1$, or in the opposite direction if $\mu_i = 0$ (branching occurs). For example, in the rotation operation, at each step, they examine p digits of z_i just to decide the value of μ_i. Then, if the examination of the p digits is sufficient to be sure of the sign of z_i, then the corresponding μ_i is chosen. Otherwise, they branch and perform *two* CORDIC computations in parallel, one with $\mu_i = 1$ and another with $\mu_i = -1$. They proved that branching occurs only when z_i is sufficiently small to ensure convergence. Hence, it will not lead to at most two parallel computations. They also reported that if no further branching occurs then both computations lead to a correct approximation of the sine and cosine. However, this technique consumes more area because it requires two conventional CORDIC algorithms to be executed in parallel.

3.1.4 Bit Parallel approach

A new CORDIC to evaluate sine and cosine functions with variable scale factor was proposed in [32]. It can reduce the number of iterations for 16-bit numbers to a maximum of six iterations and an average of 4.5 iterations. This is accomplished

by scanning two bits at a time instead of checking the sign-bit only. The algorithm works with fixed point numbers for angles between ± 1. The update equation of z is

$$z_{i+2} = z_i - q_i\theta_i - q_{i+1}\theta_{i+1} \tag{3.2}$$

where (q_i, q_{i+1}) are obtained based on the values of the scanned bits of z, and according to Booth algorithm. However, this algorithm is simulated only for the rotation operation in the circular mode.

Similarly in [24] and [33], the sign and one bit are scanned in parallel. The choice of θ_i depends on the scanned bits of z_i. Variable scale factor is obtained as well, but the update equation of z is similar to the standard CORDIC.

To the best of our knowledge, a multiple scanned bit technique was recently reported in [34], for the division problem only, not CORDIC.

3.2 Removing the Scale Factor Problem

Ahmed introduced a new convergence sequence which results to an integer power of two scale factor and an increased range of convergence [25]. It is a non-decreasing sequence where some of the basic rotations are repeated or removed. However, the method of constructing the sequence was not explained and the accuracy of rotation is decreased. Similar technique was proposed in [35] to generate an integer power of two scale factor, where it can identify the required sequence to generate any desired scale factor value. However, the number of iterations is increased by 50%.

3.3 Increasing the Range of Convergence

There are two ways to solve the CORDIC range of convergence limitation, namely, reduction of the input argument, or expansion of the CORDIC convergence domain. In the first approach, the argument is reduced, then it is evaluated, and the final result is constructed according to the original reduction. Walther has proposed a set of reduction functions for the input arguments [2]. For example, to evaluate sine of a large argument, θ, the angle is first divided by $\pi/2$ producing a quotient Q and a remainder D with $|D| < \pi/2$ which falls in the range of convergence. Hence, the evaluation of $\sin\theta$ is reduced to the evaluation of $\sin D$ or $\cos D$ only, and the value of D is within the range of convergence.

$$\sin\theta = \sin(Q\frac{\pi}{2} + D) = \begin{cases} \sin D & \text{if } Q \bmod 4 = 0 \\ \cos D & \text{if } Q \bmod 4 = 1 \\ -\sin D & \text{if } Q \bmod 4 = 2 \\ -\cos D & \text{if } Q \bmod 4 = 3 \end{cases} \tag{3.3}$$

However, this method requires more chip area for VLSI implementation and the reduction time may exceed the actual evaluation time of CORDIC; in addition to control penalty and one division operation that are required.

Haviland and Tuszynski suggested a prerotation approach [7]. If a vector falls outside the convergence range, then rotations by $\pi/2$ and $\pi/4$ are performed. This method results in both execution time and control overhead.

Hahn et al. proposed an argument reduction CORDIC with "unlimited" range

of convergence [8]. This argument reduction algorithm deals with floating point CORDIC. It operates according to the following basic steps:

1. Conversion of external floating point format to internal fixed point format.
2. Reduction of the argument to a given convergence domain by performing suitable transformations.
3. Evaluation of CORDIC functions using the reduced argument.
4. Postprocessing the result by combining the computed function with the transformation done in step 2.
5. Conversion of fixed point to floating point format.

The hardware required for executing these steps includes adders, carry save arrays, shifters, multiplexers, and a ROM. The complication is obvious in this approach.

The second approach to increase the range of convergence is by expansion which can be achieved by changing the convergence sequence. This method is unified and regular, and the hardware area needed is less than the first approach [25]. However, the expanded range is limited.

Hu et al. used negative iteration index in the convergence sequence [9]. The choice for negative indexed iterations must satisfy the convergence condition of CORDIC. For example, in the circular mode, if the convergence sequence includes

$i = -1$ and -2, the resulting θ_{max} will be 239.3°. The addition of negatively indexed iterations may result in large arithmetic errors than those introduced by positively indexed iterations. The reason is that with the addition of negatively indexed iterations, the scale factor is increased. Hence, the values of x and y may overflow. In a finite-length fixed point representation, to prevent overflow an increased number of high order bits must be reserved. However, since the word length is fixed, the increase in high order bits is taken from the low order bits. This method prevents overflow, but results in rounding error. They reported some best chosen convergence sequences that expand the range without loosing the accuracy.

3.4 Application Specific Research

Several other attempts were proposed to improve CORDIC, but these have been mainly geared toward specific applications. Some of such research work will be mentioned in this section.

3.4.1 Floating Point CORDIC

The original CORDIC implementation was for fixed point format. Since then, many fixed point and floating point implementations of CORDIC were reported. A full precision floating point CORDIC was proposed in [22]. The purpose of this floating point CORDIC is to avoid the accuracy problem. This is because the main drawback

in the floating point computation of the standard CORDIC algorithm occurs in the inherent fixed point resolution of the angle. When calculating angles close to or smaller than the angle resolution, the accuracy becomes unacceptable. In this approach, angles are represented as a combination of exponent, micro rotation bits and two bits to indicate prerotations over $\pi/2$ and π radians. The achievement was accuracy only.

3.4.2 Householder CORDIC

Householder CORDIC is a multi-dimensional CORDIC algorithm [36]. By expressing matrix computations in terms of higher dimensional rotations, they can be implemented using the Householder CORDIC. For complex large matrix computations, the maximum throughputs of parallel arrays built out of CORDIC units that process two real numbers at a time are very low compared to the throughputs achievable for real data sets of equal size. To bring the throughputs closer to real data throughputs, further parallelism must be explored. The rotation concept of Householder CORDIC was extended to vectors in spaces with more than two dimensions [18]. This method was employed to speed-up the computation of the singular value decomposition (SVD) of complex matrices.

3.4.3 On-line CORDIC for Matrix Applications

A redundant and on-line CORDIC was proposed in [17]. This new CORDIC was applied to applications in matrix triangularization and SVD. The major modifications of this CORDIC can be summarized in the following. First, redundant addition is used for calculating the rotation angle $\tan^{-1} y/x$. This approach is faster than using carry propagate adders, but results in a variable scale factor. Second, the angles are transmitted in decomposed forms. This eliminates the recurrences of angles in both CORDIC modules and reduces the communication bandwidth. Finally, on-line addition is used in the implementation of the rotation modules.

3.4.4 CORDIC-Based All-Pass Filters

For filters, a CORDIC-based pipeline architecture to perform Givens rotations was proposed in [16]. This architecture is for normalized lattice all-pass filters. By using pipeline interleaving technique, a large number of filters can be obtained. This technique fully exploits the pipeline property of the Givens rotation processor regardless of the recursive form of the infinite impulse response all-pass filters.

3.5 Concluding Remarks

In this chapter, a literature review of previous work was presented. We can classify previous work done to improve CORDIC into the following categories: reducing the

number of iterations, removing the scale factor problem, and increasing the range of convergence. Several other attempts were proposed to improve CORDIC, but these have been mainly geared toward specific applications.

Chapter 4

The HCORDIC Algorithm

In this thesis, we propose major modifications to the original iterative CORDIC such that all the drawbacks are eliminated. The modifications we propose for the CORDIC algorithm are based on the following:

- VLSI advances: With the advances in VLSI technology, most new processors come with a multiplier in their ALU and an on-chip RAM.
- Adaptive algorithm: The HCORDIC iterations are based on both the sign and magnitude of the variable to be zeroed.
- Flexible choice for the step size: Restrictions on the magnitudes of θ and δ are removed.

The modified algorithm is based on scanning several bits of the CORDIC operands in parallel. The standard CORDIC scans only the sign bit and reaches the required

accuracy in n iterations, where n is the number of bits in the mantissa, $n = 23$ in 32-bit floating point format. The proposed HCORDIC scans s bits at a time. Thus, we can achieve a speed-up factor of at least s as will be shown in this chapter.

4.1 HCORDIC Equations

At iteration i, the values of x, y, and z are updated *as well as* the current scale factor according to the following equations:

$$x_{i+1} = x_i + m y_i \delta_i \tag{4.1}$$

$$y_{i+1} = y_i - x_i \delta_i \tag{4.2}$$

$$z_{i+1} = z_i + \theta_i \tag{4.3}$$

$$K_{i+1} = K_i * \kappa_i \tag{4.4}$$

where

$$\kappa_i = \begin{cases} \cos\theta_i & m = 1 \\ 1 & m = 0 \\ \cosh\theta_i & m = -1 \end{cases} \tag{4.5}$$

δ and θ values will be chosen depending on the operation type as explained below. Notice that the scale factor now enters into our iterations since we will choose the rotation angles according to the magnitude of y or z. Another modification is that we no longer need μ, because our direction of rotation is constant due to the adaptive choice of the step size. In the standard CORDIC, the angle of rotation oscillates

between clockwise and counterclockwise directions. This is due to the rigid choice of θ and δ that tends to overcorrect the value of y in the vectoring operation, or z in the rotation operation.

Notation

Before we explain our modifications to the CORDIC algorithm, we have to state the notations we are going to use. Assuming Num to be a floating point number, we define the following symbols

$$\begin{aligned} n &= \text{number of bits in the mantissa} \\ s &= \text{number of leading bits in the mantissa to be scanned in parallel} \\ Num_e &= \text{exponent part of } Num \\ Num_f &= \text{mantissa part of } Num \end{aligned}$$

4.2 Rotation Operation ($z \rightarrow 0$)

In the rotation operation, the goal is to iteratively bring z to zero ($z \rightarrow 0$). This process is controlled by the values of θ_i. Hence, if we choose θ_i to be equal to $-z_i$, we will get the desired value in one iteration, as follows:

$$z_{i+1} = z_i + \theta_i = z_i + (-z_i) = 0 \tag{4.6}$$

However, the possible combinations of θ_i will increase to $O(2^n)$, and result in a huge look-up table. Therefore, we limit the choice of θ_i to the leading s bits of z_i.

The look-up table stores the values of $\tan\theta_i$ and $\tanh\theta_i$. Fortunately, we noticed that for smaller values of z_i, where $z_e < -n/2$, these functions can be approximated without look-up table. Hence, beyond this limit, we do not have to limit the choice of θ_i to the leading s bits of z_i.

We can summarize the values of θ_i and δ_i in the rotation operation using the following equations:

$$\theta_i = \begin{cases} -z_s 2^{z_e} & z_e \geq -n/2 \\ -z_i & z_e < -n/2 \end{cases} \tag{4.7}$$

$$\delta_i = \begin{cases} \tan\theta_i & m = 1 \\ \theta_i & m = 0 \\ \tanh\theta_i & m = -1 \end{cases} \tag{4.8}$$

where z_s has the leading s bits of z_f padded with zeros to get an n-bit number.

$$z_s = 1.\underbrace{\overbrace{z_1 z_2 \cdots z_s}^{s \text{ bits}} 00\cdots 0}_{n \text{ bits}} \tag{4.9}$$

The specified limit of $z_e < -n/2$ can be justified by looking at the Taylor series expansion as follows:

$$\tan\theta = \theta + \frac{\theta^3}{3} = \theta(1 + \frac{\theta^2}{3}) \tag{4.10}$$

$$\tan\theta = \theta \quad \text{If } \frac{\theta^2}{3} << 1 \tag{4.11}$$

This condition is true if

$$\frac{\theta^2}{3} < 2^{-n} \tag{4.12}$$

To be safe, three is approximated as two.

$$\theta^2 < 2 \times 2^{-n} \tag{4.13}$$

$$\theta < \sqrt{2} \times 2^{-n/2} \tag{4.14}$$

Assuming maximum value for θ_f which is two, and substituting in Equation 4.14, we get

$$2.2^{\theta_e} < \sqrt{2} \times 2^{-n/2} \tag{4.15}$$

$$2^{\theta_e} < 2^{-n/2} \tag{4.16}$$

Hence, the approximation of $\tan\theta = \theta$ is applicable when the exponent of θ is less than $2^{-n/2}$. This is true for $\tanh\theta$ as well.

According to Equation 4.7, the choice of θ ensures that the leading s bits of the mantissa of z will be zeroed in parallel. Therefore, we expect the speed-up in the new CORDIC, compared to the standard one, to be $4s$. This is because in the rotation operation, the standard CORDIC requires $2n$ iterations to get the result and remove the scale factor; while HCORDIC requires $n/2s$ iterations only. Table 4.1 summarizes the number of iterations required in both the standard and the new CORDIC.

Table 4.1: Number of iterations required in the rotation operation for both the standard and the new CORDIC.

CORDIC	Iterations
Standard	$2n$
New	$n/2s$

Table 4.2: Comparison between the standard and the new CORDIC in the circular mode of rotation.

Rotation	δ_i	θ_i
Standard	2^{-i}	$\tan^{-1}\delta_i$
New	$\tan\theta_i$	$-z_s 2^{z_e}$

Table 4.2 compares the main differences between the standard CORDIC algorithm and the new CORDIC algorithm in the circular mode of the rotation operation. In the standard CORDIC, δ_i is fixed to 2^{-i}. However, in HCORDIC, the value of θ_i is chosen in an adaptive manner based on z_i. In the standard CORDIC, the value of δ_i determines the value of θ_i. While in the new algorithm, the value of θ_i determines the value of δ_i. The same discussion can be applied to the other two modes, linear and hyperbolic.

