# 1992 IEEE
# INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS

## Volume 1 of 6

Sheraton Hotel
San Diego, CA
May 10-13, 1992

92CH3139-3

# A Normalization Scheme to Reduce Numerical Errors in Inverse Tangent Computations on a Fixed-point CORDIC Processor

Kishore Kota  Joseph R. Cavallaro
Department of Electrical & Computer Engineering
Rice University, Houston, TX 77251

## Abstract

CO-ordinate Rotation DIgital Computer (CORDIC) is a unified arithmetic algorithm that allows efficient VLSI implementation of several elementary functions. Many special-purpose systems for real-time signal processing applications use a fixed-point representation of numbers due to the ease of implementation. 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 presents a method to integrate the normalization operation with CORDIC iterations for efficient implementation in $O(n^{1.5})$ hardware.

## 1 Introduction

The CO-ordinate Rotation DIgital Computer (CORDIC) is an arithmetic technique that allows efficient computation of a variety of elementary functions. The CORDIC algorithm is attractive from a hardware point of view since it uses only primitive operations such as shifts and additions to implement more complex functions including sine, cosine, tangent, arctangent, sinh, cosh, tanh, arctanh, ln and exp. A number of real-time signal processing and image processing algorithms have been developed to make efficient use of CORDIC in special-purpose parallel arrays. Interest in CORDIC has spurred recent work to improve the basic algorithm. Techniques have been developed to increase the range of convergence [4], speedup the computation by reducing the number of iterations [7] and pipeline the computation through conventional techniques or through the use of on-line arithmetic [3].

The basic CORDIC iterations as specified by Walther [9] are:

$$x_{i+1} = x_i + m\delta_i y_i 2^{-i} \quad (1)$$

$$y_{i+1} = y_i - m\delta_i x_i 2^{-i}, \quad (2)$$

$$z_{i+1} = z_i + \delta_i \tan^{-1} 2^{-i} \quad (3)$$

where $x_i$, $y_i$ and $z_i$ are the states of the variables $x$, $y$ and $z$ at the start of the $i^{th}$ iteration, $0 \leq i < n$, and $\delta_i \in \{-1, +1\}$. The inverse tangents, $\alpha_i = \tan^{-1} 2^{-i}$, are pre-computed and stored in a table of angles. CORDIC iterations can be performed in three different modes, *hyperbolic*, *linear* or *circular*, depending on the choice of $m \in \{-1, 0, 1\}$. All results in this paper have been obtained for the circular mode of CORDIC. The different functions implemented in this mode are summarized in Table 1.

The computation of interest in this paper is inverse tangent, which requires CORDIC operation in the *vectoring* or *Y-reduction* mode. The Y-reduction mode is selected by a choice of $\delta_i$ at each iteration given by:

$$\delta_i = \begin{cases} 1 & \text{if } y_i x_i \geq 0 \\ -1 & \text{if } y_i x_i < 0. \end{cases} \quad (4)$$

The above choice [10] corresponds to the choice of $\delta_i$, which tries to reduce the magnitude of $y_i$ to zero.

| Z-Reduction (for Vector Rotations, sine, cosine functions) | |
|---|---|
| Initial Values | Final values of variables |
| $x_0 = \tilde{x}_0$<br>$y_0 = \tilde{y}_0$<br>$z_0 = \theta$ | $x_n = (x_0 \cos\theta + y_0 \sin\theta)/K_n$<br>$y_n = (y_0 \cos\theta - x_0 \sin\theta)/K_n$<br>$z_n = 0$ |
| **Y-Reduction (for Inverse Tangent function)** | |
| $x_0 = \tilde{x}_0$<br>$y_0 = \tilde{y}_0$<br>$z_0 = 0$ | $x_n = \sqrt{x_0^2 + y_0^2}/K_n$<br>$y_n = 0$<br>$z_n = \theta - \tan^{-1}\left(\frac{y_0}{x_0}\right)$ |

Table 1: CORDIC Functionality in the Circular Mode

CORDIC has previously been implemented primarily using a fixed-point representation of the numbers. Design of a floating-point CORDIC unit requires handling of numerous special cases, which complicates design [2]. In particular, a floating-point implementation of a system that uses CORDIC requires normalization at every stage of computation, which is not necessary with a fixed-point representation.

