Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors

Kishore Kota, *Student Member, IEEE,* and Joseph R. Cavallaro, *Member, IEEE*

Abstract—**The coordinate rotation digital computer (CORDIC) algorithm is used in numerous special-purpose systems for real-time signal processing applications. It is desirable to use fixed-point CORDIC units in such systems, since the low complexity, compared to floating-point, allows multiple CORDIC units and additional hardware to be integrated on the same chip. However, an analysis of fixed-point CORDIC in the Y-reduction mode, which allows computation of the inverse tangent function, shows that unnormalized input values can result in large numerical errors. This paper describes two approaches to tackle the numerical accuracy problem. The first approach builds on a fixed-point CORDIC unit and eliminates the problem by including additional hardware for normalization. This paper presents a method to integrate the normalization operation with the CORDIC iterations for efficient implementation in $O(n^{1.5})$ hardware. The second solution to the accuracy problem is to use a floating-point CORDIC unit but reduce the implementation complexity by using a hybrid architecture. We present arguments to support the use of such an architecture in certain special purpose arrays.**

Index Terms—**CORDIC, error analysis, error reduction, hardware complexity, VLSI.**

I. Introduction

COORDINATE rotation digital computer (CORDIC) is an arithmetic technique that allows efficient computation of a variety of transcendental functions. It was originally developed by Volder [22] to solve trigonometric problems that arise in navigation applications. Walther [23] later developed it into a unified algorithm to compute a variety of transcendental functions. The CORDIC algorithm is attractive from a hardware point of view since it uses only primitive operations like shifts and additions to implement relatively complex functions like sine, cosine, tangent, arctangent, sinh, cosh, tanh, arctanh, ln, and exp. An early fixed-point CORDIC chip capable of operating in all the modes was designed by Haviland and Tuszynski [12]. A number of real-time signal processing, robotics and image processing algorithms have been developed to make efficient use of CORDIC in special-purpose arrays. Interest in using CORDIC to implement special purpose arrays [4], [20] has spawned off a large body of work to improve the algorithm. Ercegovac and Lang [11] used on-line arithmetic techniques to allow extensive pipelining and thereby speed up CORDIC. Hu, Harber, and Bass [15] have developed techniques to increase the range of convergence of the CORDIC algorithms by performing additional iterations. Redundant arithmetic techniques have been used to speed up CORDIC [11], [21]. Extensions of CORDIC to N-dimensions have been proposed by Hsiao and Delosme [14]. A floating-point implementation of CORDIC with modifications to better integrate the various modes was reported by de Lange and Deprettere [6]. A general framework to compute elementary functions using discrete bases was developed by Muller [19]. Work on a floating-point implementation of CORDIC was done by Johnsson and Krishnaswamy [17]. Our interest in CORDIC is motivated by the design of a special-purpose processor array for computing the singular value decomposition (SVD) of a matrix. CORDIC has proved useful in SVD [4], QR Decomposition [2], FFT [9], and filtering algorithms [8].

Although the essential idea of CORDIC remains the same, different implementation choices have a large impact on the area and time complexity of a CORDIC chip. In particular, choice of a fixed-point or a floating-point representation of numbers can determine the ease of implementation. Floating-point implementations are significantly more complex than fixed-point. This complexity arises primarily due to the need to normalize after every step and to handle exceptions. The additional accuracy, however, may not be necessary in some applications. Several higher-level iterative algorithms can tolerate small numerical errors and provide accurate results even if the CORDIC methods used at an intermediate stage are not very accurate. The SVD algorithm proposed by Brent, Luk, and Van Loan [3], for example, was shown to be tolerant to small errors in the angle calculations required at each iteration [18]. The use of fixed-point in such systems can imply lower area and time costs, allowing a single chip to integrate CORDIC and additional hardware to implement certain application specific algorithms. Thus, there is a tradeoff between implementation costs and numerical accuracy, which we illustrate in this paper by presenting two implementations of CORDIC. We obtained all the results as part of our effort to implement CAPE (CORDIC array processor element) [13], a processor to be used in a systolic array, to compute the SVD for real-time applications. The results, however, are general and are applicable to other systems using CORDIC.

In this paper, we are concerned with numerical errors that occur in a fixed-point implementation of CORDIC. In one of the earliest papers that addresses this issue, Walther [23] concluded that the errors that occur in some of the

Manuscript received October 15, 1991; revised May 12, 1992 and September 14, 1992. This work was supported in part by the National Science Foundation under Research Initiation Award MIP-8909498. Use of the Connection Machine was provided by the Center for Research on Parallel Computation under NSF Cooperative Agreement CCR-9120008 with support from the Keck Foundation and Thinking Machines Corporation.

The authors are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251.

IEEE Log Number 9208768.

CORDIC variables are bounded. He added that the effect of these errors on the function being computed depends on the sensitivity of that function. Duryea and Pottle [10] concluded that errors in computation of sine, cosine, and inverse tangent using CORDIC are all bounded. However, that study did not consider some factors, which have a profound influence on the accuracy of the computed functions. Hu [16] provided an accurate description of the errors encountered in all modes of CORDIC operation. However, no results on the inverse tangent computation are provided. Since the accuracy of inverse tangent calculations has a significant impact on the system we are designing, we studied the problem and showed that the errors in inverse tangent computations using fixed-point CORDIC can be large enough to be unacceptable.

Although the result we obtained was pessimistic, our result also showed that these errors can be controlled by enforcing certain lower bounds on the input variables. This takes the form of normalization of the input values prior to CORDIC iterations. Use of preexisting techniques to solve this problem, however, did not result in an efficient implementation. We developed a normalization method that reduced the area costs and eliminated the time penalty incurred. This scheme integrates normalization with the CORDIC iterations and allows for an efficient implementation of fixed-point CORDIC, while restoring the numerical accuracy that fixed-point CORDIC was believed to achieve.

The above problem with inverse tangent calculations does not occur in a floating-point implementation. This is due to the implicit assumptions made about normalization. Floating-point numbers are always assumed to be in a normalized form. To gain the advantages of a floating-point CORDIC processor without the associated complexity, we introduce the notion of a CORDIC processor, where certain data paths are fixed-point while others are floating-point. We show that such an architecture can be advantageous in certain applications.

The rest of the paper is structured as follows. Section II briefly describes the CORDIC algorithm. Section III discusses the error analysis technique through application to two modes of CORDIC, *viz.* rotation and vectoring modes. This section also introduces the notation to be used in the rest of the paper. In addition this section provides the experimental results obtained for the numerical error. Section IV discusses the area-time tradeoffs of previous techniques to normalization and presents a new approach. Section V describes a hybrid architecture for CORDIC that incorporates the features of both fixed-point and floating-point CORDIC implementations. Section VI gives an analysis of the new architecture. Finally we present our conclusions.

II. The CORDIC Arithmetic Technique

The basic CORDIC iterations for circular mode of operation, as specified by Volder [22], are

$$x_{i+1} = x_i + \delta_i y_i 2^{-i} \tag{1}$$

$$y_{i+1} = y_i - \delta_i x_i 2^{-i} \tag{2}$$

$$z_{i+1} = z_i + \delta_i \alpha_i \tag{3}$$

TABLE I
CORDIC IN THE CIRCULAR MODE

Operation	Initial Values for Variables	Final Values for Variables	Functions Computed
Z-Reduction	$x_0 = X$	$x_n = (x_0 \cos\theta + y_0 \sin\theta)K_n$	Vector Rotations, sine, cosine
	$y_0 = Y$	$y_n = (-x_0 \sin\theta + y_0 \cos\theta)K_n$	
	$z_0 = \theta^a$	$z_n = 0$	
Y-Reduction	$x_0 = X$	$x_n = K_n\sqrt{x_0^2 + y_0^2}$	Inverse Tangents
	$y_0 = Y$	$y_n = 0$	
	$z_0 = 0$	$z_n = \theta = \tan^{-1}(y_0/x_0)$	

[a] This initialization corresponds to a modification of the z-iteration to be discussed in Section III-A. The initialization that corresponds to (3) and (6) is $z_0 = 0$ with the resultant $z_n = \theta$.

where x_i, y_i, and z_i are the states of the variables x, y, and z at the start of the ith iteration, $0 \le i < n$, and $\delta_i \in \{-1, +1\}$. The angles, $\alpha_i = \tan^{-1}(2^{-i})$, are precomputed and stored in a table of angles. All results in this paper have been obtained for the circular mode of CORDIC. The initialization of all variables for the different functions implemented in this mode are presented in Table I.