Required Look-up Table

We require a look-up table for the values δ given θ. The size of this table depends on s and z_e. As discussed earlier, due to the properties of the trigonometric and hyperbolic tangent functions, and finite word length, not all values of $\tan\theta$ and $\tanh\theta$ have to be stored. The δ look-up table will store the 2^s combinations of the mantissa, for 13 exponents. The following equations show the range of z_e where a look-up table is needed and the look-up table size (32-bit data is assumed).

$$\text{Range of } z_e = -12 \leq z_e \leq 0 \tag{4.17}$$

$$\delta\text{-Look-up table size} = 13 \times 2^s \tag{4.18}$$

4.3 Vectoring Operation ($y \rightarrow 0$)

In the vectoring operation, the goal is to iteratively bring y to zero ($y \rightarrow 0$). This process is controlled by the values of δ_i. Hence, if we choose δ_i to be equal to y_i/x_i, we will get the desired value in one iteration, as follows:

$$y_{i+1} = y_i - x_i\delta_i = y_i - x_i\frac{y_i}{x_i} = 0 \tag{4.19}$$

However, the possible combinations of x_i, y_i, and δ_i will increase to $O(2^n)$, and result in a huge look-up table. Therefore, we limit the choice of x_i, y_i, and δ_i to the leading s bits only.

We can summarize the values of δ_i and θ_i in the vectoring operation using the following equations:

$$\delta_i = \begin{cases} 1 & y_i \geq x_i \quad \text{and} \quad m \neq 0 \\ (y_s/x_s)2^{y_e - x_e} & y_i < x_i \quad \text{and} \quad m \neq 0 \\ (y_s/x_s)2^{y_e - x_e} & m = 0 \end{cases} \tag{4.20}$$

$$\theta_i = \begin{cases} \tan^{-1}\delta_i & m = 1 \\ \delta_i & m = 0 \\ \tanh^{-1}\delta_i & m = -1 \end{cases} \tag{4.21}$$

where y_s has the leading s bits of y_f padded with zeros to get an n-bit number.

$$y_s = 1.\underbrace{\overbrace{y_1 y_2 \cdots y_s}^{s \text{ bits}} 00 \cdots 0}_{n \text{ bits}} \tag{4.22}$$

and x_s has the leading s bits of x_f padded with ones to get an n-bit number.

$$x_s = 1.\underbrace{\overbrace{x_1 x_2 \cdots x_s}^{s \text{ bits}} 11 \cdots 1}_{n \text{ bits}} \tag{4.23}$$

The above masking technique is used to obtain values of x_s and y_s such that the value of the resulting δ_i will not alter the sign of y_i during iterations. The set of conditions that decide the way of computing δ_i is designed for several reasons. For $m \neq 0$, the condition will limit the maximum value of δ_i to be one, which has a great impact on the look-up table size. For $m = 0$, we do not limit the value of δ_i by one because we want to speed-up convergence; and this choice will not increase the look-up table size because no look-up is required in the linear mode ($m = 0$).

Table 4.3: Number of iterations required in the vectoring operation for both the standard and the new CORDIC.

CORDIC	Iterations
Standard	$2n$
New	n/s

According to Equation 4.20, the choice for δ_i ensures that the leading s bits of the mantissa of y will be zeroed in parallel. Therefore, we expect the speed-up in the new CORDIC, compared to the standard one, to be $2s$. This is because in the vectoring operation, the standard CORDIC requires $2n$ iterations to get the result and to remove the scale factor; while HCORDIC requires n/s iterations only. Table 4.3 summarizes the number of iterations required in both the standard and the new CORDIC.

Table 4.4 compares the main differences between the standard CORDIC algorithm and the new CORDIC algorithm in the circular mode of the vectoring operation. In the standard CORDIC, δ_i is fixed to 2^{-i}. However, in HCORDIC, the value of δ_i is adapted to the values of x_i and y_i for all three modes.

Required Look-up Table

For the vectoring operation, *two* look-up tables are required. One for the value of y_s/x_s and the other for θ given δ. The size of the first table depends on x_s and y_s

Table 4.4: Comparison between the standard and the new CORDIC in the circular mode of vectoring.

Vectoring	δ_i	θ_i
Standard	2^{-i}	$\tan^{-1}\delta_i$
New	$y_s/x_s 2^{y_e-x_e}$	$\tan^{-1}\delta_i$

and it is given by the following equation:

$$\delta\text{-Look-up table size } = 2^{2s} \tag{4.24}$$

The values stored in this table are all possible values of $\delta_f = y_s/x_s$. We limit the precision of this result to s bits only (i.e. $\delta_f = \delta_s$) since δ will be used as the *address* for the second look-up table.

The size of the second table will depend on δ_s and $\delta_e = y_e - x_e$, the exponent part of δ. Due to the properties of the trigonometric and hyperbolic tangent functions, and finite precision, we do not have to store all values of $\tan^{-1}\theta$ and $\tanh^{-1}\theta$. The θ look-up table will store the 2^s combinations of the mantissa, for 13 exponents. The following equations show the ranges of δ_e where a look-up table is needed (32-bit data is assumed).

$$\text{Range of } \delta_e \;=\; -12 \le \delta_e \le 0 \tag{4.25}$$

$$\theta\text{-Look-up table size } = 13 \times 2^s \tag{4.26}$$

4.4 Look-up Tables

The size of the required look-up tables are summarized in Table 4.5, notice the common tables are mentioned once only. In the rotation operation, we need look-up tables for δ which are $\tan\theta$ in the circular mode and $\tanh\theta$ in the hyperbolic mode. The linear mode does not require any look-up table. In the vectoring operation, we need two look-up tables. One for δ and another for θ. The look-up table for δ uses Equation 4.20 to evaluates δ given x_s and y_s. The look-up table for θ uses Equation 4.21 to evaluate θ given δ. Similarly, the linear mode does not require any look-up table. For the scale factors, we need tables for $\cos\theta$ in the circular mode, and $\cosh\theta$ in the hyperbolic mode. The linear mode does not require any scaling.

If we sum up the total size of the required look-up tables, it gives

$$\text{Size in bits} = s \times 2^{2s} + 78 \times 32 \times 2^{s} \tag{4.27}$$

Figure 4.1 shows the plot of Equation 4.27. It can be noticed that if the number of scanned bits, s, is less than eight then the look-up table size of θ will dominate. While if we increase s beyond eight, the look-up table size of δ will dominate.

4.5 Noise Model

A simple noise model is developed in this section to show the advantage of the new algorithm over the standard one in increasing the signal-to-noise ratio (SNR). The

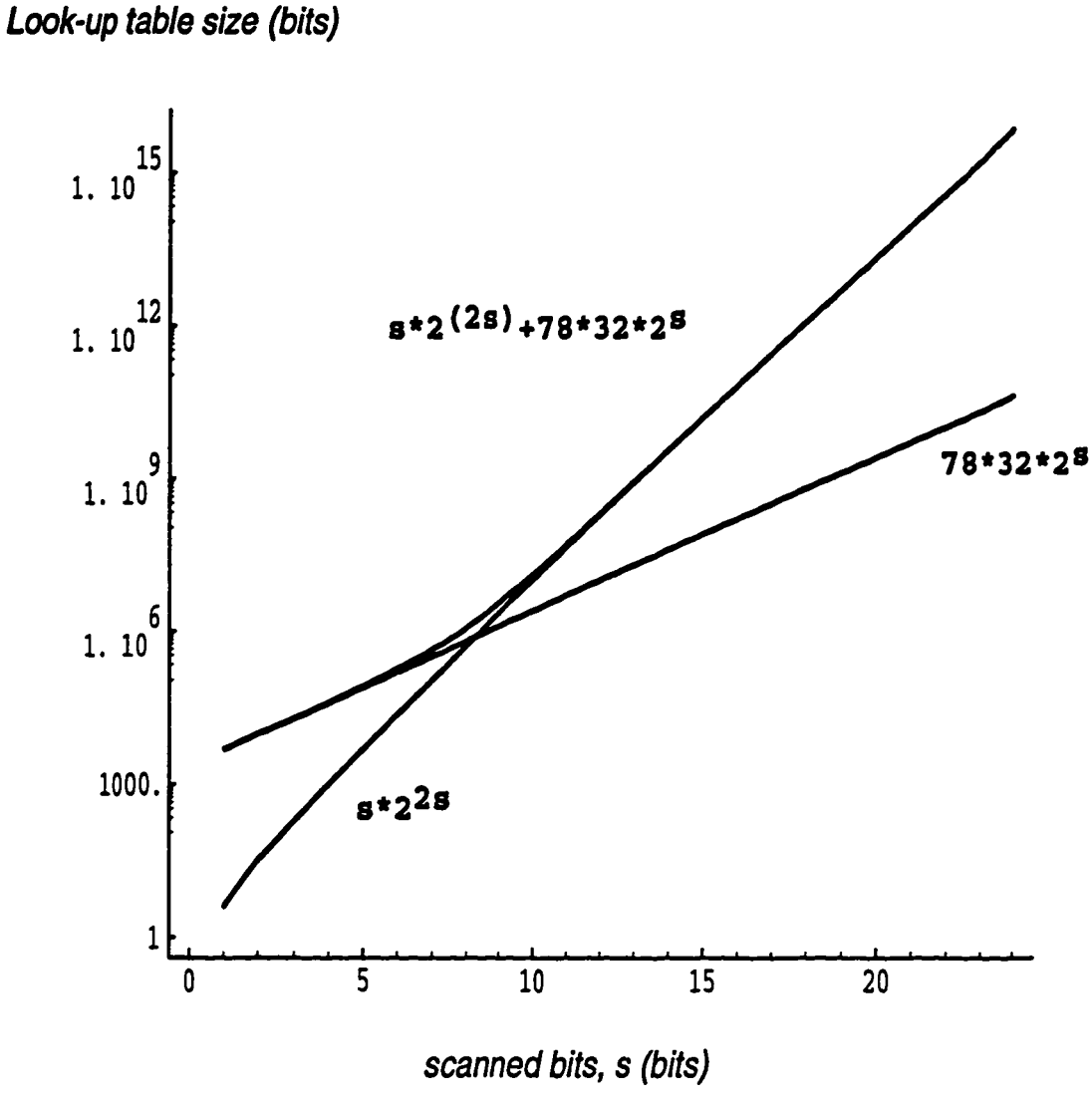

Figure 4.1: Effect of the number of bits scanned in parallel, s, on the look-up table size .

Table 4.5: Summary of the required look-up tables and their sizes.

	Circular	Linear	Hyperbolic
Rotation	$13 \times 2^s \rightarrow \tan\theta$	–	$13 \times 2^s \rightarrow \tanh\theta$
	$13 \times 2^s \rightarrow \cos\theta$	–	$13 \times 2^s \rightarrow \cosh\theta$
Vectoring	$2^{2s} \rightarrow \delta$	same	same
	$13 \times 2^s \rightarrow \tan^{-1}\delta$	–	$13 \times 2^s \rightarrow \tanh^{-1}\delta$

noise analysis is based on the following assumptions:

1. After each iteration, noise is introduced in the least significant bit (LSB) due to number rounding or truncation.

2. Noise from each iteration is additive.

3. Magnitude of relative noise ν is:

$$-2^{-n} < \nu < 2^{-n} \tag{4.28}$$

4. Noise is statistically independent.

Hence, the relative noise magnitude at the end of n iterations is:

$$-n2^{-n} < \nu_n < n2^{-n} \tag{4.29}$$

After n iterations, the number of bits in error will be:

$$\text{Bits in error} = \lceil \log_2 n \rceil \tag{4.30}$$

For the standard CORDIC, the number of iterations is always n (fixed). While for HCORDIC, the number of iterations is i (adaptive). Thus, the gain in SNR of HCORDIC compared to the standard CORDIC is:

$$\text{SNR(CORDIC)} = 2^{n-\alpha} \tag{4.31}$$

$$\text{SNR(HCORDIC)} = 2^{n-\beta} \tag{4.32}$$

$$\text{SNR gain} = 2^{\alpha-\beta} \tag{4.33}$$

where,

$$\alpha = \lceil \log_2 n \rceil \tag{4.34}$$

$$\beta = \lceil \log_2 i \rceil \tag{4.35}$$

Example

For the new algorithm, if we take the simulation result reported in Table 4.6 as an example. The operation performed is rotation in the circular mode for 32-bit data, and three bits of the variable to be zeroed are scanned in parallel. The number of iterations required by the standard CORDIC is 48 iterations, while HCORDIC requires 3 iterations only. Thus, we have,

$$\alpha = \lceil \log_2 48 \rceil = 6 \tag{4.36}$$

$$\beta = \lceil \log_2 3 \rceil = 2 \tag{4.37}$$

Table 4.6: HCORDIC iterations for the input $(x, y, z) = (2, 0, 30°)$, where the operation is rotation in the circular mode.

i	x	y	z
initial	+2.000000e+00	+0.000000e+00	+3.000000e+01
1	+2.000000e+00	+1.016499e−01	+9.044878e−02
2	+1.999851e+00	+1.045796e−01	+6.519415e−03
3	+1.999839e+00	+1.048071e−01	+0.000000e+00
Result	+1.997259e+00	+1.046719e−01	+0.000000e+00
Error	+0.000000e+00	−7.450581e−09	+0.000000e+00

$$\text{SNR gain} = 2^4 \tag{4.38}$$

which is equivalent to $20 \log_2 16 = 80$ dBs.

4.6 Advantages of the New Algorithm

In HCORDIC, all the drawbacks of the standard algorithm have been removed. The advantages of HCORDIC can be summarized in the following:

- Scale factor is easily removed: In the new algorithm the scale factor is not constant and can not be precalculated. Instead the scale factor is updated during each iteration. At the end, the corrected results are obtained by multiplying the final CORDIC output by the scale factor.

- Reduced number of iterations: At each iteration, the step size is determined based on the magnitude of the variable to be zeroed. After each iteration,

s bits are reduced to zero in parallel. Therefore, the number of iterations is reduced.

- Increased range of convergence: The choice of θ and δ is not limited as in the standard CORDIC. Therefore, the range of rotation "angle" is unrestricted.

- Increased signal-to-noise ratio: As the number of iterations is reduced, the noise introduced due to rounding or truncation is reduced as well.

These advantages have been confirmed by the numerical simulations presented in Chapter 5.