## 2 Error Analysis of CORDIC Y-Reduction

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 [5]. The approximation error is a result of the finite number of iterations in an implementation of CORDIC. A truncation error is caused by the finite word length of the data paths. The following notation is used in the derivations.

- $\hat{x}_i$, $\hat{y}_i$, $\hat{z}_i$ represent the values obtained in a real-world implementation of CORDIC. These numbers include the effects of both approximation errors and truncation errors.
- $x_i$, $y_i$, $z_i$ represent the values that will be obtained in a hypothetical CORDIC unit with infinite precision, where every iteration uses the same sequence of $\delta_i$s used in the computation of $\hat{x}_i$, $\hat{y}_i$ and $\hat{z}_i$.
- $\tilde{x}_i$, $\tilde{y}_i$, $\tilde{z}_i$ represent the values that should be obtained for the desired function with infinite precision, as mathematically defined.

Let Q[.] denote the *quantization* operator. The result of the quantization operator on a vector is defined as the vector obtained through truncation of each vector element. The following lemma, relating the vector $\mathbf{v}_n = [x_n, y_n]^T$ and $\hat{\mathbf{v}}_n = [\hat{x}_n, \hat{y}_n]^T$ can be derived from the $x$ and $y$ iterations given by Equation 1 and Equation 2 respectively.



0-7803-0593-0/92 $3.00 1992 IEEE

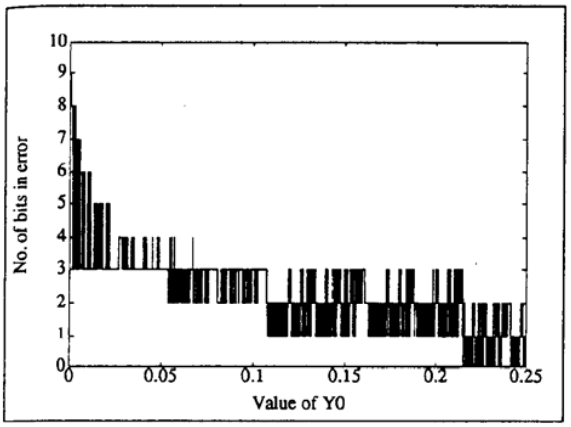


Figure 1: Error in the computed value of $\tan^{-1}(1)=\tan^{-1}(y_0/y_0)$ versus $y_0$ in a CORDIC module without normalization. The error has been quantized into the number of bits in error. A large error is observed for small values of $y_0$. This experiment has been performed on a datapath that is 16 bits wide. The selected range of $y_0$ forms about $\frac{1}{4}$th of the entire range of $y_0$.

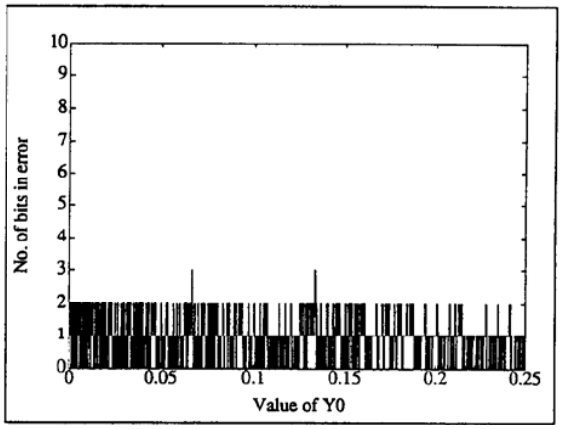


Figure 2: Error in the computed value of $\tan^{-1}(1) = \tan^{-1}(y_0/y_0)$ versus $y_0$ in a CORDIC module that implements the partial normalization scheme