The CORDIC iterations given by (1) and (2) can be rewritten to convey the idea of a vector rotation. At an iteration i the output vector $\boldsymbol{v}_{i+1} = [x_{i+1} \quad y_{i+1}]^T$ is related to its input vector $\boldsymbol{v}_i = [x_i \quad y_i]^T$ as

$$\boldsymbol{v}_{i+1} = \sec(\alpha_i)\begin{bmatrix} \cos(\delta_i\alpha_i) & \sin(\delta_i\alpha_i) \\ -\sin(\delta_i\alpha_i) & \cos(\delta_i\alpha_i) \end{bmatrix}\boldsymbol{v}_i \tag{4}$$

which is essentially a clockwise rotation of the vector $\boldsymbol{v}_i$ through an angle $\delta_i\alpha_i$, followed by a scaling with $\sec(\alpha_i)$. In addition to this observation that rotation through certain angles, α_i, can be performed using only shifts and adds, CORDIC also provides a simple mechanism to approximate any angle,[1] θ as a summation of angles α_i.

$$\theta \approx \alpha = \sum_{i=0}^{n-1} \delta_i \alpha_i \quad \text{where } \delta_i \in \{-1, 1\}. \tag{5}$$

A third variable z and the corresponding CORDIC iteration given by (3) are used to compute this approximation conveniently. Several CORDIC iterations with the appropriate directions of rotation δ_i can compute the rotation of an initial vector $\boldsymbol{v}_0$ through any desired angle θ, but for a scale factor $K_n = \prod_{i=0}^{n-1}\sec(\alpha_i)$. After n CORDIC iterations a special correction step is performed to eliminate this scale factor. Many techniques have been developed to speed up this scale factor correction [1], [4], [7], [8], [12].

The choice of δ_i at each iteration determines the functions that can be computed in the circular mode of CORDIC. The *Z-reduction* or the *rotation* operation of CORDIC may be performed by a choice of δ_i given by

$$\delta_i = \begin{cases} 1 & \text{if } z_i \le \theta \\ -1 & \text{if } z_i > \theta. \end{cases} \tag{6}$$

[1] In this paper, this angle is restricted to lie between $-\pi/2$ and $\pi/2$.

The final rotated vector $\boldsymbol{v}_n$ is obtained in the x and y variables as shown in Table I. The inverse tangent computation, which requires CORDIC operation in the *Y-reduction* or *vectoring* mode, can be selected by a choice of δ_i at each iteration given by

$$\delta_i = \begin{cases} 1 & \text{if } y_i x_i \geq 0 \\ -1 & \text{if } y_i x_i < 0. \end{cases} \tag{7}$$

The above choice [24] corresponds to the choice of δ_i, which tries to reduce the magnitude of y_i to zero. The inverse tangent is accumulated in the z variable.

III. Error Analysis of CORDIC

In some of the earliest work, Walther devoted a section in his paper to the accuracy of CORDIC and the implications on processor design. He analyzed the x and y iterations independently of the z iterations and concluded that $\log n$ extra bits in the data paths can provide sufficient accuracy. Hu [16] has recently obtained more precise bounds on the various errors in all modes of CORDIC for both floating-point and fixed-point representations. However, his study did not include the accuracy problem with inverse tangent calculations.

In the study of numerical error in CORDIC, it is convenient to split the errors due to different factors as an *approximation* error and a *truncation* error [16]. The approximation error is a result of the finite number of iterations in any practical implementation of CORDIC. The truncation error is caused by the finite word length of any data path.

Let $Q[\cdot]$ denote the *quantization* operator. The quantization operator on a scalar x_i is defined as

$$Q[x_i] = x_i - e_i$$

where $e_i < 2^{-b}$ is the truncation error, for a fixed-point data path with b fractional bits. Similarly, the quantization of a vector, $\boldsymbol{v_i} = [x_i \quad y_i]^T$, is defined as

$$\boldsymbol{Q}[\boldsymbol{v_i}] = \begin{bmatrix} Q[x_i] \\ Q[y_i] \end{bmatrix}. \tag{8}$$

For example, consider a fixed-point representation of 5 bits, with the lower order 3 bits after the binary point. If $x_i = 1.2345678$, then the approximate representation of this number is 01001. Hence $Q[x_i] = 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 + 0.125 = 1.125$. Hence the quantization error $e_i = 1.2345678 - 1.125 = 0.1095678 < 2^{-3} = 0.125$.

The following notation is used in the derivations.

- $\hat{x}_i, \hat{y}_i, \hat{z}_i, \hat{\boldsymbol{v}}_i, \varphi$ represent the values obtained in a real-world implementation of CORDIC. These numbers include the effects of both approximation error and the truncation error.
- $x_i, y_i, z_i, \boldsymbol{v}_i, \alpha, \alpha_i$ represent the values that will be obtained in a hypothetical CORDIC unit with infinite precision, where every iteration uses the same δ_i obtained in a real implementation. Hence, the same sequence of rotations as those performed in a real CORDIC unit, *viz.* iterations given by (1)–(3) are performed on numbers represented with an infinite number of bits.
- $\tilde{x}_i, \tilde{y}_i, \tilde{z}_i, \tilde{\boldsymbol{v}}_i, \theta$ represent the values that should be obtained for the desired function with infinite precision, as mathematically defined.

The following lemma, relating $\boldsymbol{v}_n$ and $\hat{\boldsymbol{v}}_n$ can be derived from the x and y iterations given by (1) and (2).

Lemma 3.1: Let $\hat{\boldsymbol{v}}_n = [\hat{x}_n \quad \hat{y}_n]^T$ be the vector obtained in a practical implementation of the CORDIC unit and $\boldsymbol{v}_n = [x_n \quad y_n]^T$ be the vector obtained in a hypothetical CORDIC unit with infinite precision. If ϵ is the precision of the fixed-point representation of $\hat{x}_i$ and $\hat{y}_i$, defined as $\epsilon = 2^{-b}$, where b is the number of fractional bits, then

$$\begin{aligned} \|\hat{\boldsymbol{v}}_n - \boldsymbol{v}_n\| &< \sqrt{2}\epsilon \left(1 + \sum_{j=1}^{n-1} \left[\prod_{l=j}^{n-1} \sqrt{1 + 2^{-2l}} \right] \right) \\ &< 1.5\epsilon n \qquad \text{for} \quad n > 1. \end{aligned} \tag{9}$$

Proof: This is a result derived by Hu [16]. A plot of $(1 + \sum_{j=1}^{n-1} [\prod_{l=j}^{n-1} \sec(\alpha_l)])$ versus n shows that it is very close to n and is less than $1.06n$ for all practical values of n. Thus the result reduces to

$$\|\hat{\boldsymbol{v}}_n - \boldsymbol{v}_n\| < 1.06\sqrt{2}\epsilon n < 1.5\epsilon n. \tag{10}$$

□

A. Bounds for Z-Reduction

The goal of Z-reduction iterations is to obtain the result of the rotation of a vector, stored in x_0 and y_0, through an angle θ initially stored in z_0. The result is obtained in $\hat{x}_n$ and $\hat{y}_n$ except for a scale factor, which is eliminated through further scale factor correction iterations. Implementation of (6) requires a comparison of z_i with θ at each iteration. This can be avoided by substituting the variable z_i with $(\theta - z_i)$. In other words, the variable z_i now stores the remainder of the angle to rotate through. Equations (3) and (6) may now be rewritten to reflect this substitution.

$$z_{i+1} = z_i - \delta_i \alpha_i, \tag{11}$$

$$\delta_i = \begin{cases} 1 & \text{if } z_i \geq 0 \\ -1 & \text{if } z_i < 0. \end{cases} \tag{12}$$

The initialization for z-reduction in Table I corresponds to this modified z-iteration. It is important to note that the vector $\boldsymbol{v}_n$ is the result of an *exact* rotation of the initial vector through an angle $\alpha = \sum_{i=0}^{n-1} \delta_i \alpha_i = \sum_{i=0}^{n-1} \delta_i \tan^{-1}(2^{-i})$, except for the scale factor. The following lemmas can be obtained from the z iterations for Z-reduction to relate this angle α to the intended rotation angle θ. Lemma 3.2 quantifies the approximation error in z iterations.