4.7 Concluding Remarks

CORDIC is an algorithm that was introduced when multipliers were very expensive. Consequently it could not take advantage of VLSI advances in its basic formulation. In this chapter, we successfully parallelized the standard CORDIC algorithm that was thought to be inherently serial. Our high-performance, HCORDIC, removed all the drawbacks of the standard CORDIC using a multiplier and look-up tables. The main ideas of HCORDIC in different operations and modes were discussed. The required look-up table size was analyzed, and a noise model of HCORDIC was developed. Finally, the main advantages of the HCORDIC algorithm were summarized.

Chapter 5

Simulation of HCORDIC

To illustrate the superiority of the HCORDIC algorithm, we implemented the standard CORDIC and the HCORDIC algorithms in C-code. We performed numerical simulations on SUN SPARCstation 10 which is a 32-bit machine using the standard IEEE floating-point data format. For reference, the results were compared with those obtained from the standard CORDIC and the C-language math library.

The program reads the input parameters shown in Table 5.1. The iterations will stop when the variable to be zeroed is equal to zero or its exponent is very small (around 10^{-8} for 32-bit floating point data). After the outputs of the required CORDIC functions are produced, they are compared with the correct results and accordingly the errors are calculated. Currently, the program works on single precision, 32-bit floating point format. However, it can be easily extended to double precision, 64-bit floating point format.

Table 5.1: Input parameters of the program.

Parameter	Comments
s-bits	number of bits to be scanned
Op	rotation or vectoring
Mode	circular, linear or hyperbolic
x, y	(x, y) components of the vector
z	rotation angle accumulator

5.1 Simulation Results

The program flow chart is shown in Figure 5.1. The pseudocode of the algorithm is available in Appendix C.

In all the simulation results presented here, three bits of the variable to be zeroed are scanned in parallel (i.e. s=3). This choice is made for illustration purposes only. The simulations were carried out for rotation and vectoring operations in the three different modes.

5.1.1 Rotation Operation in the Circular Mode

This operation is equivalent to vector rotation by a specified angle. The result presented here is for the following input:

$$(x, y, z) = (2, 0, 30°) \tag{5.1}$$

Thus, the objective is to rotate the vector $[2 \;\; 0]^t$ counterclockwise by 30°.

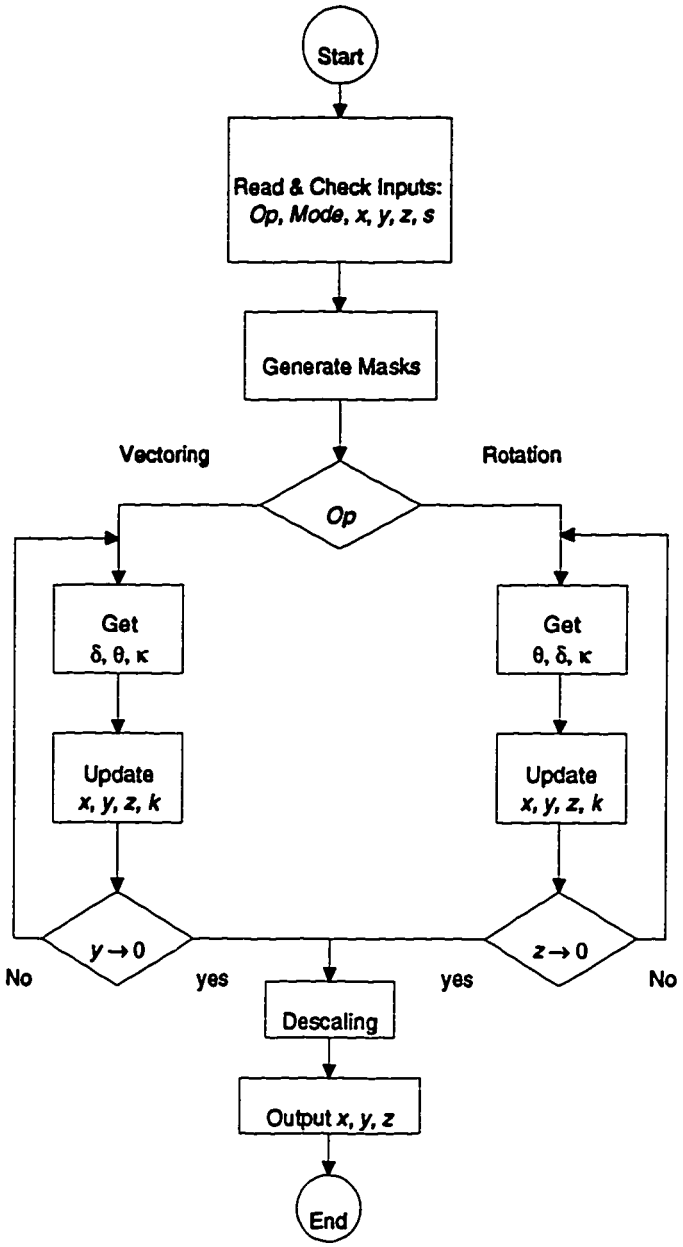

Figure 5.1: Flow chart of the new algorithm.

Table 5.2 shows the iterations performed by the standard CORDIC algorithm. Notice that the variable to be zeroed (z) oscillates around the desired value. The iteration index of the circular mode has to start from zero according to the convergence sequence. After each iteration, one bit of the desired answer is obtained such that after 24 iterations, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

Table 5.3 shows the iterations performed by the new CORDIC algorithm. Notice that the variable to be zeroed (z) monotonically approaches the desired value. After each iteration, several bits of the desired answer are obtained such that after 3 iterations only, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

Figure 5.2 shows the decay of $|x - x_{computed}|/x_{computed}$ versus the iterations. It can be noticed that our algorithm converges monotonically at a much faster rate, while the standard one oscillates. The initial error values of both algorithms are not necessary the same, because our scaling factor is not fixed.

To compare the number of operations performed in each iteration for both the standard CORDIC and HCORDIC, the following assumptions are taken:

Table 5.2: CORDIC iterations for the input $(x, y, z) = (2, 0, 30°)$, where the operation is rotation in the circular mode.

i	x	y	z	μ
Initial	+2.000000e+00	+0.000000e+00	+3.000000e+01	
0	+2.000000e+00	+2.000000e+00	−4.200000e+01	−1
1	+3.000000e+00	+1.000000e+00	−1.543495e+01	1
2	+3.250000e+00	+2.500000e−01	−1.398707e+00	1
3	+3.281250e+00	−1.562500e−01	+5.726310e+00	1
4	+3.291016e+00	+4.882812e−02	+2.149975e+00	−1
5	+3.289490e+00	+1.516724e−01	+3.600647e−01	−1
6	+3.287120e+00	+2.030706e−01	−5.351090e−01	−1
7	+3.288706e+00	+1.773900e−01	−8.749481e−02	1
8	+3.289399e+00	+1.645435e−01	+1.363157e−01	1
9	+3.289078e+00	+1.709681e−01	+2.441002e−02	−1
10	+3.288911e+00	+1.741801e−01	−3.154287e−02	−1
11	+3.288996e+00	+1.725742e−01	−3.566418e−03	1
12	+3.289038e+00	+1.717712e−01	+1.042181e−02	1
13	+3.289017e+00	+1.721727e−01	+3.427696e−03	−1
14	+3.289007e+00	+1.723734e−01	−6.936077e−05	−1
15	+3.289012e+00	+1.722731e−01	+1.679168e−03	1
16	+3.289009e+00	+1.723233e−01	+8.049035e−04	−1
17	+3.289008e+00	+1.723484e−01	+3.677713e−04	−1
18	+3.289007e+00	+1.723609e−01	+1.492053e−04	−1
19	+3.289007e+00	+1.723672e−01	+3.992226e−05	−1
20	+3.289007e+00	+1.723703e−01	−1.471926e−05	−1
21	+3.289007e+00	+1.723687e−01	+1.260150e−05	1
22	+3.289007e+00	+1.723695e−01	−1.058879e−06	−1
23	+3.289007e+00	+1.723691e−01	+5.771310e−06	1
24	+3.289007e+00	+1.723693e−01	+2.356215e−06	−1
Result	+1.997259e+00	+1.046718e−01	+4.112371e−08	
Error	−1.476325e−07	−1.411175e−07	+4.112371e−08	

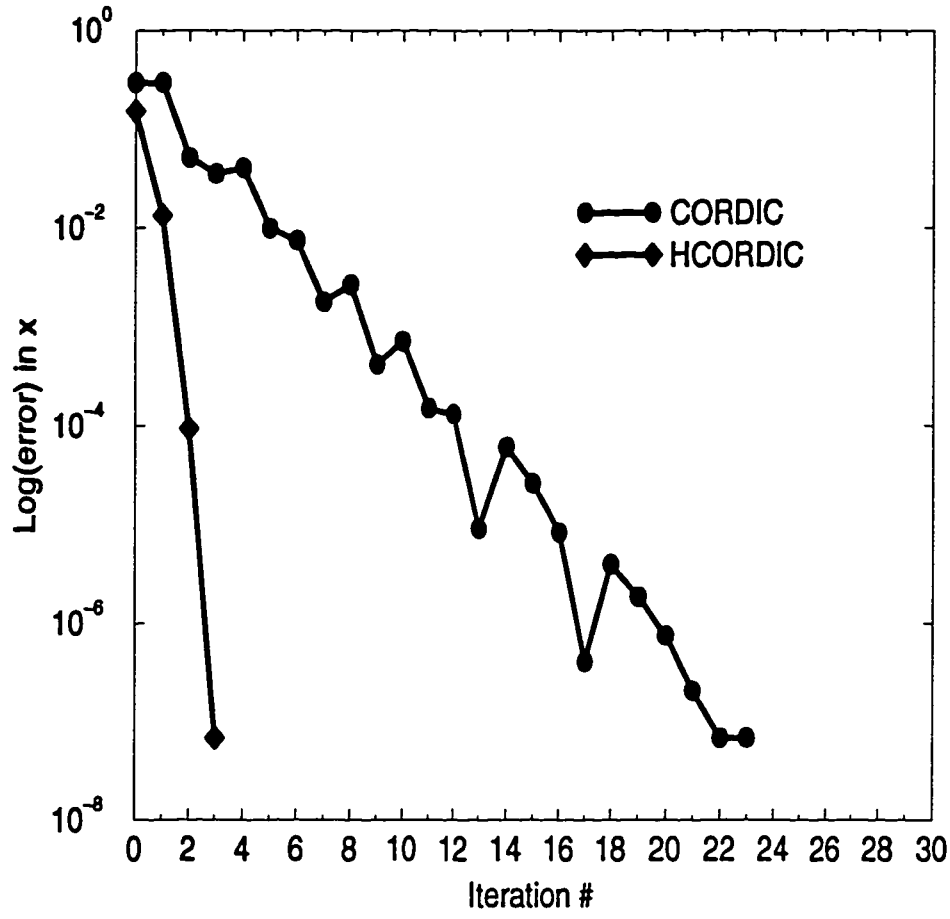

Figure 5.2: Comparison between the standard and the new CORDIC in the convergence rate of x in the rotation operation of the circular mode, where $(x, y, z) = (2, 0, 30°)$.

Table 5.3: HCORDIC iterations for the input $(x, y, z) = (2, 0, 30°)$, where the operation is rotation in the circular mode.

i	x	y	z
Initial	+2.000000e+00	+0.000000e+00	+3.000000e+01
1	+2.000000e+00	+1.016499e−01	+9.044878e−02
2	+1.999851e+00	+1.045796e−01	+6.519415e−03
3	+1.999839e+00	+1.048071e−01	+0.000000e+00
Result	+1.997259e+00	+1.046719e−01	+0.000000e+00
Error	+0.000000e+00	−7.450581e−09	+0.000000e+00

1. We are comparing two different algorithms implemented on the same hardware platform.

2. The ALU is a single-bus system and accordingly no operations are overlapped.

3. Floating point multiplication or addition requires the same number of clock cycles, one FLOPS.

4. Floating point multiply-accumulate requires two FLOPS.

5. Look-up with masking operation are considered as one FLOPS as worst case.

Figure 5.3 shows the number of operations performed in each iteration of the standard and the new CORDIC, respectively. The standard CORDIC requires four FLOPS, one for look-up operation and three for updating the values of x_i, y_i and z_i. While the new CORDIC in the rotation-circular operation requires eight FLOPS, two for look-up operations of δ_i and κ_i, four for updating the values of x_i and y_i, and

two for updating the values of z_i and k_i. However, when θ_i gets smaller than 2^{-12}, δ_i can be approximated as θ_i, and κ_i becomes one. Hence, there is no need to look-up for δ_i and κ_i. This implies that updating k_i is not needed as well. Thus, the total FLOPS required in each iteration drops to six FLOPS, if we still count the masking operation as one FLOPS. For this case, the total number of operations performed in the HCORDIC algorithm is 44, while it is 192 in the standard one, which means a saving of 77%. It should be noticed that using multiplication in HCORDIC can lead to a substantial saving in the descaling step. The saving is because HCORDIC can work with any values of δ.

The reason that the value of κ_i becomes one when θ_i gets smaller than 2^{-12} can be shown from the Taylor series expansion, as follows:

$$\kappa_i = \cos\theta_i = 1 - \frac{\theta_i^2}{2!} + \frac{\theta_i^4}{4!} - \cdots \tag{5.2}$$

When θ_i gets smaller than 2^{-12}, the second term in the right hand side of Equation 5.2 needs at least $n+1$ bits to be represented. This means that the scale factor value will not change after $n/2$ iterations.

5.1.2 Rotation Operation in the Linear Mode

This operation is equivalent to the multiplication operation. The objective is to multiply $x \times y$. Since we have a multiplier in HCORDIC, the number of iterations required is only one. No results will be presented here because it is obvious.