**Lemma 2.1**

$$|\hat{\mathbf{v}}_n - \mathbf{v}_n| < \sqrt{2}\epsilon\left(1 + \sum_{j=0}^{n-1}\left[\prod_{l=j}^{n-1} K_l\right]\right) \tag{5}$$

**Proof** This is a result derived by Hu [5]. The quantity $\epsilon$ is the precision of the xy-data path. The scale factor $K_l$ is defined as $K_l = \prod_{i=0}^{l-1}\cos\alpha_i$. The above relation holds for fixed-point data paths. A plot of $\sum_{j=0}^{n-1}\left[\prod_{l=j}^{n-1} K_l\right]$ versus $n$ shows that it is very close to $n$ for all practical values of $n$ such as 16, 32 or 64. Thus the result reduces to:

$$|\hat{\mathbf{v}}_n - \mathbf{v}_n| < \sqrt{2}\epsilon(1+n) \tag{6}$$

The number of iterations, $n$, and the width of the data path are related. Since a wider datapath implies greater precision, an approximation error that is of the same order, can be achieved only through more iterations.

□

The truncation error due to finite precision of the $z$ datapath can be quantified by the following lemma.

**Lemma 2.2**

$$\alpha - \sum_{i=0}^{n-1}\delta_i \mathbf{Q}[\alpha_i] = \sum_{i=0}^{n-1}\delta_i\mu_i \quad \text{where} \quad \mu_i = \alpha_i - \mathbf{Q}[\alpha_i] \tag{7}$$

**Proof** The angle $\alpha$ is defined as $\alpha = \sum_{i=0}^{n-1}\delta_i\alpha_i$. Hence, $\alpha$ is the total angle that is either added to or subtracted from $z_0$ in a hypothetical CORDIC unit with infinite precision. The result follows from definitions of $\alpha$ and $\mu_i$. If a fixed-point representation is used, then all the $\mu_i$s are bounded above by $\mu$, the precision of the $z$-data path.

□

### 2.1 Perturbation in the Computed Value of the Inverse Tangent

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 is accumulated in $\hat{z}_n$. The angle actually accumulated in $\hat{z}_n$ is $\varphi$. This is achieved by performing iterations given by Equation 1, Equation 2 and Equation 3 with the choice of $\delta_i$ at each iteration given by Equation 4. The convergence property of the CORDIC algorithm guarantees that $\hat{y}_n$ is reduced to close to zero. A detailed error analysis [6] of Y-reduction shows that the approximation error forms an insignificant fraction of the final error and does not affect our results. Hence, in this paper we present a simplified analysis that neglects the approximation error.

In a hypothetical CORDIC unit with infinite precision, the following relation holds:

$$y_n = (y_0\cos\alpha - x_0\sin\alpha)/K_n \tag{8}$$

where $\alpha$ has been defined in lemma 2.2. Let $r$ and $\theta$ be defined as

$$r = \sqrt{x_0^2 + y_0^2}\ ;\ \tan\theta = \frac{y_0}{x_0}. \tag{9}$$

Equation 8 can now be rewritten as:

$$\begin{aligned} K_n y_n &= r\sin(\theta-\alpha) \\ |\theta-\alpha| &= \left|\sin^{-1}\left(\frac{K_n y_n}{\sqrt{x_0^2+y_0^2}}\right)\right| \end{aligned}$$

Using lemma 2.1, we can conclude that $y_n - \hat{y}_n$ is bounded by $(n+1)\sqrt{2}\epsilon$. Similarly, using lemma 2.2, we can conclude that $|\varphi-\alpha|$, which is the angle $|\hat{z}_n - z_n|$, is bounded by $n\mu$. Substituting these worst case bounds, we obtain the following relation

$$|\theta-\varphi| < \left|\sin^{-1}\left(\frac{K_n y_n}{\sqrt{x_0^2+y_0^2}}\right)\right| + n\mu \tag{10}$$

Here, $\theta$ 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 relation (10) gives the numerical error in inverse tangent calculations.