Lemma 3.2:

$$\left| \theta - \sum_{i=0}^{n-1} \delta_i Q[\alpha_i] \right| = \hat{z}_n < Q[\alpha_{n-1}] \tag{13}$$

Proof: This is a result of the convergence theorem of CORDIC. The interested reader is referred to Walther [23]. This lemma states that the approximation of the angle θ will be accurate to within $Q[\alpha_{n-1}]$, which is the smallest angle in the sequence of angles $Q[\alpha_i]$. Mathematically, CORDIC converges for $|\theta| \leq \sum_{i=0}^{n-1} \alpha_i$. However, in a practical implementation, this range of convergence reduces to $|\theta| \leq \sum_{i=0}^{n-1} Q[\alpha_i]$. □

The truncation error in the z iterations is quantified by the following lemma.

Lemma 3.3:

$$\alpha - \sum_{i=0}^{n-1} \delta_i Q[\alpha_i] = \sum_{i=0}^{n-1} \delta_i \mu_i \quad \text{where} \quad \mu_i = \alpha_i - Q[\alpha_i] \tag{14}$$

$$\left|\alpha - \sum_{i=0}^{n-1} \delta_i Q[\alpha_i]\right| < n\mu \tag{15}$$

Proof: The errors μ_i are the result of quantization of the angles α_i. If a fixed-point representation is used, all the errors μ_i are bounded by μ, the precision of the z-data path. Hence, the result follows from the definitions of α and μ_i. □

The following lemma relates the desired result $\tilde{\boldsymbol{v}}_n$ and $\boldsymbol{v}_n$.

Lemma 3.4:

$$\begin{aligned} ||\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n|| &= 2K_n \left|\sin\left(\frac{\theta-\alpha}{2}\right)\right| \cdot ||\boldsymbol{v}_0|| \\ &\cdot < K_n|\theta - \alpha|\ ||\boldsymbol{v}_0|| \end{aligned} \tag{16}$$

Proof: The desired result $\tilde{\boldsymbol{v}}_n$ is an exact rotation of the initial vector $\boldsymbol{v}_0$ through an angle θ except for the scale factor K_n. Similarly, $\boldsymbol{v}_n$ is an exact rotation of the same vector $\boldsymbol{v}_0$ through an angle α, but for the scale factor K_n.

$$\begin{aligned} \tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n &= K_n \begin{bmatrix} \cos\theta - \cos\alpha & \sin\theta - \sin\alpha \\ -(\sin\theta - \sin\alpha) & \cos\theta - \cos\alpha \end{bmatrix} \boldsymbol{v}_0 \\ &= 2K_n \sin\left(\frac{\theta-\alpha}{2}\right) \\ &\cdot \begin{bmatrix} -\sin\left(\frac{\theta+\alpha}{2}\right) & \cos\left(\frac{\theta+\alpha}{2}\right) \\ -\cos\left(\frac{\theta+\alpha}{2}\right) & -\sin\left(\frac{\theta+\alpha}{2}\right) \end{bmatrix} \boldsymbol{v}_0. \end{aligned}$$

Since orthogonal transformations do not affect the 2-norm,

$$||\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n|| = 2K_n \left|\sin\left(\frac{\theta-\alpha}{2}\right)\right| ||\boldsymbol{v}_0||.$$

The following relation holds for any θ and α:

$$\left|\sin\left(\frac{\theta-\alpha}{2}\right)\right| < \frac{1}{2}|\theta - \alpha|.$$

Substituting this inequality in the previous relation gives the desired result.

$$||\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n|| < K_n|\theta - \alpha|\ ||\boldsymbol{v}_0||$$

□

Combining the results of the above lemmas, a bound on the error, $\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n$, can be obtained as follows:

$$\begin{aligned} \tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n &= (\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n) + (\boldsymbol{v}_n - \hat{\boldsymbol{v}}_n) \\ ||\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n|| &< ||\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n|| + ||\boldsymbol{v}_n - \hat{\boldsymbol{v}}_n|| \\ &< K_n|\theta - \alpha|\ ||\boldsymbol{v}_0|| \\ &\quad + \sqrt{2}\epsilon\left(1 + \sum_{j=1}^{n-1}\left[\prod_{l=j}^{n-1} \sec\alpha_l\right]\right) \\ &< K_n||\boldsymbol{v}_0||\left(\left|\theta - \sum_{i=0}^{n-1}\delta_i Q[\alpha_i]\right| \right. \\ &\quad \left. + \left|\alpha - \sum_{i=0}^{n-1}\delta_i Q[\alpha_i]\right|\right) + 1.5\epsilon n \end{aligned}$$

$$||\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n|| < K_n(Q([\alpha_{n-1}] + n\mu)||\boldsymbol{v}_0|| + 1.5\epsilon n. \tag{17}$$

The above relation has important implications regarding the design of a CORDIC processor. The error consists of a component due to the x and y iterations and a component due to the z iterations. Since the errors are bounded, this offers a simple scheme to produce results that are accurate to an arbitrary number of bits.

Consider a CORDIC unit with all data paths that are m bits wide. Since at any iteration the condition $-(\pi/2) \leq \hat{z}_i \leq (\pi/2)$ holds, which is a total range of 3.14, only two bits are necessary before the binary point. Hence the precision of the z data path is $\mu = 2^{-(m-2)}$. The maximum value of the RHS in the bound occurs when the 2-norm of the initial vector is such that $K_n||\boldsymbol{v}_0|| \approx 2^{(m-1)}\epsilon$. Thus the error is

$$\begin{aligned} ||\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n|| &< (Q[\alpha_{n-1}] + n\mu)K_n||\boldsymbol{v}_0|| + 1.5\epsilon n \\ &< (2^{-(n-1)} + n2^{-(m-2)})2^{m-1}\epsilon + 1.5\epsilon n \\ &< 2^{m-n}\epsilon + 3.5\epsilon n. \end{aligned}$$

Hence to obtain a precision of n bits, a CORDIC unit with $(n + \log n + 2)$ bits and n iterations would be sufficient.

B. Bounds for Y-Reduction

The goal of Y-reduction iterations is to compute the inverse tangent $\theta = \tan^{-1}(y_0/x_0)$. Initially, z_0 is set to zero and the inverse tangent in accumulated in $\hat{z}_n$. This is achieved by performing iterations given by (1)–(3) with the choice of δ_i at each iteration given by (7). The convergence property of CORDIC guarantees that $\hat{y}_n$ is reduced to a quantity close to zero, in the order of a few ϵ, the precision of the xy-datapath. Considering the hypothetical CORDIC unit with infinite precision, the following relation holds:

$$y_n = (y_0 \cos\alpha - x_0 \sin\alpha)K_n. \tag{18}$$

Let r and θ be defined as

$$r = \sqrt{x_0^2 + y_0^2}$$

$$\tan\theta = \frac{y_0}{x_0}.$$

Equation (18) can now be rewritten as

$$\frac{y_n}{K_n} = r\sin\theta\cos\alpha - r\sin\alpha\cos\theta$$

$$\frac{y_n}{K_n\sqrt{x_0^2+y_0^2}} = \sin(\theta-\alpha)$$

$$|\theta-\alpha| = \left|\sin^{-1}\left(\frac{y_n}{K_n\sqrt{x_0^2+y_0^2}}\right)\right|. \quad (19)$$