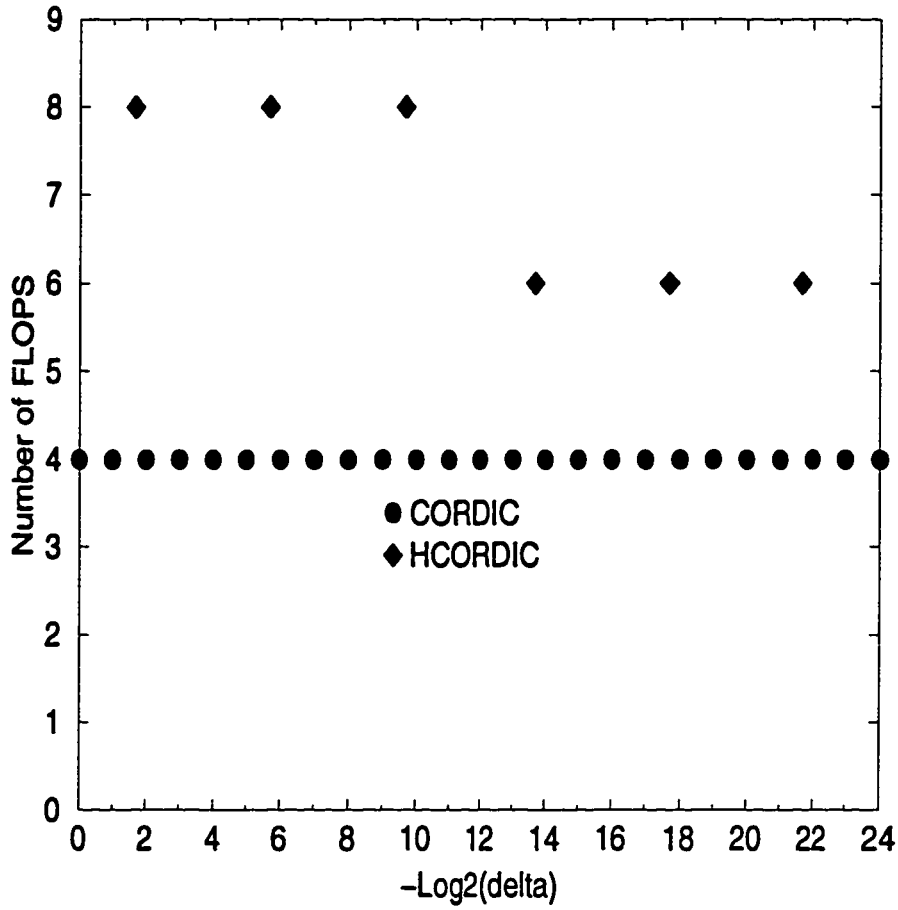

Figure 5.3: Plot of the number of operations at each CORDIC iteration, where the operation is rotation in the circular mode, and $(x, y, z) = (2, 0, 30°)$.

5.1.3 Rotation Operation in the Hyperbolic Mode

This operation is equivalent to vector rotation by a specified "angle". The result presented here is for the following input:

$$(x, y, z) = (1, 0, 1) \tag{5.3}$$

Thus, the objective is to rotate the vector $[1 \;\; 0]^t$ counterclockwise by the "angle" 1.

Table 5.4 shows the iterations performed by the standard CORDIC algorithm. Notice that the variable to be zeroed (z) oscillates around the desired value. After each iteration, one bit of the desired answer is obtained such that after 24 iterations, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

Table 5.5 shows the iterations performed by the new CORDIC algorithm. Notice that the variable to be zeroed (z) monotonically approaches the desired value. After each iteration, several bits of the desired answer are obtained such that after 2 iterations only, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

Table 5.4: CORDIC iterations for the input $(x, y, z) = (1, 0, 1)$, where the operation is rotation in the hyperbolic mode.

i	x	y	z	μ
Initial	+1.000000e+00	+0.000000e+00	+1.000000e+00	
1	+1.000000e+00	+5.000000e−01	+4.506938e−01	−1
2	+1.125000e+00	+7.500000e−01	+1.952810e−01	−1
3	+1.218750e+00	+8.906250e−01	+6.962381e−02	−1
4	+1.274414e+00	+9.667969e−01	+7.042244e−03	−1
4	+1.334839e+00	+1.046448e+00	−5.553932e−02	−1
5	+1.302137e+00	+1.004734e+00	−2.427915e−02	1
6	+1.286438e+00	+9.843881e−01	−8.652875e−03	1
7	+1.278748e+00	+9.743378e−01	−8.402159e−04	1
8	+1.274942e+00	+9.693427e−01	+3.066054e−03	1
9	+1.276835e+00	+9.718328e−01	+1.112927e−03	−1
10	+1.277784e+00	+9.730797e−01	+1.363637e−04	−1
11	+1.278259e+00	+9.737037e−01	−3.519176e−04	−1
12	+1.278022e+00	+9.733916e−01	−1.077770e−04	1
13	+1.277903e+00	+9.732356e−01	+1.429330e−05	1
13	+1.278022e+00	+9.733916e−01	−1.077770e−04	−1
14	+1.277962e+00	+9.733136e−01	−4.674186e−05	1
15	+1.277933e+00	+9.732746e−01	−1.622428e−05	1
16	+1.277918e+00	+9.732551e−01	−9.654905e−07	1
17	+1.277910e+00	+9.732453e−01	+6.663904e−06	1
18	+1.277914e+00	+9.732502e−01	+2.849207e−06	−1
19	+1.277916e+00	+9.732527e−01	+9.418582e−07	−1
20	+1.277917e+00	+9.732538e−01	−1.181616e−08	−1
21	+1.277916e+00	+9.732533e−01	+4.650210e−07	1
22	+1.277917e+00	+9.732535e−01	+2.266024e−07	−1
23	+1.277917e+00	+9.732537e−01	+1.073931e−07	−1
24	+1.277917e+00	+9.732538e−01	+4.778849e−08	−1
Result	+1.543081e+00	+1.175201e+00	+4.778849e−08	
Error	+1.415535e−07	−4.342661e−08	+4.778849e−08	

Table 5.5: HCORDIC iterations for the input $(x, y, z) = (1, 0, 1)$, where the operation is rotation in the hyperbolic mode.

i	x	y	z
Initial	+1.000000e+00	+0.000000e+00	+1.000000e+00
1	+1.000000e+00	+7.615942e−01	+0.000000e+00
Result	+1.543081e+00	+1.175201e+00	+0.000000e+00
Error	−1.192093e−07	+0.000000e+00	+0.000000e+00

5.1.4 Vectoring Operation in the Circular Mode

This operation is equivalent to the coordinate conversion from Cartesian to polar coordinates. The result presented here is for the following input:

$$(x, y, z) = (3, 1, 0) \tag{5.4}$$

Table 5.6 shows the iterations performed by the standard CORDIC algorithm. Notice that the variable to be zeroed (y) oscillates around the desired value. The iteration index of the circular mode has to start from zero according to the convergence sequence. After each iteration, one bit of the desired answer is obtained such that after 24 iterations, the correct answer is obtained within 24 bits of precision. The last two rows of the table display the final results and the difference from the computed values (math functions), respectively.

Table 5.7 shows the iterations performed by the new CORDIC algorithm. Notice that the variable to be zeroed (y) monotonically approaches the desired value. After each iteration, several bits of the desired answer are obtained such that after 8

iterations only, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

5.1.5 Vectoring Operation in the Linear Mode

This operation is perhaps the most important and useful operation performed by the new algorithm. It is equivalent to the division operation of y/x. The result presented here is for the following inputs:

$$(x, y, z) = (1, 3, 0) \tag{5.5}$$

Thus, the objective is to divide 1/3.

Table 5.8 shows the iterations performed by the standard CORDIC algorithm. Notice that the variable to be zeroed (y) oscillates around the desired value. After each iteration, one bit of the desired answer is obtained such that after 24 iterations, the correct answer is obtained within 24 bits of precision. The last two rows of the table display the final results and the difference from the computed values (math functions), respectively.

Table 5.9 shows the iterations performed by the new CORDIC algorithm. Notice that the variable to be zeroed (y) monotonically approaches the desired value. After each iteration, several bits of the desired answer are obtained such that after 8

Table 5.6: CORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the circular mode.

i	x	y	z	μ
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00	
0	+4.000000e+00	−2.000000e+00	+4.500000e+01	1
1	+5.000000e+00	+0.000000e+00	+1.843495e+01	−1
2	+5.000000e+00	−1.250000e+00	+3.247120e+01	1
3	+5.156250e+00	−6.250000e−01	+2.534618e+01	−1
4	+5.195312e+00	−3.027344e−01	+2.176984e+01	−1
5	+5.204773e+00	−1.403809e−01	+1.997993e+01	−1
6	+5.206966e+00	−5.905628e−02	+1.908476e+01	−1
7	+5.207428e+00	−1.837686e−02	+1.863714e+01	−1
8	+5.207500e+00	+1.964658e−03	+1.841333e+01	−1
9	+5.207504e+00	−8.206240e−03	+1.852524e+01	1
10	+5.207512e+00	−3.120787e−03	+1.846929e+01	−1
11	+5.207513e+00	−5.780566e−04	+1.844131e+01	−1
12	+5.207513e+00	+6.933090e−04	+1.842732e+01	−1
13	+5.207513e+00	+5.762622e−05	+1.843432e+01	1
14	+5.207513e+00	−2.602152e−04	+1.843781e+01	1
15	+5.207513e+00	−1.012945e−04	+1.843607e+01	−1
16	+5.207513e+00	−2.183412e−05	+1.843519e+01	−1
17	+5.207513e+00	+1.789605e−05	+1.843475e+01	−1
18	+5.207513e+00	−1.969036e−06	+1.843497e+01	1
19	+5.207513e+00	+7.963507e−06	+1.843486e+01	−1
20	+5.207513e+00	+2.997236e−06	+1.843492e+01	1
21	+5.207513e+00	+5.140998e−07	+1.843495e+01	1
22	+5.207513e+00	−7.274681e−07	+1.843496e+01	1
23	+5.207513e+00	−1.066841e−07	+1.843495e+01	−1
24	+5.207513e+00	+2.037079e−07	+1.843495e+01	−1
Result	+3.162278e+00	+2.037079e−07	+3.217506e−01	
Error	−1.092230e−07	+2.037079e−07	+0.000000e+00	

Table 5.7: HCORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the circular mode.

i	x	y	z
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00
1	+3.281250e+00	+1.562500e−01	+1.570864e+01
2	+3.287964e+00	+1.525879e−02	+1.816905e+01
3	+3.288023e+00	+2.415180e−03	+1.839286e+01
4	+3.288025e+00	+4.083299e−04	+1.842783e+01
5	+3.288025e+00	+5.713099e−05	+1.843395e+01
6	+3.288025e+00	+6.959715e−06	+1.843483e+01
7	+3.288025e+00	+6.883056e−07	+1.843494e+01
8	+3.288025e+00	+1.003609e−07	+1.843495e+01
Result	+3.162278e+00	+1.003609e−07	+1.843495e+01
Error	+0.000000e+00	+1.003609e−07	−1.707547e−06

iterations only, the correct answer is obtained within 24 bits of precision. The last two rows of the table display the final results and the difference from the computed values (math functions), respectively.

Figure 5.4 shows the decay of $|z - z_{computed}|/z_{computed}$ versus the iterations. It can be noticed that our algorithm converges at a much faster rate compared to the standard CORDIC.

To compare the number of operations performed in each iteration for both the standard CORDIC and HCORDIC, the same assumptions made for the rotation operation are adopted.

Figure 5.5 shows the number of operations performed in each iteration of the standard and the new CORDIC, respectively. The standard CORDIC requires four

FLOPS, one for look-up operation and three for updating the values of x_i, y_i and z_i. While the new CORDIC in the vectoring-linear operation requires six FLOPS, one for look-up operation, four for updating the values of x_i and y_i, and one for updating the value of z_i. This is because in the linear mode there is no scale factor, and the value of $\delta_i = \theta_i$. For this case, the total number of operations performed in the HCORDIC algorithm is 36, while it is 96 in the standard one, which means a saving of 62%. Notice that even if the standard CORDIC does not require descaling the saving is considerable. The saving is because HCORDIC can work with any values of δ.

5.1.6 Vectoring Operation in the Hyperbolic Mode

This operation is equivalent to obtaining the radius of a hyperbola knowing its x-y coordinates. The result presented here is for the following input:

$$(x, y, z) = (3, 1, 0) \tag{5.6}$$

Table 5.10 shows the iterations performed by the standard CORDIC algorithm. Notice that the variable to be zeroed (y) oscillates around the desired value. After each iteration, one bit of the desired answer is obtained such that after 24 iterations, the correct answer is obtained within 24 bits of precision. The last two rows of the table display the final results and the difference from the computed values (math functions), respectively.