Figure 1 shows the errors observed in the computation of inverse tangent for various initial values. It shows that when $x_0$ and $y_0$ are close to zero, a large error results. An intuitive explanation of this error is the successively larger right shift in the CORDIC iterations given by Equation 1 and Equation 2, which can result in a loss of all significant bits if $x_0$ and $y_0$ are not large enough. A similar problem does not occur in a floating-point CORDIC module, 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 smaller since the initial values are, by definition, always in a normalized form.

## 3 Normalization for CORDIC Y-Reduction

Figure 1 shows that a large error in the computed inverse tangent can occur if $|\hat{v}_0|$ is small. However, if the initial values are bounded below by a value that is in the order of $2^n\epsilon$, since the numerator in Equation 10 is bounded above by $K_n\sqrt{2}(n+1)\epsilon$, the error will be bounded. Hence, a lower bound on $|\hat{v}_0|$, enforced through normalization of the input values, can be used to control the error. If this normalization takes the form of a left shift of both the inputs, it appears in the final output, $\hat{v}_n$, as a shift and does not affect the computed inverse tangent, except for better accuracy.

We observed the above problem when designing a processor [6] that computes the Singular Value Decomposition (SVD) of a matrix using CORDIC arithmetic [1]. In a fixed-point implementation, when CORDIC is used to perform operations on numbers obtained in a long chain of calculations, one cannot guarantee that the numbers are always in a normalized form. A straightforward way to eliminate the resulting numerical problems is to explicitly normalize the inputs before proceeding with CORDIC.

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 uses two leading-zero encoders to determine the amounts through which each component, $\hat{x}_0$ and $\hat{y}_0$, need to be shifted; a comparator to choose the smaller of the two shifts; and a barrel shifter to implement the shifts. This generic solution to achieve normalization incurs an $O(1)$ time penalty and $O(n^2)$ area penalty.

A tradeoff between time and area [8] can be made by implementing a shifter that can shift through selected powers of two, and performing the desired normalization shift in $\log n$ cycles. Such an approach reduces the complexity of the barrel shifter to $O(n \log n)$, while the complexity of the leading-zero encoders is still $O(n^2)$. The implementation is considerably more complex and incurs a time penalty of $O(\log n)$.

In this section we present a new normalization technique that incurs an area penalty of $O(n^{1.5})$ but no time penalty. This technique, which 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 $\hat{x}_i$ and $\hat{y}_i$ 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 any iteration. The normalization shift introduced at every iteration prevents CORDIC from shifting out and losing significant bits in 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. The following theorem quantifies this idea.

**Theorem 3.1** *Let $t' = \tan^{-1}(\hat{y}_0 2^q / \hat{x}_0 2^q)$, be the value calculated by preshifting $\hat{x}_0$ and $\hat{y}_0$ and using an unmodified* CORDIC *unit, where $q$ is an integral multiple of a constant $p$. Let $t = \tan^{-1}(\hat{y}_0/\hat{x}_0)$ be the value calculated by using a modified* CORDIC *unit and unnormalized initial values. The modified* CORDIC *unit initially performs $q/p$ iterations of the form*

$$\hat{x}_{i+1} = Q\left[\hat{x}_i + \delta_i \hat{y}_i 2^{-i}\right] 2^p \qquad (11)$$

```
begin
  for i := 0 to n do
    if (x_i < MAX_X and y_i < MAX_Y) then
      Choose xindex such that
        x_i 2^shift[xindex] < MAX_X and
        x_i 2^shift[xindex+1] > MAX_X ;
      Choose yindex such that
        y_i 2^shift[yindex] < MAX_Y and
        y_i 2^shift[yindex+1] > MAX_Y ;
      index := min( xindex, yindex );
    else
        index := 0;
    end
    Normalization Shift p := shift[index]
    Perform modified CORDIC Iteration;
  end
end
```

Figure 3: Modified CORDIC Iterations, invoked during Y-reduction when the input is not normalized

$$\hat{y}_{i+1} = Q\left[\hat{y}_i - \delta_i \hat{x}_i 2^{-i}\right] 2^p, \qquad (12)$$