By definition, the 2-norm of a vector $\boldsymbol{v}_n = [x_n \quad y_n]^T$ is $||\boldsymbol{v}_n|| = \sqrt{x_n^2+y_n^2}$. Hence, from Lemma 3.1

$$||\boldsymbol{v}_n - \hat{\boldsymbol{v}}_n|| < 1.5\epsilon n$$

$$\sqrt{(x_n-\hat{x}_n)^2+(y_n-\hat{y}_n)^2} < 1.5\epsilon n$$

$$|y_n-\hat{y}_n| < 1.5\epsilon n$$

$$|y_n| < 1.5\epsilon n + |\hat{y}_n|.$$

If n is large, the approximation error $\hat{y}_n$, which is of the order of a few ϵ, can be neglected. The result for the inverse tangent, obtained from a real CORDIC unit, is the angle accumulated in the z register $\hat{z}_n = \varphi$. Since y-reduction iterations are performed with $\hat{z}_0 = 0$, the final angle $\hat{z}_n$ can be expressed as a summation

$$\hat{z}_n = \sum_{i=0}^{n-1} \delta_i Q[\alpha_i].$$

Hence, using Lemma 3.3 we can conclude that $|\varphi-\alpha|$ is bounded by $n\mu$. Combining these relations

$$|\theta-\varphi| < |\theta-\alpha| + |\alpha-\varphi|$$

$$< \left|\sin^{-1}\left(\frac{y_n}{K_n\sqrt{x_0^2+y_0^2}}\right)\right| + n\mu$$

$$|\theta-\varphi| < \left|\sin^{-1}\left(\frac{y_n}{K_n\sqrt{x_0^2+y_0^2}}\right)\right| + n\mu \quad (20)$$

where

$$y_n < 1.5\epsilon n.$$

Here, θ is the true inverse tangent that is to be evaluated. The final value accumulated in the z variable, $\hat{z}_n = \varphi$, is the angle actually obtained. Thus (20) gives the numerical error in inverse tangent calculations. However, this relation does not provide a constant bound for all x_0 and y_0. In the worst case, x_0 and y_0 can be close to zero, resulting in a large error. Fig. 1 shows the errors observed in the computation of inverse tangent for various initial values. An intuitive explanation of this error is the successively larger right shift in the CORDIC iterations given by (1) and (2), which can result in a loss of all significant bits if x_0 and y_0 are not large enough. The same problem is not observed in a floating-point implementation of CORDIC since in such an implementation the relative error, rather than the absolute error, is bounded at each iteration. Intuitively, the errors in such an implementation have to be less since the initial values at any iteration are, by definition, always in a normalized form.

The result we obtained differs from that obtained by Duryea and Pottle [10]. Duryea and Pottle used an infinite precision CORDIC unit as an intermediate stage to couple the errors in xy iterations with those that occur in z iterations. The intermediate stage used in their analysis differs from ours in that the sequence of directions of rotation δ_i are different. We use the same sequence of δ_i as that obtained in a real system, while Duryea and Pottle used a sequence that would be generated in an infinite precison CORDIC unit. Since the sequence of rotations δ_i is not the same in the real system and in an infinite precision system, the bound on the error due to z iterations is affected. This leads to the conclusion that the error in inverse tangent computations in a fixed-point CORDIC unit is always less than $\log((n+5)/2)$ bits.

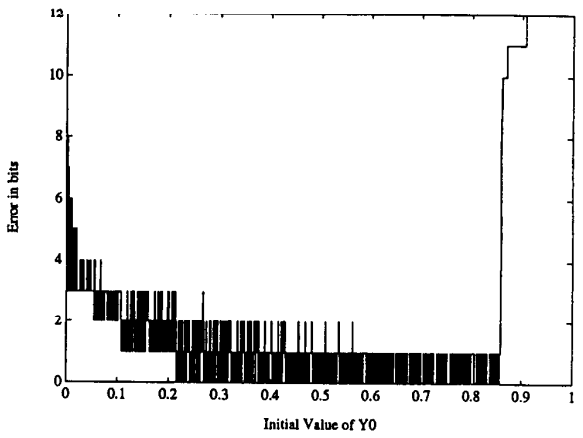

Fig. 1. This graph illustrates the numerical problems that occur in inverse tangent calculations using a fixed-point CORDIC unit without normalization. The graph plots the value of $\tan^{-1}(1) = \tan^{-1}(y_0/x_0)$ where $x_0 = y_0$, over the entire range of y_0. The error has been quantized into the number of bits in error. A large error is observed for small values of y_0. For very large y_0, the large error is due to overflow. In the SVD problem, by choosing small values for the initial matrix, the values at all subsequent iterations can be kept sufficiently small as to avoid overflow. This experiment was performed on a data path that is 16 bits wide.

IV. Normalization for CORDIC Y-Reduction

This section presents one way to control the errors that occur in CORDIC Y-reduction. Equation (20) indicates that enforcing a lower bound on $||\boldsymbol{v}_0||$ can help bound the errors. Such an approach is feasible only because of two properties that CORDIC iterations satisfy. The first property is that a scaled input vector results in an output that can be easily corrected for the scaling. Scaling of input does not affect the inverse tangent calculation, while it affects the resultant vector $\boldsymbol{v}_n$ only as scaling. It is convenient to limit the scaling to powers of two, since scaling is used only to artificially enforce a lower bound on the norm of the input vector and the exact amount of scaling is never a matter of concern. The second property of CORDIC that allows normalization of input is the bound on the norm of the final rotated vector. Since vector rotation is an orthogonal transformation, we have the relation $||\boldsymbol{v}_n||/||\boldsymbol{v}_0|| = K_n$, which implies that the norm of the rotated vector is within a constant factor of the norm of the input vector. The constant K_n is always between 1.414 and 1.646 for all word sizes. This guarantees that providing two additional bits at the higher end is sufficient to avoid any overflows.

The above can be quantified as follows.

Theorem 4.1: Given an initial vector $[x'_0 \quad y'_0]^T$, the inverse tangent, $\tan^{-1}(y'_0/x'_0)$ can be computed with bounded errors using CORDIC, if the initial vector $\hat{\boldsymbol{v}}_0$ is set to $[x'_0 2^j \quad y'_0 2^j]$, where

$$j = \left\lfloor \log_2 \left[\min \left\{ \frac{2^{m-4}\epsilon}{|x'_0|}, \frac{2^{m-4}\epsilon}{|y'_0|} \right\} \right] \right\rfloor \tag{21}$$

and ϵ and m are the precision and the width, respectively, of the x and y registers.

Proof: The quantity $\log_2(2^{m-4}\epsilon/|x'_0|)$ represents the number of leading zeros in x'_0 after allowing one sign bit and two bits for overflow. Hence j is the smaller of the number of leading zeros in x'_0 and y'_0. This choice of j ensures that

$$2^{m-4}\epsilon < \max\{x'_0 2^j, y'_0 2^j\} < 2^{m-3}\epsilon$$
$$2^{m-4}\epsilon < ||\hat{\boldsymbol{v}}_0|| < 2^{m-3}\sqrt{2}\epsilon$$
$$||\hat{\boldsymbol{v}}_n|| < K_n 2^{m-3}\sqrt{2}\epsilon < 2^{m-1}\epsilon.$$

Hence the choice of j ensures a lower bound on the value of $||\hat{\boldsymbol{v}}_0||$ and prevents any overflow as shown by the bound on $||\hat{\boldsymbol{v}}_n||$. Substituting the bound on $||\hat{\boldsymbol{v}}_0||$ in (20), we get

$$|\theta - \varphi| < \left| \sin^{-1}\left(\frac{1.5\epsilon n}{K_n 2^{m-4}\epsilon} \right) \right| + n\mu \approx \frac{n}{2^{m-4}} + n\mu.$$

□

The above theorem demonstrates that normalization results in bounded numerical errors in the computation of the inverse tangent. If all data paths are m bits wide, then the precision of z-data path μ equals $2^{-(m-2)}$. Hence, the above error translates to less than $5n\mu$.

A. An $O(n^2)$ Area- $O(1)$ Time Solution

Implementation of normalization requires hardware to determine the amount by which components of the vector can be shifted left, and hardware to implement the shifts. The simplest implementation may use two leading-zero encoders to determine the amounts through which each component, x_0 and y_0, can be shifted, a comparator to choose the smaller of the two shifts, and barrel shifters to implement the shifts in a single cycle. Note that the barrel shifters to implement (1) and (2) cannot be reused here, since the normalization shifts are left shifts, while the CORDIC iterations require right shifts. This generic solution to the normalization incurs an $O(1)$ penalty on the number of clock cycles required and an $O(n^2)$ penalty on the gate count. The duration of an individual clock cycle may be affected depending on where this shifter is placed in the CORDIC architecture.

B. An $O(n \log n)$ Area- $O(\log n)$ Time Solution

A tradeoff between time and area can be achieved by implementing a variable shifter with only selected shifts of powers of two, and implementing the desired normalization shift in log n cycles. Such an approach results in a shifter with an area complexity $O(n^2)$, which may, however, be smaller than a full barrel shifter for practical values of n. The complexity of leading-zero encoders is still $O(n^2)$. The implementation is considerably more complex and incurs a penalty of $O(\log n)$ on the number of clock cycles required. The duration of an individual clock cycle may incur a smaller penalty than the previous implementation.