Table 5.8: CORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the linear mode.

i	x	y	z	μ
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00	
1	+3.000000e+00	−5.000000e−01	+5.000000e−01	1
2	+3.000000e+00	+2.500000e−01	+2.500000e−01	−1
3	+3.000000e+00	−1.250000e−01	+3.750000e−01	1
4	+3.000000e+00	+6.250000e−02	+3.125000e−01	−1
5	+3.000000e+00	−3.125000e−02	+3.437500e−01	1
6	+3.000000e+00	+1.562500e−02	+3.281250e−01	−1
7	+3.000000e+00	−7.812500e−03	+3.359375e−01	1
8	+3.000000e+00	+3.906250e−03	+3.320312e−01	−1
9	+3.000000e+00	−1.953125e−03	+3.339844e−01	1
10	+3.000000e+00	+9.765625e−04	+3.330078e−01	−1
11	+3.000000e+00	−4.882812e−04	+3.334961e−01	1
12	+3.000000e+00	+2.441406e−04	+3.332520e−01	−1
13	+3.000000e+00	−1.220703e−04	+3.333740e−01	1
14	+3.000000e+00	+6.103516e−05	+3.333130e−01	−1
15	+3.000000e+00	−3.051758e−05	+3.333435e−01	1
16	+3.000000e+00	+1.525879e−05	+3.333282e−01	−1
17	+3.000000e+00	−7.629395e−06	+3.333359e−01	1
18	+3.000000e+00	+3.814697e−06	+3.333321e−01	−1
19	+3.000000e+00	−1.907349e−06	+3.333340e−01	1
20	+3.000000e+00	+9.536743e−07	+3.333330e−01	−1
21	+3.000000e+00	−4.768372e−07	+3.333335e−01	1
22	+3.000000e+00	+2.384186e−07	+3.333333e−01	−1
23	+3.000000e+00	−1.192093e−07	+3.333334e−01	1
24	+3.000000e+00	+5.960464e−08	+3.333333e−01	−1
Result	+3.000000e+00	+5.960464e−08	+3.333333e−01	
Error	+0.000000e+00	+5.960464e−08	−2.980232e−08	

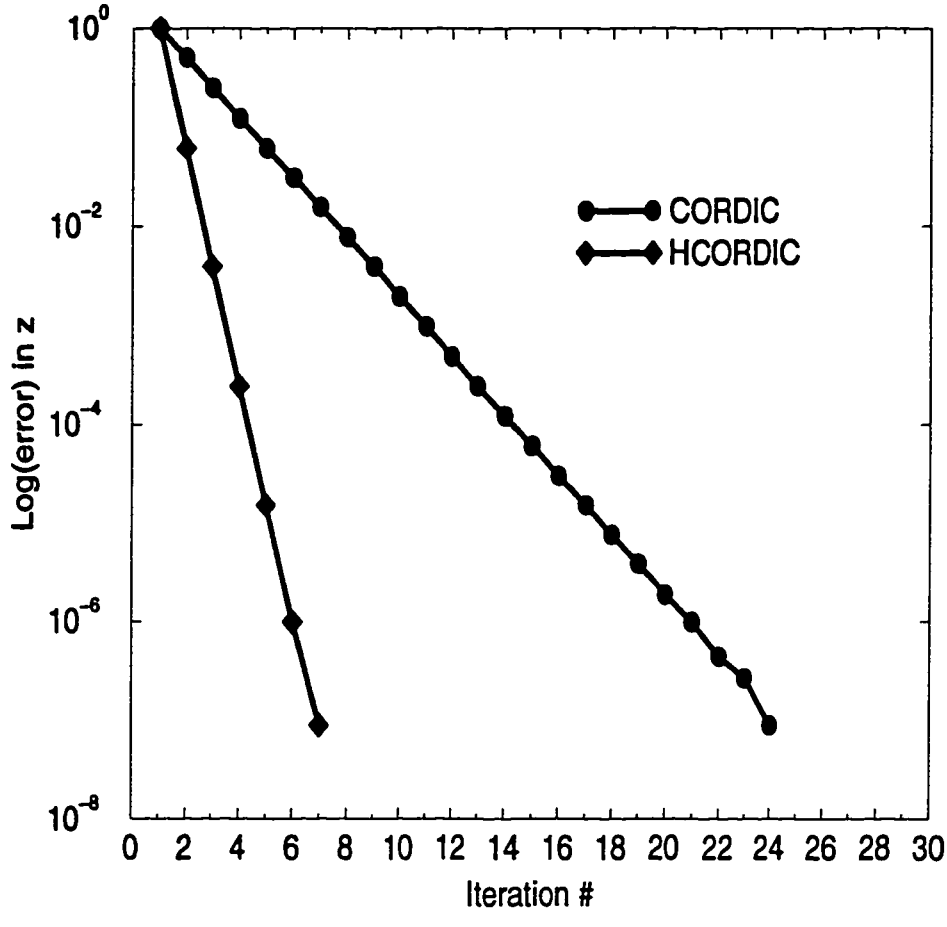

Figure 5.4: Comparison between the standard and the new CORDIC in the convergence rate of z in the vectoring operation of the linear mode, where $(x, y, z) = (3, 1, 0)$.

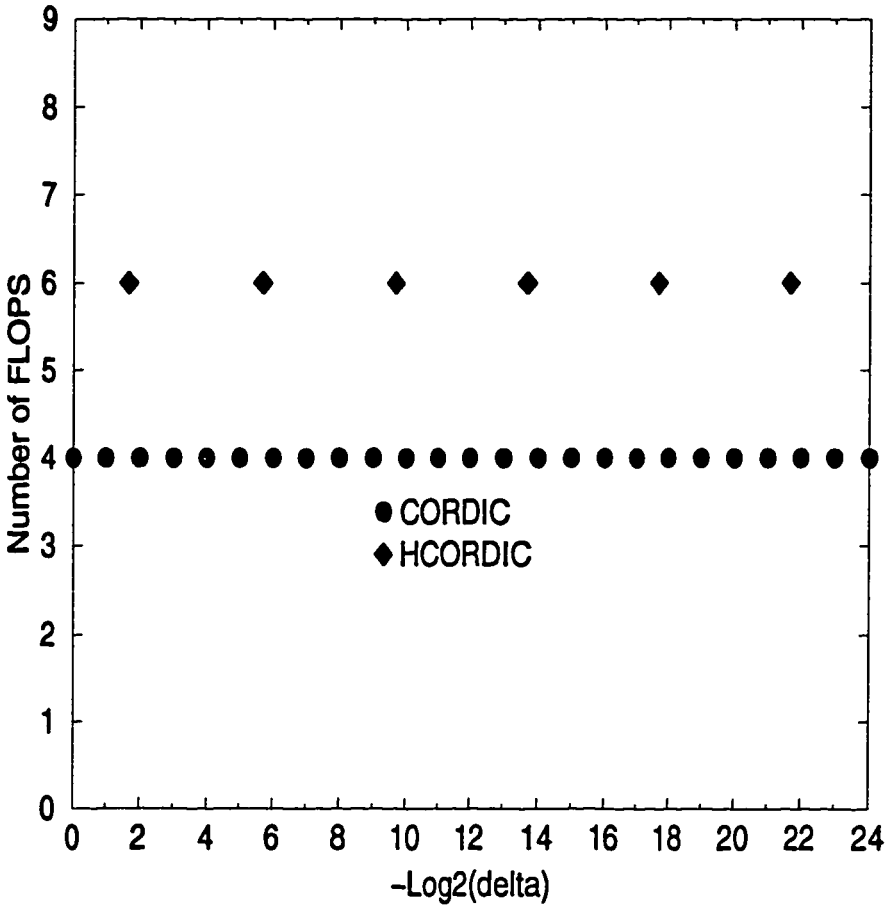

Figure 5.5: Plot of the number of operations at each CORDIC iteration, where the operation is vectoring in the linear mode, and $(x, y, z) = (3, 1, 0)$.

Table 5.9: HCORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the linear mode.

i	x	y	z
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00
1	+3.000000e+00	+1.562500e−01	+2.812500e−01
2	+3.000000e+00	+1.562500e−02	+3.281250e−01
3	+3.000000e+00	+2.441406e−03	+3.325195e−01
4	+3.000000e+00	+2.441406e−04	+3.332520e−01
5	+3.000000e+00	+3.814697e−05	+3.333206e−01
6	+3.000000e+00	+3.814697e−06	+3.333321e−01
7	+3.000000e+00	+5.960464e−07	+3.333331e−01
8	+3.000000e+00	+5.960464e−08	+3.333333e−01
Result	+3.000000e+00	+5.960464e−08	+3.333333e−01
Error	+0.000000e+00	+5.960464e−08	−2.980232e−08

Table 5.11 shows the iterations performed by the new CORDIC algorithm. Notice that the variable to be zeroed (y) monotonically approaches the desired value. After each iteration, several bits of the desired answer are obtained such that after 7 iterations only, the correct answer is obtained within 24 bits of precision. Notice also that the final x and y values are scaled by the scale factor. The last two rows of the table display the descaled results and the difference from the computed values (math functions), respectively.

Table 5.10: CORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the hyperbolic mode.

i	x	y	z	μ
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00	
1	+2.500000e+00	−5.000000e−01	+5.493062e−01	1
2	+2.375000e+00	+1.250000e−01	+2.938933e−01	−1
3	+2.359375e+00	−1.718750e−01	+4.195505e−01	1
4	+2.348633e+00	−2.441406e−02	+3.569690e−01	−1
4	+2.347107e+00	+1.223755e−01	+2.943874e−01	−1
5	+2.343283e+00	+4.902840e−02	+3.256476e−01	1
6	+2.342517e+00	+1.241460e−02	+3.412739e−01	1
7	+2.342420e+00	−5.886307e−03	+3.490865e−01	1
8	+2.342397e+00	+3.263770e−03	+3.451802e−01	−1
9	+2.342390e+00	−1.311224e−03	+3.471334e−01	1
10	+2.342389e+00	+9.762666e−04	+3.461568e−01	−1
11	+2.342389e+00	−1.674781e−04	+3.466451e−01	1
12	+2.342389e+00	+4.043941e−04	+3.464009e−01	−1
13	+2.342389e+00	+1.184580e−04	+3.465230e−01	1
13	+2.342389e+00	−1.674781e−04	+3.466451e−01	1
14	+2.342389e+00	−2.451005e−05	+3.465841e−01	−1
15	+2.342389e+00	+4.697398e−05	+3.465535e−01	−1
16	+2.342389e+00	+1.123196e−05	+3.465688e−01	1
17	+2.342389e+00	−6.639046e−06	+3.465764e−01	1
18	+2.342389e+00	+2.296458e−06	+3.465726e−01	−1
19	+2.342389e+00	−2.171294e−06	+3.465745e−01	1
20	+2.342389e+00	+6.258188e−08	+3.465736e−01	−1
21	+2.342389e+00	−1.054356e−06	+3.465740e−01	1
22	+2.342389e+00	−4.958871e−07	+3.465738e−01	−1
23	+2.342389e+00	−2.166526e−07	+3.465737e−01	−1
24	+2.342389e+00	−7.703537e−08	+3.465736e−01	−1
Result	+2.828428e+00	−7.703537e−08	+3.465736e−01	
Error	+4.238100e−07	−7.703537e−08	+2.980232e−08	

Table 5.11: HCORDIC iterations for the input $(x, y, z) = (3, 1, 0)$, where the operation is vectoring in the hyperbolic mode.

i	x	y	z
Initial	+3.000000e+00	+1.000000e+00	+0.000000e+00
1	+2.718750e+00	+1.562500e−01	+2.890389e−01
2	+2.710205e+00	+7.568359e−03	+3.437811e−01
3	+2.710186e+00	+9.516478e−04	+3.462225e−01
4	+2.710186e+00	+1.245645e−04	+3.465276e−01
5	+2.710186e+00	+1.084054e−05	+3.465696e−01
6	+2.710186e+00	+5.020011e−07	+3.465734e−01
7	+2.710186e+00	+5.776691e−08	+3.465736e−01
Result	+2.828427e+00	+5.776691e−08	+3.465736e−01
Error	+2.384186e−07	+5.776691e−08	+0.000000e+00

5.2 Increasing the Range of Convergence

The range of convergence of the standard CORDIC algorithm is known to be limited. In HCORDIC, the range of convergence is increased, except for the inherent limitation of the hyperbolic mode as explained before. Tables 5.12 shows the result of rotating the vector $[1 \quad 0]^t$ in the circular mode by an angle of 360°. Table 5.13 shows the result of the division operation where dividend is greater than the divisor, 27/3. Table 5.14 shows the result of rotating the vector $[1 \quad 0]^t$ in the hyperbolic mode by an "angle" of 29.

All the results in these tables shows how the new algorithm operated on the increased range, while the standard CORDIC failed to get the correct results.

Table 5.12: Rotating the vector $[1 \ \ 0]^t$ in the circular mode by an angle of 360°.

	x	y	z
Initial	+1.000000e+00	+0.000000e+00	+3.600000e+02
NewCordic	+1.000000e+00	+1.977346e−07	+0.000000e+00
NewError	+0.000000e+00	+2.288907e−08	+0.000000e+00
OldCordic	−1.716362e−01	+9.851604e−01	+4.539899e+00
OldError	−1.171636e+00	+9.851603e−01	+4.539899e+00

Table 5.13: Division operation of 27/3.

	x	y	z
Initial	+3.000000e+00	+2.700000e+01	+0.000000e+00
NewCordic	+3.000000e+00	+2.235174e−08	+9.000000e+00
NewError	+0.000000e+00	+2.235174e−08	+0.000000e+00
OldCordic	+3.000000e+00	+2.400000e+01	+9.999999e−01
OldError	+0.000000e+00	+2.400000e+01	−8.000000e+00

Table 5.14: Rotating the vector $[1 \ \ 0]^t$ in the hyperbolic mode by an "angle" of 29.

	x	y	z
Initial	+1.000000e+00	+0.000000e+00	+2.900000e+01
NewCordic	+1.965667e+12	+1.965667e+12	+0.000000e+00
NewError	+0.000000e+00	+0.000000e+00	+0.000000e+00
OldCordic	+1.693068e+00	+1.366191e+00	+2.788182e+01
OldError	−1.965667e+12	−1.965667e+12	+2.788182e+01

5.3 Concluding Remarks

In this chapter, we presented the simulation results of HCORDIC. These results show clearly that HCORDIC is superior to the standard CORDIC. HCORDIC requires less number of iterations such that the number of iterations in the worst case is around $n/2s$ for the rotation operation and n/s for the vectoring operation, where n is number of bits in the mantissa, and s is the number of leading bits in the mantissa to be scanned in parallel. The total number of operations is also reduced. For the results presented, in the rotation operation of the circular mode, the saving in the number of FLOPS is 77%. It should be noticed that using multiplication in HCORDIC can lead to a substantial saving in the descaling step. In the vectoring operation of the linear mode, the saving is 62%. Furthermore, the range of convergence of HCORDIC is increased.

Chapter 6

Implementation of HCORDIC

Three different techniques are available to us to implement the HCORDIC algorithm:

1. ASIC implementation by standard cell using COMPASS tools.

2. Software implementation using Texas Instruments TMS 320C5x DSP chip.

3. VHDL modeling using V-System/Plus workstation package version 4.6c.

The ASIC option requires detailed gate description of all the components. Consequently, it is a "hardwired" design that is not flexible for future design modifications. The DSP option is appealing for several reasons: TMS chips are widely used in many DSP applications and the software implementations makes design changes easy to effect. However, we encountered some delays in finding a suitable power adapter. We chose the VHDL option mainly because it is a standard hardware description language, flexible for future configuration, and synthesizable to logic level.

6.1 Advantages of VHDL

VHDL stands for VHSIC Hardware Description Language, where VHSIC is the project of Very High Speed Integrated Circuits supported by the US government. VHDL was adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in 1987. The revised standard of VHDL was published in 1993.