C. A New $O(n^{1.5})$ Area Solution with No Time Penalty

In this section we present a new normalization technique that incurs an area penalty of $O(n^{1.5})$ but no penalty on the number of clock cycles. It will, however, incur a small penalty on the length of the clock cycle. This technique, we call *partial normalization,* achieves the desired normalization shift as a combination of smaller shifts merged with the CORDIC iterations. At the end of each iteration, the variables x and y are shifted left by a few bits, decided by control logic, until the desired total shift is achieved. The design goal is to minimize the maximum shift that needs to be introduced at each iteration. The normalization shift introduced at every iteration prevents CORDIC from shifting out and losing significant bits at subsequent iterations. The above scheme results in two competing processes. The partial normalization scheme can only implement a total shift that grows linearly as the number of iterations. CORDIC, however, tends to shift out bits as a square of the number of iterations due to the successively increasing shift i at each iteration.

Before presenting the details of the algorithm, it is convenient to define *Type 1* and *Type 2* CORDIC iterations.

Definition 4.1 (Type 1 CORDIC Iterations): Given a vector $\hat{\boldsymbol{v}}_i = [\hat{x}_i \quad \hat{y}_i]^T$ and the direction of rotation δ_i, the result vector $\hat{\boldsymbol{v}}_{i+1} = [\hat{x}_{i+1} \quad \hat{y}_{i+1}]^T$ is defined as

$$\hat{x}_{i+1} = Q[\hat{x}_i + \delta_i \hat{y}_i 2^{-i}]$$
$$\hat{y}_{i+1} = Q[\hat{y}_i - \delta_i \hat{x}_i 2^{-i}].$$

The *Type 1* CORDIC iterations are the same equations defined by (1) and (2) as implemented in practice. The modified CORDIC iterations called *Type 2* iterations are defined as follows.

Definition 4.2 (Type 2 CORDIC Iterations): Given a vector $\hat{\boldsymbol{v}}_i = [\hat{x}_i \quad \hat{y}_i]^T$, the direction of rotation δ_i, and a constant j, the result vector $\hat{\boldsymbol{v}}_{i+1} = [\hat{x}_{i+1} \quad \hat{y}_{i+1}]^T$ is defined as

$$\hat{x}_{i+1} = Q[\hat{x}_i + \delta_i \hat{y}_i 2^{-i}]2^j$$
$$\hat{y}_{i+1} = Q[\hat{y}_i - \delta_i \hat{x}_i 2^{-i}]2^j.$$

The result of a *Type 2* iteration $\hat{\boldsymbol{v}}_{i+1}$ is the result of a *Type 1* iteration followed by a left shift of j.

The normalization algorithm is based on certain properties that ensure that quantization has no effect on the iterations. The following lemmas describe these properties. The quantity ϵ which appears in the following derivations is the precision of the x and y datapaths as defined earlier.

Lemma 4.1: If a *Type 1* or a *Type 2* CORDIC iteration is performed on a vector $\hat{\boldsymbol{v}}_i = [\hat{x}_i \quad \hat{y}_i]^T$, a sufficient condition to ensure that the quantization operator has no effect on the output of the iteration, is that the components $\{\hat{x}_i, \hat{y}_i\}$ are multiples of $2^p\epsilon$, where p is an arbitrary integer greater than i.

Proof: Consider the computation $Q[\hat{x}_i + \delta_i \hat{y}_i 2^{-i}]$. In order to ensure that the quantization operator has no effect on the result, the quantity $\hat{x}_i + \delta_i \hat{y}_i 2^{-i}$ should be representable in m bits, where m is the word length of $\hat{x}_i$ and $\hat{y}_i$. Quantization

will result in loss of precision only if the right shift of y_i through i bits results in lost significant bits. If $\hat{y}_i$ is a multiple of $2^p\epsilon$, where $p \geq i$, the lower order bits shifted out are all zeros, leading to the correct result even in the presence of truncation. An identical argument holds for the computation of $\hat{y}_{i+1}$. □

Lemma 4.2: If n CORDIC iterations of *Type 1* are performed on a vector $\hat{\boldsymbol{v}}_0 = [\hat{x}_0 \quad \hat{y}_0]^T$, where the initial vector components are multiples of $2^r\epsilon$, then quantization has no effect on all iterations with index i that satisfy the following condition:

$$i(i+1) \leq 2r. \tag{22}$$

Proof: Suppose the initial vector $\hat{\boldsymbol{v}}_0$ has components that are multiples of $2^r\epsilon$. After the iteration with $i = 0$, the components of the vector $\hat{\boldsymbol{v}}_1 = [\hat{x}_1 \quad \hat{y}_1]^T$ are both multiples of $2^r\epsilon$. If r is greater than 1, then from Lemma 4.1, quantization has no effect on that iteration and the results are multiples of $2^{r-1}\epsilon$. If $r-1$ is greater than 2, then the iteration $i = 2$ is not affected by quantization. The result of this iteration, $\hat{\boldsymbol{v}}_3 = [\hat{x}_3 \quad \hat{y}_3]^T$, has components that are multiples of $2^{r-1-2}\epsilon$. The same argument holds at an iteration i; the components of $\hat{\boldsymbol{v}}_i = [\hat{x}_i \quad \hat{y}_i]^T$, are multiples of $2^{r-[1+2+\cdots+(i-1)]}\epsilon$ and the following condition ensures that quantization has no effect on the result:

$$\begin{aligned} r - [1+2+\cdots+(i-1)] &\geq i \\ r - \frac{(i-1)i}{2} &\geq i \\ i(i+1) &\leq 2r. \end{aligned}$$

□

Lemma 4.3: If n CORDIC iterations of *Type 2* with iteration index $i, 0 \leq i < n$, and a constant left shift j, are performed on an arbitrary initial vector, then the quantization operator has no effect on the iterations with iteration index i, which satisfy the following relation:

$$0 \leq i \leq 2j-1. \tag{23}$$

Proof: The quantization has no effect on the first iteration with $i = 0$, since there is no right shift of any of the operands. The results of the first iteration, $\{\hat{x}_1, \hat{y}_1\}$, are multiples of $2^j\epsilon$. If j is greater than i, then $Q[\cdot]$ has no effect on the iteration with $i = 1$. The result of this iteration is the vector $\hat{\boldsymbol{v}}_2 = [\hat{x}_2 \quad \hat{y}_2]^T$, with each component a multiple of $2^{j-1+j}\epsilon = 2^{2j-1}\epsilon$. If $2j-1 \geq 2$, the third iteration is also unaffected by the quantization operator. The resulting $\{\hat{x}_3, \hat{y}_3\}$ are multiples of $2^{(2j-1)-2+j}\epsilon = 2^{3j-(1+2)}\epsilon$. This continues until the components $\{\hat{x}_i, \hat{y}_i\}$ are multiples of $2^{ij-[1+2+\cdots+(i-1)]}\epsilon$. The quantization operator has no effect on the iterations as long as

$$\begin{aligned} ij - [1+2+\cdots+(i-1)] &\geq i \\ ij - \frac{(i-1)i}{2} &\geq i \\ i &\leq 2j-1. \end{aligned}$$

□

Lemma 4.2 gives the number of iterations of *Type 1* that can be performed without any quantization effects, if the initial vector values are prenormalized by a factor $2^r\epsilon$. Lemma 4.3 gives the number of iterations of *Type 2* that can be performed without any quantization effects if a left shift of j is introduced at each step. The following theorem describes the conditions to be satisfied, if the same effect as a prenormalization is to be achieved using iterations of *Type 2*.

Theorem 4.2: Suppose the vector $\hat{\boldsymbol{v}}'_n$ is the result of n iterations of *Type 1* performed on inputs that are prenormalized to be multiples of $2^r\epsilon$, where $r \bmod j = 0$. Suppose the result vector $\hat{\boldsymbol{v}}''_n$ is obtained by performing r/j iterations of *Type 2* with a left shift of j on unnormalized inputs, followed by $n - r/j$ iterations of *Type 1*. The results $\hat{\boldsymbol{v}}'_n$ and $\hat{\boldsymbol{v}}''_n$ will be identical if the condition, $r < 2j^2$, is satisfied.

Proof: If the vectors $\hat{\boldsymbol{v}}'_n$ and $\hat{\boldsymbol{v}}''_n$ could be computed without any quantization, then the two vectors would be identical. In the presence of quantization, if quantization had no effect on the first k/j iterations, then the result vectors will be identical since the rest of the iterations would be *Type 1* with the same starting values.

Using (23), if quantization has to have no effect on the first r/j iterations, the condition to be satisfied is

$$\begin{aligned} \frac{r}{j} - 1 &\leq 2j-1 \\ r &\leq 2j^2. \end{aligned}$$

Similarly, using (22), the condition required to ensure that quantization has no effect on the first r/j iterations is

$$\begin{aligned} \left(\frac{r}{j}-1\right)\left(\frac{r}{j}\right) &\leq 2r \\ \frac{r}{j} - 1 &\leq 2j \\ r &< 2j^2 + j. \end{aligned}$$

Thus, the condition $r < 2j^2$ ensures that quantization has no effect on the first r/j iterations in the computation of either $\hat{\boldsymbol{v}}'_n$ or $\hat{\boldsymbol{v}}''_n$. Hence the result. □

This theorem shows that if *Type 2* CORDIC iterations can be performed with a constant left shift j, then normalization through any multiple of j but less than $2j^2$ can be achieved. In practice, the parameter j can be made a variable, allowing one of several shifts to be chosen at each iteration. An appropriate choice of these shifts can achieve any desired total shift.

Suppose the shift j in *Type 2* iterations can be chosen from a total of λ shifts, and the condition shift $[\lambda-1]$ > shift $[\lambda-2]$ > $\cdots$ > 0 holds. The algorithm given in Fig. 2 gives a choice of shifts at each iteration to normalize the input. The parameters MAX_X and MAX_Y are the maximum allowed values of x_0 and y_0 chosen to prevent overflow. The required control can be implemented using combinational logic: a pair of leading-zero encoders and a comparator to select the smaller shift from the output of the encoders. The additional hardware required to implement partial normalization is a pair of shifters, which implement a subset of all the shifts implemented by a complete barrel shifter. Consequently, the leading-zero encoders need to encode very few bits to determine the appropriate shift.

```
begin
  for i := 0 to n do
    if (x_i < MAX_X and y_i < MAX_Y) then
      Choose largest xindex such that
        x_i 2^shift[xindex] < MAX_X ;
      Choose largest yindex such that
        y_i 2^shift[yindex] < MAX_Y ;
      index := min( xindex, yindex );
    else
        index := 0;
    end
    Normalization Shift j := shift[index]
    Perform modified CORDIC Iteration;
  end
end
```

Fig. 2. Modified CORDIC iterations, invoked during Y-reduction when the input is not normalized.

TABLE II
TOTAL SHIFTS THAT CAN BE ACHIEVED WITH A SMALL SHIFT IN EACH ITERATION, AS PART OF A NOVEL NORMALIZATION SHEME

Shift Introduced in Each CORDIC Iteration j	Maximum Shift that Can Be Achieved k
1	2
2	8
3	18
4	32
8	128

1) Choice of a Minimal Set of Shifts: The following is an example to illustrate the choice of the elements of the array *shift* for a CORDIC unit with data-width $m = 16$ bits and $n = 16$ iterations. The higher order three bits correspond to one sign and two overflow bits. The parameters MAX_X and MAX_Y are $2^{13}\epsilon$ for this example, where ϵ is the smallest value that can be represented. The array has to be chosen such that any normalization shift up to 13 that can occur in practice can be implemented without any loss of precision. The array elements can be chosen as follows.

- To permit unmodified CORDIC iterations in a modified CORDIC unit a shift zero is necessary $\Rightarrow shift[0] = 0$.
- An overall shift of 1 can be achieved only by choosing $j = 1$. Thus a shift of 1 is necessary in any implementation $\Rightarrow shift[1] = 1$.
- Multiple iterations with $j = 1$ can achieve any shift up to a maximum of $2j^2 = 2$. A shift of 3, however, cannot be implemented as three modified CORDIC iterations with $j = 1$, since this will result in a loss of significant bits. This necessitates the inclusion of the shift $j = 3$ to the set $\Rightarrow shift[2] = 3$.
- The shift $j = 3$ can achieve any shift up to a maximum $2j^2 = 18$ as given by Table II. Since the maximum shift required is 13, the normalization shifter does not have to implement any shift other than 0, 1, and 3. Any shift that is not a multiple of 3 is performed as shifts of 3 for $\lfloor k/3 \rfloor$ iterations, followed by iterations with $j = 1$ to achieve the remaining shift.

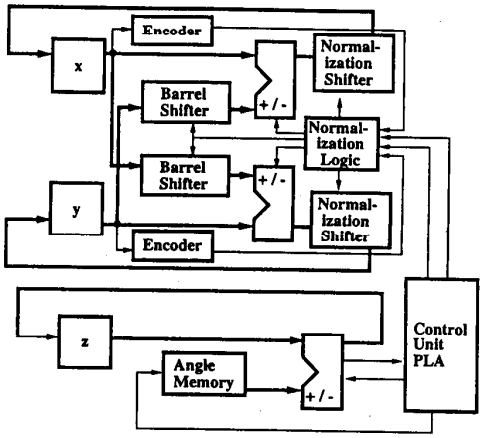

Fig. 3. Implementation of the modified CORDIC iterations in hardware.

Fig. 3 shows a hardware implementation of the CORDIC unit that includes the above normalization scheme. The number of bits in error using such a scheme is bounded even for small initial values, as shown in Fig. 4.

2) Cost of Partial Normalization Implementation: In a CORDIC implementation where the x and y datapaths are m bits wide, a shifter with a maximum shift $\sqrt{m/2}$ is required. The shifts at each iteration are obtained from leading-zero encoders that encode the $\sqrt{m/2}$ most significant bits of the x and y variables. The hardware costs involved in these operations are as follows.

Shifter: The area complexity of the shifter grows as

$$O(m \times \text{Maximum Shift required}) = O(m \times \sqrt{m}) = O(m^{1.5}).$$

In our design of a fixed-point CORDIC unit, the use of partial normalization reduced the additional area required for normalization from 14% of the die area to 4%. A full barrel shifter would have required m^2 gates, while the partial normalization scheme reduced this gate count to $m^{1.5}/\sqrt{2}$.

Control Logic: The size of the control logic, *viz.* the leading -zero encoders, grows as $O((\text{Maximum Shift required})^2) = O(\sqrt{m}^2) = O(m)$.

Time Penalty: The shifting is performed as part of the CORDIC iterations. Thus, the only time penalty is a decrease in the clock rate due to extra propagation delay caused by the presence of the shifter in the CORDIC data path. In our designs, the propagation delay through the normalization shifter was approximately $5ns$. The propagation delay through the leading-zero encoders was higher. However, the leading-zero encoders determine the normalization shift as a function of the inputs to an iteration, and the shift needs to be applied only on the outputs of the iteration. Hence they can work in parallel with the rest of the CORDIC circuitry. The clock cycle comprises of two parts. The first part is the larger of the propagation delays through the CORDIC circuitry and the leading-zero encoders. The second part is the propagation delay through the normalization shifter. In many cases, this second part can be minimized through careful design.

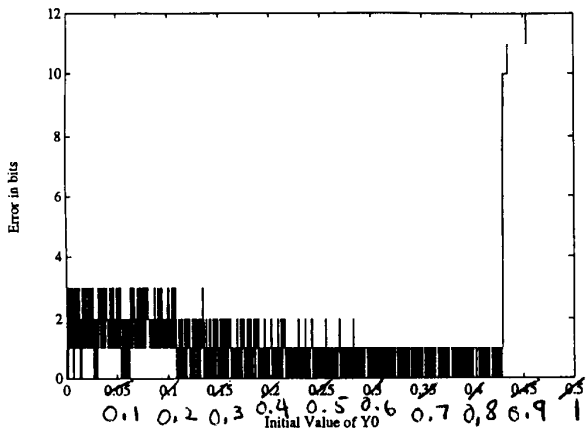

Fig. 4. Error in the computed value of $\tan^{-1}(1) = \tan^{-1}(y_0/x_0)$ with $x_0 = y_0$ over the entire range of y_0 in a CORDIC module that implements the partial normalization scheme.