VHDL provides a medium for design modeling, simulation and documentation. It has several advantages that can be summarized in the following points [37]:

- Modularity: It supports modular and object programming concepts.
- Hierarchy: It supports three types of hierarchical description:
    1. Top-down, such as system level down to gate level
    2. Bottom-up, such as gate level up to system level
    3. Middle-out, some combination of the first two types.
- Portability: It is a standard machine-independent design description.
- Design abstraction: It supports description of the design in:
    1. Behavioral description, where the design is described by its functionality.
    2. Structural description, where the design is described by its interconnected components.

3. Data flow description, where the design is described by data movement between registers and the operations done on register contents.

- Multiple views: It allows for the same entity to have multiple types of descriptions.

6.2 HCORDIC Design Assumptions

In the hardware design of HCORDIC we made the following assumptions:

1. There exists a host system that translates the application commands into the proper inputs of the HCORDIC processor. Since the input-output requirements of the HCORDIC processor are simple, it can appear to the host system as a slave device.

2. A start signal (*START*) will be required to request the HCORDIC processor to start the computations.

3. The time required for HCORDIC to complete an operation varies depending on the input data. Thus, a function completion signal (*DONE*) should be provided to indicate to the host system valid results. This *DONE* signal could be used as interrupt to the host.

4. The combination of *START* and *DONE* signals makes the HCORDIC processor independent of any particular bus design or system clock speed. Hence,

it can be easily programmed and interfaced to many platforms.

5. Floating point arithmetic is considered here since the target end application of the HCORDIC processor is to act as a "DSP co-processor" that can be interfaced to a modest general-purpose computer to produce a high-performance computer.

6. On-chip SRAM for the look-up tables is used.

7. The number of bits to be scanned in parallel is read as an input parameter, s-bit. This will allow the HCORDIC chip designer to upgrade the design to follow advances in RAM capacity.

8. The HCORDIC contains only one multiplier/accumulator (MAC). Thus, the operations associated with each HCORDIC iteration will be multiplexed in time.

6.3 Interfacing with HCORDIC

The host system will access the HCORDIC processor as if it is a memory module. Figure 6.1 shows the interface signals between the host system and the HCORDIC processor. The interface signals are summarized as follows:

- $CHIP\_SELECT$, a signal that selects HCORDIC and directs the data to its internal registers.

- *ADDRESS_BUS*, the address bus of the HCORDIC which is a subset of the address bus of the host system. As an example, 16-bit width is chosen.
- *DATA_BUS*, a 32-bit data bus where the data and control words are exchanged.
- *START*, a hand-shaking signal where the host informs HCORDIC to start.
- *READ*/$\overline{WRITE}$, a control signal for loading inputs and reading outputs.
- *DONE*, a hand-shaking signal where the HCORDIC informs the host to read the results.

6.4 VHDL Design Methodology

The task at hand is to follow a systematic approach for developing an algorithm-specific processor architecture. This design strategy can be broken down into making decision in the algorithm space and the architecture space. Figure 6.2 shows the main decisions that were considered for both spaces [26].

6.5 Major Design Decisions

Based on the above, we arrived at the following system decisions:

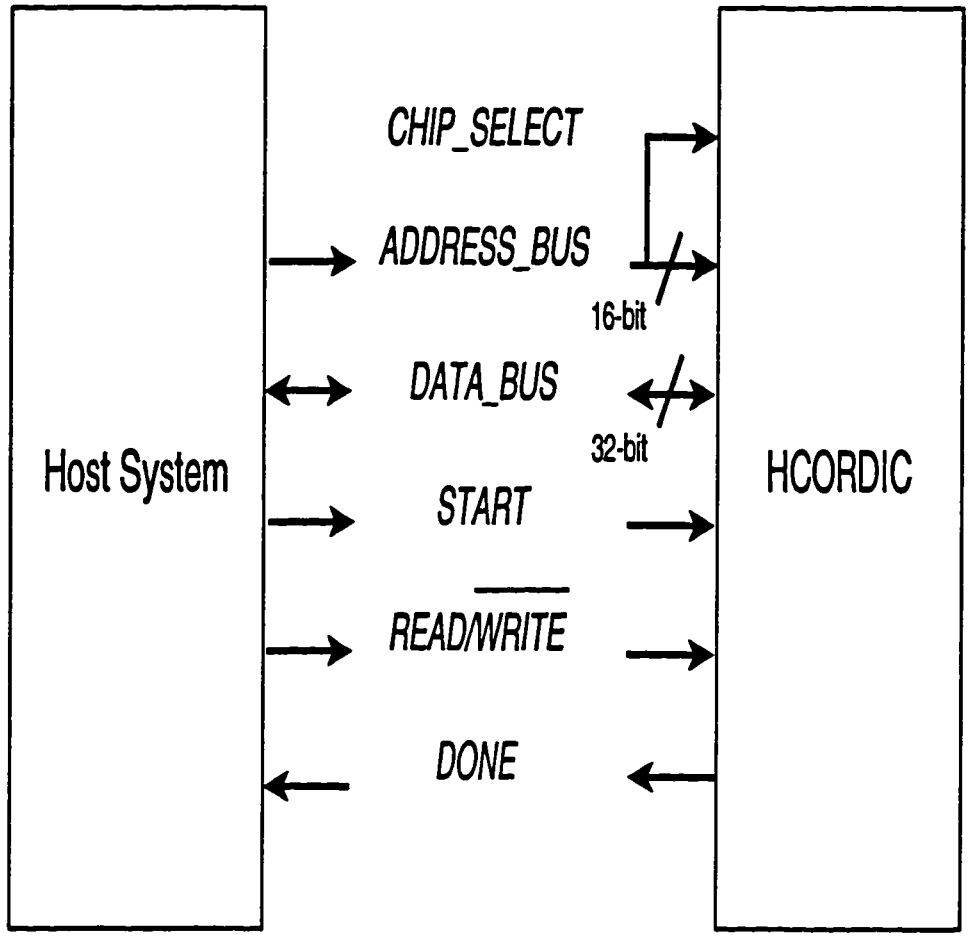

Figure 6.1: Interface between the host system and the HCORDIC processor.

Algorithm Space		Architecture Space
Operations	$\longleftrightarrow$	Functional
Constants and variables	$\longleftrightarrow$	Communication
Data dependencies	$\longleftrightarrow$	Control

Figure 6.2: Mapping of the design issues in the algorithm space and the corresponding architecture space.

1. Some parts of the design are modeled in VHDL, while others are modeled in C to simulate operations that are not supported directly in VHDL. To accomplish this, the foreign language interfacing capability of V-System/Plus VHDL is used.

2. The constants and variables will be in 32-bit floating point format.

3. The ALU will be a floating-point ALU based on a parallel MAC. This ALU is implemented in C since VHDL does not allow us to access different bits and fields of the floating-point data.

4. Data masking, for getting θ and δ, is done in C.

5. Look-up tables are implemented in C.

6. Input/output arithmetic data and control flow are implemented in VHDL.

6.5.1 Program Inputs-Outputs

HCORDIC takes the x, y, z values as data words and s-bit, operation and mode as control words. The outputs are the descaled values of x, y, z, and the $DONE$ signal.

6.5.2 Design Break Down

Figure 6.3 shows the architecture of a bus-based design HCORDIC. There are three common buses for data, address, and control. When the host system se-

lects HCORDIC, the $CHIP\_SELECT$ signal will enable the connections between the internal buses and the system buses. When the computation is done, HCORDIC will send an interrupt signal $DONE$ to the host, so it can read the result. During the computation, the three internal buses will be disconnected from the host buses, and used exclusively by HCORDIC.

We divided our design into several components. Some of the components namely, the controller, RAM, MAC, and Check_Done are implemented as C-functions, while others are written in pure VHDL code. The whole design is controlled by VHDL codes.

The tasks that HCORDIC performs are as follows:

1. Extraction of δ, θ and κ.

2. Updating the values in each iteration.

3. Checking for termination condition.

6.5.3 Interfacing with C-Functions

In our design, the tasks that need accessing the fields of floating point numbers or evaluating trigonometric or hyperbolic functions are done as C-functions. To call a foreign language function in VHDL, we need to declare it as an entity and use it as a component. Thus, two components were implemented as C-functions to perform the following actions:

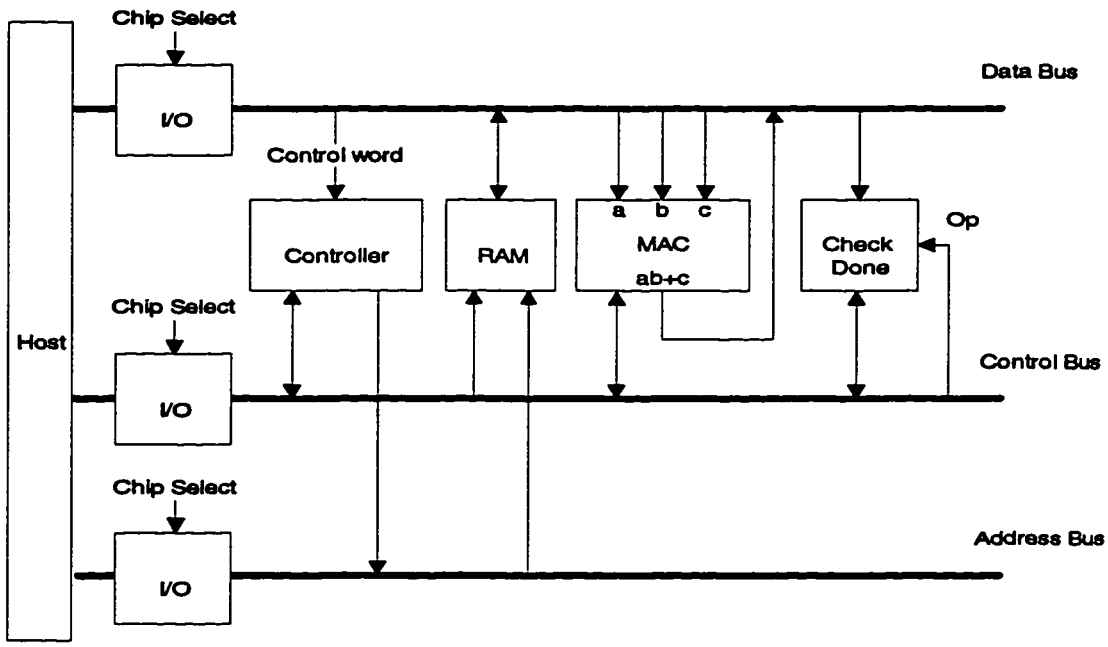

Figure 6.3: Architecture of the HCORDIC processor.

1. Extraction of δ_i, θ_i and κ_i.

2. Detecting the termination condition, y or z approached zero.

Illustrative Example

For the rotation operation, we need to determine θ_i based on the value of z_i. Then we should obtain the corresponding values of δ_i and κ_i. The input parameters that we pass to the C-function are the values of z_i, s-bit, and the mode m, while the returned output values are δ_i, θ_i, and κ_i.

Figure 6.4 shows the declaration of the VHDL entity that performs the rotation operation. Notice that the architecture of this entity works as a pointer to the C-function. In VHDL, the C-function will be treated as a normal component where we map the input and output signals to the input and output ports of the component. In the C-function, we need to perform the following steps:

1. Get Inputs: For the rotation operation, it gets the values of z, s-bit, and m.

2. Sensitization: Make the C-function sensitive for the values of z, s-bit, and m.

3. Drive Outputs: Drive the output ports δ, θ, and κ.

4. Schedule Drivers: Schedule the drivers of δ, θ, and κ.

In between getting input values and driving output values the C-function will mask the value of z and use it as θ to look-up the values of δ and κ.

```
entity rotation_c is
        port(
        iSbits_in: in integer;     -- s
        iM_in: in integer;         -- mode
        fZ_in: in real;            -- z
        fDelta_out: out real;      -- delta
        fTheta_out: out real;      -- theta
        fKappa_out: out real       -- kappa
        );
end;

architecture c_function of rotation_c is
        attribute foreign: string;
        attribute foreign of c_function:
                  architecture is "rotation_init ./rotation.so";
begin
        -- The C code for the foreign architecture
        -- mimics the rotation operation
end c_function;
```

Figure 6.4: Architecture of the rotation operation built in C.

6.5.4 HCORDIC Controller

Options for designing a controller are many. It can be designed using one-hot encoding, PLA, microprogram, or random logic. Our HCORDIC controller is designed as a finite-state machine having three states. Figure 6.5 shows the state diagram of the controller which consists of the following states:

1. READ_INPUT is the initial state where the external or feed back input values are loaded for iterations. The choice of loading either the external new inputs or internal feed back inputs depends on the $START$ signal.

2. UPDATE is the state where the updated values of x, y and z are saved into registers.

3. CHECK_DONE is the state where we check for stopping condition. If the stopping condition is satisfied then HCORDIC will interrupt the host by issuing a $DONE$ signal. The machine will wait in the initial state for new inputs.

The $START$ and $DONE$ signals can asynchronously put the controller into the initial READ_INPUT state.

6.6 VHDL Simulation Results

As was assumed, the HCORDIC has only one MAC for multiplication and addition operations. The delay of the MAC is based on the design of the modified Booth

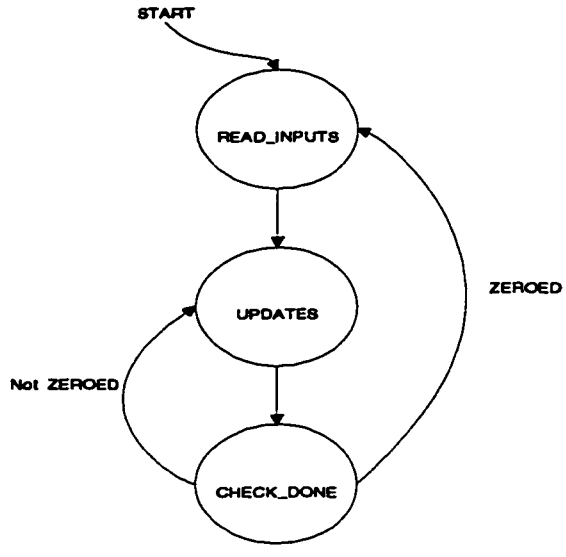

Figure 6.5: State diagram of the HCORDIC controller.

algorithm with carry-look-ahead adder. This MAC delay is around 15 ns, which was obtained from the simulation results of the modified Booth multiplier[1]. The masking with look-up operation takes 15 ns.