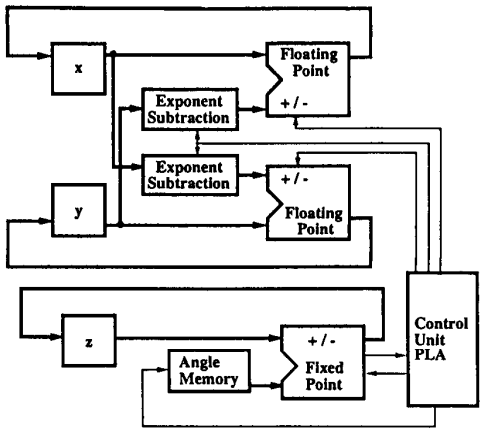

Fig. 5. Implementation of a hybrid CORDICprocessor.

The partial normalization scheme has been implemented in the CAPE processor[13]. As illustrated by Fig. 4 the numerical errors reduced considerably when the normalization unit was added to the CORDIC units.

V. A Hybrid Architecture for CORDIC

This section introduces the notion of a hybrid architecture for CORDIC that can be used to solve the numerical accuracy problems with inverse tangent calculations. As discussed before, the accuracy problem does not arise when the inputs are normalized. Since floating-point numbers are always normalized, this problem should not occur. However, a full floating-point implementation implies that the CORDIC module has to be capable of rotating through very small angles, which would be mapped to zero in a fixed-point implementation. By noting that such fine accuracy is not necessary at a higher level in some iterative algorithms, the complexity of the implementation can be reduced considerably. We first examine issues associated with floating-point implementations and then present the hybrid architecture.

A. Full Floating Point CORDIC

The conversion of CORDIC from fixed-point to floating-point formats is first described by Walther. The basic iteration equations are easily expressed using floating-point arithmetic. The shifting required in the fixed-point algorithm translates to exponent subtraction and mantissa alignment in the floating-point case.

Although at least n iterations are still performed, the first iteration is no longer rooted to the angle $\alpha_0 = \pi/4$ as in the fixed-point case. The first rotation angle α_0 may float to correspond to the most significant bit in the argument. By continuing for n iterations, the mantissa is accurate to n bits. However, a larger table of angles is necessary than in the fixed-point case. Also, a different scale factor is required for each family of iterations. For small angles, the approximations, $\sin z \approx z$ for $|z| < 2^{-n/2+1}$, and $\cos z \approx 1$ for $|z| < 2^{-n/2}$ can be used. Similarly for the inverse tangent, the approximation, $\tan^{-1}(y/x) = y/x$ for $|y/x| < 2^{-n/2}$, can be performed. These approximations reduce the size of the angle table. However, the recognition of these data dependent cases within the CORDIC processor can complicate the control and timing if used in special-purpose systems, such as systolic arrays.

B. Floating-point CORDIC with a Fixed-Point Angle Path

A general floating-point CORDIC processor must produce accurate results to full machine precision for arbitrary arguments. This requires the use of a larger angle memory and nonstandard scale factor correction constants. However, for matrix computations, such as the QRD, and for iterative algorithms, such as the SVD, such accuracy in the rotation angle calculation is not required. The parallel Jacobi algorithm for the SVD of a matrix A is required to produce an error less than $\epsilon||A||_F$ where ϵ is the machine precision. Therefore, it is not necessary to produce angles that are more accurate than ϵ.

In many numerical applications, rotations by such small angles do not further contribute to the overall accuracy of the algorithm. A scheme where the smallest rotation angle is 2^{-n} may possess sufficient accuracy even if all n bits of the normalized floating-point mantissa have not been explicitly determined. By limiting the table of angles to n words, the floating-point CORDIC algorithm reduces to an algorithm with complexity similar to the fixed-point case.

In Fig. 5, the design of a floating-point CORDIC processor for use in matrix computations is presented [5]. The x and y data paths use a standard floating-point format. The shifter that is required in the fixed-point CORDIC processor is logically replaced by an exponent subtraction unit. The actual exponent subtraction can occur within the floating-point adder. The important reduction in algorithm complexity occurs in the z data path where the angle information is accumulated. The angle memory, z register, and adder remain in a fixed-point format. In this hybrid processor structure, the z data path only requires the same number of bits as the mantissas of the x and y data paths. For example, if IEEE single precision is used, then the z data path would be composed of 24 bits and the angle memory would contain 24 words. In all designs,

additional bits would be needed to control numerical errors within the processor. Since a fixed number of iterations starting with the first angle, $i = 0, 1, \cdots, n-1$ is performed, a standard scale factor correction algorithm can be used.

VI. Numerical Errors in a Hybrid CORDIC Unit

In this section we extend the error analysis of Section III to the hybrid CORDIC architecture presented in Section V. We follow the convention used in the earlier analysis. However, it should be noted that $\hat{x}_n, \hat{y}_n$ are floating point quantities, with ϵ denoting the precision of the floating-point representation used. The quantities relating to the z data path are fixed-point with a precision μ.

The following lemma quantifies the truncation error in the x and y data paths, if $\hat{x}_i$ and $\hat{y}_i$ are floating-point quantities.

Lemma 6.1 Let $\hat{v}_n = [\hat{x}_n \quad \hat{y}_n]^T$ be the vector obtained in a practical implementation of the CORDIC unit and $v_n = [x_n \quad y_n]^T$ be the vector obtained in a hypothetical CORDIC unit with infinite precision. If ϵ is the precision of the floating-point representation of $\hat{x}_i$ and $\hat{y}_i$, defined as $\epsilon = 2^{-b}$, where b is the number of bits in the representation of the mantissa, then

$$\frac{||\hat{\boldsymbol{v}}_n - \boldsymbol{v}_n||}{||\boldsymbol{v}_n||} < n\epsilon. \tag{24}$$

Proof: This is a result derived by Hu [16]. As expected, the floating-point representation ensures that the relative error is bounded. □

The following lemma relates the 2-norms of the initial and the final vectors obtained in a hypothetical CORDIC unit with infinite precision.

Lemma 6.2:

$$||\boldsymbol{v}_n|| = K_n||\boldsymbol{v}_0|| \tag{25}$$

Proof: The $\boldsymbol{v}_n$ is an *exact* rotation of the initial vector $\boldsymbol{v}_0$ through an angle α, except for the scale factor.

$$\boldsymbol{v}_n = K_n \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \boldsymbol{v}_0$$

Since orthogonal transformations do not affect the 2-norm, the result follows. □

In addition, Lemmas 3.2 and 3.3 are valid for a hybrid architecture, since the z data path is fixed-point. In addition, Lemma 3.4 holds, since it does not depend on the number representation.

A. Bounds for Z-Reduction

Combining Lemma 3.4 with Lemma 6.1 the following result is obtained for the error in z-reduction.

$$\begin{aligned} \tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n &= (\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n) + (\boldsymbol{v}_n - \hat{\boldsymbol{v}}_n) \\ ||\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n|| &< ||\tilde{\boldsymbol{v}}_n - \boldsymbol{v}_n|| + ||\boldsymbol{v}_n - \hat{\boldsymbol{v}}_n|| \\ &< K_n|\theta - \alpha| \cdot ||\boldsymbol{v}_0|| + n\epsilon||\boldsymbol{v}_n|| \\ \frac{||\tilde{\boldsymbol{v}}_n - \hat{\boldsymbol{v}}_n||}{K_n||\tilde{\boldsymbol{v}}_0} &< (Q[\alpha_{n-1}] + n\mu) + n\epsilon \end{aligned} \tag{26}$$

Relation (26) shows that the relative error in the computed vector is bounded.

B. Bounds for Y-Reduction

The analysis of the Y-reduction mode of operation is similar to the analysis of the complete fixed-point CORDIC unit. Using Lemma 3.3 and (19) a bound on the error in the computed value of inverse tangent is obtained as follows:

$$\begin{aligned} |\theta - \varphi| &< |\theta - \varphi| + |\alpha - \varphi| \\ &< \left|\sin^{-1}\left(\frac{y_n}{K_n\sqrt{x_0^2 + y_0^2}}\right)\right| + n\mu \\ &< \left|\sin^{-1}\left(\frac{y_n}{||\boldsymbol{v}_n||}\right)\right| + n\mu. \end{aligned}$$

Since the iterations given by (1) and (2) are performed on floating-point numbers, the final value in the y register, $\hat{y}_n$, is not zero. Mathematically $\tilde{y}_n$ should be $||\tilde{\boldsymbol{v}}_n|| \sin\alpha_n \approx ||\tilde{\boldsymbol{v}}_n||\alpha_n$. The approximation is only valid if α_n is very small. We expect the actual value in the y register, $\hat{y}_n$, to be a quantity close to $\tilde{y}_n$.

From Lemma 6.1,

$$\begin{aligned} \frac{||\hat{\boldsymbol{v}}_n - \boldsymbol{v}_n||}{||\boldsymbol{v}_n||} &< n\epsilon \\ \frac{|\hat{y}_n - y_n|}{||\boldsymbol{v}_n||} &< n\epsilon \\ \frac{|y_n|}{||\boldsymbol{v}_n||} &< n\epsilon + \frac{|\hat{y}_n|}{||\boldsymbol{v}_n||}. \end{aligned}$$

The number of iterations n is typically chosen such that the angle α_n is close to the precision of the z data path. Hence, the quantity $|\hat{y}_n|/||\boldsymbol{v}_n||$ is in the order of μ. If the precision of the z datapath is of the same order as that of the x and y data paths, then this term is negligible. Thus the error in the computed inverse tangent is

$$|\theta - \varphi| < n\epsilon + \frac{|\hat{y}_n|}{||\boldsymbol{v}_n||} + n\mu. \tag{27}$$

Relation (27) shows that the hybrid architecture does not suffer from any numerical problems as the fixed-point version does.

VII. Conclusions

An analysis of CORDIC Y-reduction assuming a fixed-point representation of numbers shows that unnormalized input values can result in large numerical errors in the computed inverse tangent. We propose two solutions to the problem that try to achieve a low cost solution for implementation in VLSI. The first solution involves modifying the fixed-point CORDIC unit to include normalization units. This solution avoids the complexity of a floating-point implementation of the entire system by locally normalizing the values before using CORDIC. We have implemented the normalization using a method that integrates it with the CORDIC iterations resulting in an elegant solution to the problem. This method requires only $O(n^{1.5})$ extra hardware and in most cases has a minimal effect on latency. We believe this scheme can be extended to the other modes of CORDIC. We have implemented the

above scheme in the CAPE processor, which is to be used in an array to compute the SVD.

The notion of a hybrid architecture is attractive when the whole system is being built on top of a floating-point representation. If the CORDIC unit is not required to rotate through very small angles, the complexity of the CORDIC unit can be considerably reduced by implementing the z iterations in fixed-point and the others in floating-point. This implementation is attractive in an SVD array. The same idea may result in considerable savings in other applications.

ACKNOWLEDGMENT

The authors wish to thank the reviewers for their valuable comments that helped improve the contents and the presentation of this paper. The ideas in this paper were tested on a systolic array simulator on a Connection Machine CM-2, written by N. D. Hemkumar.

REFERENCES

[1] H. M. Ahmed, "Signal processing algorithms and architectures," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, June 1982.
[2] H. M. Ahmed, M. Morf, D. Lee, and P. Ang, "A VLSI speech analysis chip set based on square-root normalized ladder forms," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing,* 1981, pp. 648–653.
[3] R. P. Brent, F. T. Luk, and C. F. Van Loan, "Computation of the singular value decomposition using mesh-connected processors," *J. VLSI Comput. Syst.*, vol. 1, no. 3, pp. 242–270, 1985.
[4] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for an SVD processor," *J. Parallel Distributed Comput.*, vol. 5, no. 3, pp. 271–290, June 1988.
[5] ——, "Floating-point CORDIC for matrix computations," in *Proc. IEEE Int. Conf. Computer Design,* Oct. 1988, pp. 40–42.
[6] A. A. J. de Lange and E. F. Deprettere, "Design and implementation of a floating-point quasi-systolic general purpose CORDIC rotator for high-rate parallel data and signal processing," in *Proc. IEEE 10th Symp. Computer Arithmetic,* 1991, pp. 272–281.
[7] J. M. Delosme, "VLSI implementation of rotations in pseudo-Euclidean spaces," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing,* vol. 2, Apr. 1983, pp. 927–930.
[8] E. F. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC architectures for fast VLSI filtering and array processing," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing,* Mar. 1984, pp. 3:41A.6.1–41A.6.4.
[9] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.
[10] R. A. Duryea and C. Pottle, "Finite precision arithmetic units in Jacobi SVD architectures," School of Elec. Eng., Cornell Univ., Ithaca, NY, Tech. Rep. EE-CEG-87-11, Mar. 1987.
[11] M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: Application to matrix triangularization and SVD," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 725–740, June 1990.
[12] G. L. Haviland and A. A. Tuszynski, "A CORDIC arithmetic processor chip," *IEEE Trans. Comput.*, vol. C-29, no. 2, pp. 68–79, Feb. 1980.
[13] N. D. Hemkumar, K. Kota, and J. R. Cavallaro, "CAPE—VLSI implementation of a systolic processor array: Architecture, design and testing," in *Proc. 9th IEEE Biennial University/Government/Industry Microelectronics Symp.,* June 1991, pp. 64–69.
[14] S.-F. Hsiao and J.-M. Delosme, "The CORDIC Householder algorithm," in *Proc. IEEE 10th Symp. Computer Arithmetic,* 1991, pp. 256–263.
[15] X. Hu, R. G. Harber, and S. C. Bass, "Expanding the range of the CORDIC algorithm," *IEEE Trans. Comput.*, vol. 40, pp. 13–21, Jan. 1991.
[16] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *IEEE Trans. Signal Processing,* pp. 834–844, Apr. 1992.
[17] S. L. Johnsson and V. Krishnaswamy, "Floating-point CORDIC," Dep. Comput. Sci., Yale Univ., New Haven, CT, Tech. Rep. YALEU/DCS/RR-473, Apr. 1986.
[18] J. J. Modi and J. D. Pryce, "Efficient implementation of Jacobi's diagonalization method on the DAP," *Numer. Math.*, vol. 46, pp. 443–454, 1985.
[19] J. M. Muller, "Discrete basis and computation of elementary functions," *IEEE Trans. Comput.*, vol. C-34, no. 9, pp. 857–862, Sept. 1985.
[20] T. Y. Sung, Y. H. Hu, and H. J. Yu, "Doubly pipelined CORDIC array for digital signal processing algorithms," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing,* vol. 2, Apr. 1986, pp. 1169–1172.
[21] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," *IEEE Trans. Comput.*, vol. 40, no. 9, pp. 989–995, Sept. 1991.
[22] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sept. 1959.
[23] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. AFIPS Spring Joint Computer Conf.,* 1971, pp. 379–385.
[24] B. Yang and J. F. Böhme, "Reducing the computations of the SVD array given by Brent and Luk," *SIAM J. Matrix Anal. Appl.*, vol. 12, no. 4, pp. 713–725, Oct. 1991.

Kishore Kota (S'90) was born in Bapatla, India, on August 31, 1968. He received the B.Tech degree from Indian Institute of Technology, Madras, India, in 1989 and the M.S. degree from Rice University, Houston, TX, in 1991, both in electrical engineering.

He is currently pursuing the Ph.D. degree in electrical engineering at Rice University. His research interests are in the area of parallel computer hardware, currently with special emphasis on the practical aspects of VLSI in special-purpose processors, computer arithmetic, fault tolerance, and fault detection.

Mr. Kota is a member of Eta Kappa Nu.

Joseph R. Cavallaro (S'78–M'82) was born in Philadelphia, PA, on August 22, 1959. He received the B.S. degree from the University of Pennsylvania, Philadelphia, in 1981, the M.S. degree from Princeton University, Princeton, NJ, in 1982, and the Ph.D. degree from Cornell University, Ithaca, NY, in 1988, all in electrical engineering.

From 1981 to 1983, he was with AT&T Bell Laboratories, Holmdel, NJ. In 1988 he joined the faculty of Rice University, Houston, TX, where he is an Assistant Professor of Electrical and Computer Engineering. His research interests include computer arithmetic, fault tolerance, VLSI design and microlithography, and VLSI architectures and algorithms for parallel processing and robotics.

Dr. Cavallaro is a recipient of the NSF Research Initiation Award for 1989–1992 and the IBM Graduate Fellowship for 1987–1988, and is a member of Tau Beta Pi and Eta Kappa Nu.