The VHDL simulations were performed on SUN SPARCstation 10 using the workstation package of V-System/Plus version 4.6c. The simulations verified the functionality of our VHDL design. A sample result is shown in Figure 6.6, where the operation is vectoring in the linear mode, the (x, y, z) inputs are (4, 24, 0). The final results were ready before 800 ns. Notice in the linear mode, there is only one look-up operation since the value of δ and θ are equal, and no scale factor.

6.7 Concluding Remarks

In this chapter, the implementation issues of HCORDIC were discussed. We have assumed that the HCORDIC processor will operate as a slave device that takes inputs from a master system. The design methodology and major decisions were explained. The HCORDIC was modeled in VHDL with some C-functions. The interface technique of VHDL with C-functions was illustrated.

The HCORDIC design that we aimed at was a bus-based architecture. It is clear that bus-based architecture can save more routing areas than non-bus architecture [7], [21]. The delay estimate that we obtained was promising at this stage. Though,

[1]Private communication with Prof. El-Guibaly and Mr. Al-Fuqaha.

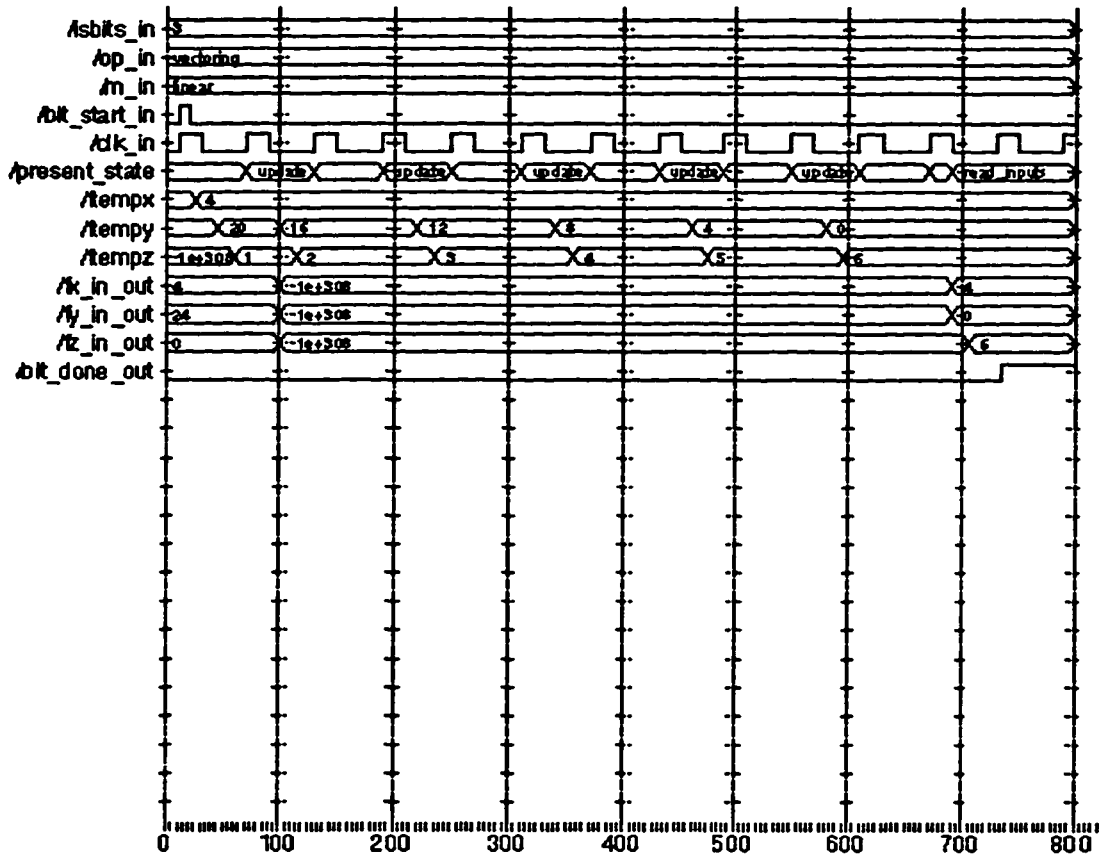

Figure 6.6: Signal wave chart of HCORDIC in the vectoring operation of the linear mode, where the inputs are (x, y, z)=(4, 24, 0).

we can not perform any accurate comparison with other designs till we have an equivalent low-level implementation.

Chapter 7

Conclusions

In this thesis, we studied a very useful algorithm, CORDIC, that has many applications. However, it is inherently plagued with many problems that prevented its widespread acceptance. Many researchers have attempted unsuccessfully to solve some of these problems. The most serious problem was the bit serial nature of the algorithm that makes it slow.

7.1 Main Contributions

We have introduced major modifications in the standard CORDIC algorithm to make it faster and adaptive to input data. The standard algorithm relies on shift and add operations, while the new algorithm relies on multiply/accumulate operations since a parallel multiplier/accumulator is present nowadays in most data

processing chips. The number of iterations of the new CORDIC algorithm is much smaller than those of the standard algorithm. According to the simulation results, we have achieved savings of approximately 60% to 80%, depending on the operation and mode. Hence, we have justifiably named our new algorithm HCORDIC for high-performance CORDIC. The new algorithm requires larger look-up table. For example, in the circular mode, we need three tables for $\tan\theta$, $\tan^{-1}\theta$ and $\cos\theta$. However, advances in VLSI and microprocessor technologies make this a trivial cost.

In this thesis, we have successfully removed all the problems associated with the standard CORDIC. The new algorithm is more efficient than the standard algorithm. Hence, it can compete with existing techniques for evaluating arithmetic functions, such as polynomial techniques.

7.2 Future Work

There are several promising research directions that can be pursued based on the results of this thesis. We have already started some of the future work based on the HCORDIC, they are as follows:

- Three-dimensional CORDIC: The standard CORDIC rotates vectors in two-dimension. Hence, rotating a vector in three dimensional space requires several CORDIC operations. We are considering to develop a three-dimensional HCORDIC, where the rotations can be done in a faster manner. This is useful

in robotics as an example.

- Generalized CORDIC: The standard CORDIC can reduce either y or z to zero, while a more general view of CORDIC is to bring any value of x, y or z to any specified final value. Hence, more functions can be performed, and fewer CORDIC operations are required. As an example of these functions is the evaluation of $\cos^{-1} q$, where q is known, while the rotation angle is unknown [38]. To get $z = \cos^{-1} q$, we can use the rotation operation in the circular mode with input values of $(x, y, z) = (1, 0, 0)$, and the termination condition is $x \rightarrow q$.

- Look-up table optimization: By studying other look-up based algorithms, such as ROM multiplication, we can eliminate some repeated entries of the look-up tables. Also, we can store only $1/x_s$ instead of y_x/x_s.

- Algorithm optimization: We have performed some experiments to find an "optimum" choice of the number of bits to be scanned in parallel, s-bit. The optimization aim is to find the best s-bit value that provides better tradeoff between speed and look-up table size.

- Detailed hardware design: The HCORDIC implementation described in this thesis uses both VHDL and C-language constructs. A real hardware processor of HCORDIC is to be considered in the future.

- Systematic high-level synthesis for testability: A systematic high-level synthesis for testability approach is being investigated. This approach is suitable for us since we can start from a high-level description for our design to make it testable. It is also proven to produce better optimized designs in terms of testability, area, and performance [39], [40].

Appendix A

Mathematical Identities

A.1 Derivation of the Linear Mode

The relationship between x and y and the rotation "angle" can be obtained using the area identities. Consider the area of the small triangle in Figure 2.3. It can be obtained in terms of x and y, or R in the polar coordinate [41].

Theorem

If f is continuous and $f(\theta) \geq 0$ on $[\alpha, \beta]$, where $0 \leq \alpha < \beta \leq 2\pi$, then the area of the region bounded by the graphs of $r = f(\theta)$, $\theta = \alpha$, and $\theta = \beta$ is

$$\text{area} = \int_{\alpha}^{\beta} 1/2[f(\theta)]^2 d\theta = \int_{\alpha}^{\beta} 1/2r^2 d\theta. \tag{A.1}$$

$$\text{area} = xy/2 \qquad \text{from geometry} \tag{A.2}$$

$$= R^2\phi/2 \qquad \text{from Equation A.1} \tag{A.3}$$

Since,

$$R = x = x' \tag{A.4}$$

we get

$$y = x\phi \tag{A.5}$$

A.2 Derivation of the Unified θ_i

The following mathematical identities are used in the derivation of the unified formula of θ for each mode.

$$j = (-1)^{1/2} \tag{A.6}$$

$$a \equiv \lim_{m \to 0} m^{-1/2} \sin(am^{1/2}) \tag{A.7}$$

$$a \equiv \lim_{m \to 0} m^{-1/2} \tan^{-1}(am^{1/2}) \tag{A.8}$$

$$\sinh a \equiv -j \sin(ja) \tag{A.9}$$

$$\cosh a \equiv \cos(ja) \tag{A.10}$$

$$\tanh^{-1} a \equiv -j \tan^{-1}(ja) \tag{A.11}$$

Appendix B

CORDIC Basic Functions

In this appendix, we summarize the basic functions of CORDIC and how to set the proper inputs to obtain the required functions. The basic functions of CORDIC are the trigonometric functions, hyperbolic functions, exponential and logarithmic functions, multiplication, division and square root. Some of the functions can be obtained in more than one way. However, for simplicity we will illustrate one way only for each function. Some functions need more than one CORDIC operation in order to be computed, while others can be computed in a single CORDIC operation.

B.1 Trigonometric Functions

The trigonometric functions that CORDIC can compute in one operation are sin, cos and $\tan^{-1}$. Those functions that need more than one operation are tan, csc, sec, and

Table B.1: Evaluating trigonometric functions in CORDIC.

Function	x_{in}	y_{in}	z_{in}	Operation and Mode	Output	Remark
$\sin\theta$	1	0	θ	rotation-circular	y_{out}	√
$\cos\theta$	1	0	θ	rotation-circular	x_{out}	√
$\tan\theta$	$\cos\theta$	$\sin\theta$	0	vectoring-linear	z_{out}	
$\csc\theta$	$\sin\theta$	1	0	vectoring-linear	z_{out}	
$\sec\theta$	$\cos\theta$	1	0	vectoring-linear	z_{out}	
$\cot\theta$	$\tan\theta$	1	0	vectoring-linear	z_{out}	
$\tan^{-1}\delta$	1	δ	0	vectoring-circular	z_{out}	√

cot. Table B.1 shows how to get these functions by controlling the input values of (x, y, z), the operation and mode. It also shows in which variable of $(x_{out}, y_{out}, z_{out})$ the result is stored. A check mark indicates single-operation function.

B.2 Hyperbolic Functions

The hyperbolic functions that CORDIC can compute in one operation are sinh, cosh and $\tanh^{-1}$. Those functions that need more than one operation are tanh, csch, sech, and coth. Table B.2 shows how to get these functions by controlling the input values of (x, y, z), the operation and mode. It also shows in which variable of $(x_{out}, y_{out}, z_{out})$ the result is stored. A check mark indicates single-operation function.

Table B.2: Evaluating hyperbolic functions in CORDIC.

Function	x_{in}	y_{in}	z_{in}	Operation and Mode	Output	Remark
$\sinh\theta$	1	0	θ	rotation-hyperbolic	y_{out}	√
$\cosh\theta$	1	0	θ	rotation-hyperbolic	x_{out}	√
$\tanh\theta$	$\cos\theta$	$\sin\theta$	0	vectoring-linear	z_{out}	
csch θ	$\sin\theta$	1	0	vectoring-linear	z_{out}	
sech θ	$\cos\theta$	1	0	vectoring-linear	z_{out}	
$\coth\theta$	$\tan\theta$	1	0	vectoring-linear	z_{out}	
$\tanh^{-1}\delta$	1	δ	0	vectoring-hyperbolic	z_{out}	√

B.3 Exponential and Logarithmic Functions

Exponential function can be computed in one CORDIC operation, while the logarithmic function requires more than one operation.

$$\exp a = \sinh a + \cosh a \tag{B.1}$$

$$\ln w = 2 \times \tanh^{-1}(y/x) \tag{B.2}$$

where,

$$x = w + 1 \tag{B.3}$$

$$y = w - 1 \tag{B.4}$$

Table B.3 shows how to get these functions by controlling the input values of (x, y, z), the operation and mode. It also shows in which variable of $(x_{out}, y_{out}, z_{out})$ the result is stored. A check mark indicates single-operation function.

Table B.3: Evaluating exponential and logarithmic functions in CORDIC.

Function	x_{in}	y_{in}	z_{in}	Operation and Mode	Output	Remark
$\exp a$	1	1	a	rotation-hyperbolic	x_{out} or y_{out}	$\surd$
$\ln w$	2	0	$\tanh^{-1}\frac{w-1}{w+1}$	rotation-linear	y_{out}	

Table B.4: Evaluating multiplication and division operations in CORDIC.

Function	x_{in}	y_{in}	z_{in}	Operation and Mode	Output
$a \times b$	a	0	b	rotation-linear	y_{out}
a/b	b	a	0	vectoring-linear	z_{out}

B.4 Multiplication and Division

The multiplication and division operations can be computed in one CORDIC operation. Table B.4 shows how to get these functions by controlling the input values of (x, y, z), the operation and mode. It also shows in which variable of $(x_{out}, y_{out}, z_{out})$ the result is stored.

B.5 Square Root

The square root operation can be computed in CORDIC by proper adjustment of the input values. Table B.5 shows how to get these functions by controlling the input values of (x, y, z), the operation and mode. It also shows in which variable of

Table B.5: Evaluating square root operation in CORDIC.

Function	x_{in}	y_{in}	z_{in}	Operation and Mode	Output
$\sqrt{w}$	$w+1/4$	$w-1/4$	–	vectoring-hyperbolic	x_{out}

$(x_{out}, y_{out}, z_{out})$ the result is stored.

Appendix C

Software Pseudocode

The main blocks of the HCORDIC algorithm are as follows:

- InputChecking:

  This function checks if the entered values are legal or not. It returns TRUE if there is an illegal input, or FALSE if not.

- Function MaskGeneration:

  This function takes the number of bits to be scanned and generates the proper masks of the leading s bits. There are two kinds of masks, one with leading s bits of zeros, and another with leading s bits of ones.

  iMaskANDing = $\overbrace{1\cdots1}^{s}\underbrace{0\cdots0}_{23\text{-}s}$; all zeros except the leading s bits.

  iMaskORing = $\overbrace{0\cdots0}^{s}\underbrace{1\cdots1}_{23\text{-}s}$; all ones except the leading s bits.

- Function TooSmall:

  This function checks if the values to be zeroed (fTestValue) is equal to zero or small enough, around 10^{-8} in 32-bit floating point format. It returns TRUE or FALSE accordingly.

- Function Rotation:

  This function takes the mode, and the values of z to generate the proper values of δ, θ and κ.

- Function Vectoring:

  This function takes the mode, and the values of x and y to generate the proper values of δ, θ and κ.

- Function ErrorCalculation:

  This function compares the final outputs of CORDIC with those computed from C math functions. The computed functions are according to the operations and modes of CORDIC.

- Function HCORDIC:

  This is the main function of the new CORDIC algorithm. It takes the control inputs (operation and mode), with the data inputs (x, y, and z) to generate the desired output values of x, y, and z.

The psudocode of the algorithm is presented below. It shows the algorithm in an abstract manner.

```
#define iERROREXP 10^-8

/* IEEE 32-bit floating pointing format. */
struct floatFields
        {
        unsigned iSign: 1;
        unsigned iExponent: 8;
        unsigned iMantissa: 23;
        };

InputChecking INPUT (int iOperation, iMode; float fX, fY)
RETURN (int iError)
{
        if (iOperation in [Vectoring, Rotation])
                print(iOperation type);
        else
                {
                print(Error_Msg);
                return TRUE;
                }
        if (iMode in [Circular, Linear, Hyperbolic])
                print(iMode type);
        else
                {
                print(Error_Msg);
                return TRUE;
                }

        if ((iMode == Hyperbolic) and ( fY >= fX ))
                {
                print(Error_Msg);
                return TRUE;
                }

        return FALSE;  /* no errors */
}

MaskGeneration INPUT (int iSbits)
RETURN (int iMaskANDing, iMaskORing)
{
```

```
        iMaskANDing = all zeros except the leading iSbits are ones;

        iMaskORing = all ones except the leading iSbits are zeros;
}

TooSmall INPUT (float fTestValue)
RETURN (int iTestResult)
{
        if ( (fTestValue == 0) or
             (iExponent of fTestValue < iERROREXP) )
                return TRUE;
        else
                return FALSE;
}

Rotation INPUT (int iMode, float fZ)
RETURN (float fDelta, fTheta, fKappa)
{
        fMaskedZ = fZ AND iMaskANDing;
        if (iExponent of fZ >= -12)
                fTheta = -1 * fMaskedZ;
        else
                fTheta = -1 * fZ;

        if (iMode == Circular)
                {
                fDelta= tan (fTheta);
                fKappa= cos(fTheta);
                }
        if (iMode == Linear)
                {
/* in linear mode, we have a multiplier so take the whole fZ */
                fTheta= -1 * fZ;
                fDelta=fTheta;
                fKappa=1;
                }
        else if (iMode == Hyperbolic)
                {
                fDelta= tanh(fTheta);
                fKappa= cosh(fTheta);
                }
```

```
}

Vectoring INPUT (int iMode, float fX, fY)
RETURN (float fDelta, fTheta, fKappa)
{
        fMaskedX = fX AND iMaskANDing;
        fMaskedY = fY OR iMaskORing;
        if (fX <= fY) and (iMode != Linear)
                fDelta = 1;
        else
                {
                fDelta = fMaskedY/fMaskedX;
                fMaskedDelta = fDelta AND iMaskANDing;
                fDelta = fMaskedDelta;
                }

        if (iMode == Circular)
                {
                fTheta= atan(fDelta);
                fKappa= cos(fTheta);
                }
        else if (iMode == Linear)
                {
                fTheta= fDelta;
                fKappa= 1;
                }
        else if (iMode == Hyperbolic)
                {
                fTheta= atanh(fDelta);
                fKappa= cosh(fTheta);
                }
}

ErrorCaculation INPUT (int iOperation, iMode;
                       float fTrueX, fTrueY, fTrueZ)
RETURN (float fErrorX, fErrorY, fErrorZ)
{
        for (operation = iOperation, mode = iMode)
                compute fComputedX, fComputedY, fComputedZ;

        fErrorX = fTrueX - fComputedX;
```

```
        fErrorY = fTrueY - fComputedY;
        fErrorZ = fTrueZ - fComputedZ;
}

HCORDIC INPUT (int iOperation, iMode; float fX, fY, fZ)
RETURN (float fTrueX, fTrueY, fTrueZ)
{
        if (InputChecking(iOperation, iMode, fX, fY) == TRUE)
                exit;

        MaskGeneration(iSbits);

        Loop until ( (TooSmall(fY) and Vectoring)
                or   (TooSmall (fZ) and Rotation) )
                {
                if (iMode == Vectoring)
                        Vectoring(iMode, fX, fY);
                else  /* Rotation */
                        Rotation(iMode, fZ);

                /* CORDIC iterations */
                fUpdatedX =  fX + iMode  * fY * fDelta ;
                fUpdatedY =  fY - fX * fDelta;
                fUpdatedZ =  fZ + fTheta;
                fUpdatedK =  fK * fKappa;

                fX = fUpdatedX;
                fY = fUpdatedY;
                fZ = fUpdatedZ;
                fK = fUpdatedK;
                } /* end loop */

        if (descaling is needed)
                {
                fTrueX = fX * fK;
                fTrueY = fY * fK;
                }
        print(fTrueX, fTrueY, fTrueZ);
        ErrorCaculation(iOperation, iMode, fTrueX, fTrueY, fTrueZ);
}
```

Bibliography

[1] J. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sept. 1959.

[2] J. S. Walther, "A Unified Algorithm for Elementary Functions", in *Proc. of the 1971 Spring Joint Computer Conference.* Atlantic city, N.J., USA, 1971, pp. 379–385.

[3] J. McLeod, "Spec Puts Video Conferencing Closer to Desktop", *Electronics*, vol. 68, no. 2, pp. 1, Jan. 1995.

[4] A. Noore, S. Nestor, and M. Lawson, "Computer-Based Multimedia Video Conferencing System", *IEEE Transactions on Consumer Electronics*, vol. 39, no. 3, pp. 587–592, Aug. 1993.

[5] J. McLeod, "Companies to Develop Cable Modem", *Electronics*, vol. 66, no. 23, pp. 14, Dec. 1993.

[6] J. Puttre, "Safe Home: High Speed Internet Video Access Now, Despite Cable Modem Limits?", *Advanced Imaging*, vol. 11, no. 4, pp. 3, Apr. 1996.

[7] G. Haviland and A. Tuszynski, "A CORDIC Arithmetic Processor Chip", *IEEE Transactions on Computers*, vol. C-29, no. 2, pp. 68–79, Feb. 1980.

[8] H. Hahn, D. Timmermann, B. Hosticka, and B. Rix, "A Unified and Division-Free CORDIC Argument Reduction Method with Unlimited Convergence Domain Including Inverse Hyperbolic Functions", *IEEE Transactions on Computers*, vol. 43, no. 11, pp. 1339–1344, Nov. 1994.

[9] X. Hu, R. Harber, and S. Bass, "Expanding the Range of Convergence of the CORDIC Algorithm", *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 13–21, Jan. 1991.

[10] H. M. Ahmed, "Efficient Elementary Function Generation with Multipliers", in *Proceedings of the 9th Symposium on Computer Arithmetic.* Santa Monica, CA, USA, Sept. 6-8 1989, pp. 52–59.

[11] D. Timmermann, H. Hahn, and B. Hosticka, "Modified CORDIC Algorithm with Reduced Iterations", *Electronics Letters*, vol. 25, no. 15, pp. 950–951, July 1989.

[12] E. Antelo, J. Bruguera, and E. Zapata, "Unified Mixed Radix 2-4 Redundant CORDIC Processor", *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 1068–1073, Sept. 1996.

[13] H. Dawid and H. Meyr, "The Differential CORDIC Algorithm: Constant Scale Factor Redundant Implementation without Correcting Iterations", *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 307–318, Mar. 1996.

[14] J. Duprat and J. Muller, "The CORDIC Algorithm: New Results for Fast VLSI Implementation", *IEEE Transactions on Computers*, vol. 42, no. 2, pp. 168–178, Feb. 1993.

[15] Y. Hu, "CORDIC-Based VLSI Architectures for Digital Signal Processing", *IEEE Signal Processing Magazine*, vol. 9, no. 3, pp. 16–35, July 1992.

[16] S. Nikolaidis, D. Metafas, and C. Goutis, "CORDIC Based Pipeline Architecture for All-Pass Filters", in *Proc. IEEE International Symposium on Circuits and Systems*. Chicago, IL, USA, 3-6 April 1993, pp. 1917–1920.

[17] M. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD", *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 725–740, June 1990.

[18] S. Hsiao and J. Delosme, "Parallel Singular Value Decomposition of Complex Matrices Using Multidimensional CORDIC Algorithms", *IEEE Transactions on Signal Processing*, vol. 44, no. 3, pp. 685–697, Mar. 1996.

[19] C. Krieger and B. Hosticka, "Inverse Kinematics Computations with Modified CORDIC Iterations", *IEE Proceedings: Computers and Digital Techniques*, vol. 143, no. 1, pp. 87–92, Jan. 1996.

[20] A. Dhar and S. Banerjee, "Doppler Ultrasonograph with CORDIC Based Spectrum Analyzer for Blood Flow Velocity Estimation", in *Proceedings of the 1st 1995 Regional Conference IEEE Engineering in Medicine and Biology Society.* New Delhi, India, 1995, pp. 2.107–2.108.

[21] D. Timmermann, H. Hahn, B. Hosticka, and G. Schmidt, "A Programmable CORDIC Chip for Digital Signal Processing Applications", *IEEE Journal of Solid-State Circuits*, vol. 26, no. 9, pp. 1317–1321, Sept. 1991.

[22] G. Hekstra and E. Deprettere, "Floating Point CORDIC", in *Proceedings of the 1993 IEEE 11th Symposium on Computer Arithmetic.* Windsor, Ont. Can., 1993, pp. 130–137.

[23] A. Despain, "Fourier Transform Computers Using CORDIC Iterations", *IEEE Transactions on Computers*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.

[24] F. El-Guibaly, A. Almulhem, A. Sabaa, and D. Shpak, "A High Speed CORDIC Algorithm", in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing.* Victoria, B. C., May 1995, pp. 485–488.

[25] H. M. Ahmed, *Signal Processing Algorithms and Architectures*, PhD thesis, Information Systems Laboratory, Stanford University, June 1982.

[26] S. Leung and M. Shanblatt, *ASIC System Design with VHDL: A Paradigm*, Kluwer Academic Publishers, Boston, USA, 1989.

[27] B. C. McKinney and F. El-Guibaly, "A Multiple-Access Pipeline Architecture for Digital Signal Processing", *IEEE Transactions on Computers*, vol. 37, no. 3, pp. 283–290, Mar. 1988.

[28] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC Architectures for Fast VLSI Filtering", in *Proc. IEEE International Conference on ASSP.* Florida, USA, April 1984, pp. 41.A.6.1–41.A.6.4.

[29] F. El-Guibaly, "Matrix Triangularization Using Givens Rotations", in *Proc. Can. Conf. on Electrical and Computer Engineering.* Vancouver, Nov. 1988, pp. 525–528.

[30] G. Golub and C.Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, USA, 1991.

[31] Y. Hu and S. Naganathan, "An Angle Recoding Method for CORDIC Algorithm Implementation", *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 99–102, Jan. 1993.

[32] H. Lo, H. Lin, and K. Yang, "A New Method of Implementation of VLSI CORDIC for Sine and Cosine Computation", in *Proceedings of the 1995 IEEE International Symposium on Circuits and Systems.* Seattle, WA, USA, July 1995, pp. 1984–1987.

[33] F. El-Guibaly and A. Sabaa, "A High Speed CORDIC Algorithm", in *Advanced Signal Processing Algorithms, Architectures, and Implementations.* Denver, Colorado, 6-8 Aug. 1996, vol. 6, pp. 51–54.

[34] D. Wong and M. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations", *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 981–995, Aug. 1992.

[35] F. El-Guibaly, "Analysis of the CORDIC Algorithm", in *Cyprus Intl. Conf. Comp. Appl. to Engineering Systems.* Nicosia, Cyprus, July 1991, pp. 12–17.

[36] S. Hsiao and J. Delosme, "Householder CORDIC Algorithms", *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 990–1001, Aug. 1995.

[37] Z. Navabi, *VHDL Analysis and Modeling of Digital Systems*, McGraw-Hill, Inc, Hightstown, N.J. USA, 1993.

[38] C. Mazenc, X. Merrheim, and J. Muller, "Computing Functions $\cos^{-1}$ and $\sin^{-1}$ Using CORDIC", *IEEE Transactions on Computers*, vol. 42, no. 1, pp. 118–122, Jan. 1993.

[39] C. Chen and D. Saab, "A Novel Behavioral Testability Measure", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, pp. 1960–1970, Dec. 1993.

[40] C. Chen, T. Karnik, and D. Saab, "Structural and Behavioral Synthesis for Testability Techniques", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 6, pp. 777–785, June 1994.

[41] E. Swokowski, *Calculus*, PWS-KENT Publishing Company, Boston, USA, 1991.

Vita

- Ahmad Nour Al-Islam Ayoub Al-Sawi

  Also know as, Nien-Tse Sui

- Received Bachelor of Science degree in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, in July 1995.

- Completed Master of Science degree in Computer Engineering from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia, in January 1997.