*where $i$ is the iteration index and $Q[.]$ is the quantization operator. This is followed by $(n - q/p)$ unmodified* CORDIC *iterations. The two inverse tangents computed, $t$ and $t'$, will be identical if $q < \xi$, where $\xi$ is an upper bound given by*

$$\xi = 2p^2$$

**Proof** This result is obtained by observing that the left shift of $p$ introduced at the end of a CORDIC iteration introduces $p$ zeros in the lower order bits that can be safely shifted out without losing any significant bits. However, as the iteration index increases, the right shift at each iteration increases, while the left shift introduced by normalization is a constant. The upper bound $\xi$ is the point at which the $p$ is no longer sufficient to prevent any loss of bits. A detailed proof of this result can be found in a thesis [6]. □

This theorem shows that if the CORDIC iterations can be modified to include a constant left shift $p$ then normalization through any multiple of $p$ but less than $2p^2$ can be achieved. In practice, the parameter $p$ 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 $p$ in Equation 11 can be chosen from a total of $\lambda$ shifts, and the condition $\text{shift}[\lambda - 1] > \text{shift}[\lambda - 2] > \cdots \geq 0$ holds. The algorithm given in Figure 3, 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.

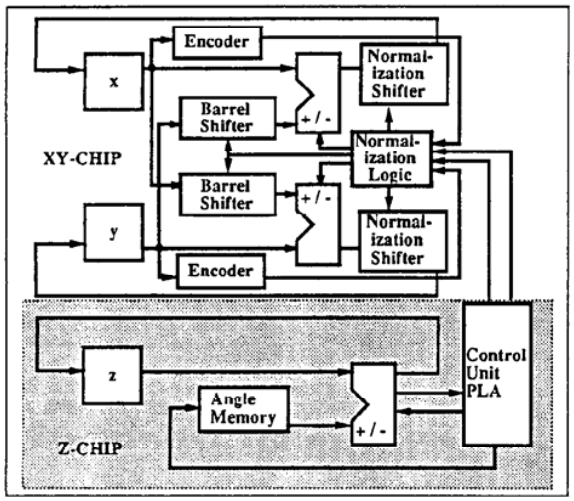


Figure 4: Implementation of the CORDIC Iterations in Hardware

### 3.1 Choice of a Minimal Set of Shifts

The following is an example of a CORDIC unit with a data-width $w = 16$ bits and $n = 16$ iterations. The higher order three bits correspond to one sign bit and two overflow bits. Hence, for this example, the parameters MAX_X and MAX_Y are set to the values of $x$ and $y$ corresponding to the bit pattern "0010 0000 0000 0000". The algorithm in figure 3 chooses the largest of {0,1,3} shifts, such that the resulting bit string after, has three leading zeros (or leading ones if negative).

- To permit unmodified iterations in a modified CORDIC unit a shift of zero is necessary $\Rightarrow shift[0] = 0$.
- An overall normalization shift of 1 can be achieved only by choosing $p = 1$. Thus a shift of 1 is necessary in any implementation $\Rightarrow shift[1] = 1$.
- Multiple iterations with $p = 1$ can achieve a shift up to a maximum of $2p^2 = 2$. A shift of 3, however, cannot be implemented as three modified CORDIC iterations with $p = 1$, since this will result in a loss of significant bits. This necessitates the inclusion of the shift $p = 3$ to the set $\Rightarrow shift[2] = 3$.
- The shift $p = 3$ can achieve any shift up to a maximum $2p^2 = 18$. 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, q, which is not a multiple of 3 is performed as shifts of 3 for $\lfloor q/3 \rfloor$ iterations, followed by $q \bmod 3$ iterations with $p = 1$.

Figure 4 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 Figure 2.

### 3.2 Cost of Partial Normalization Implementation

For a CORDIC implementation with the data width of the $x$ and $y$ datapaths $w$ bits, a shifter with a maximum shift $\sqrt{w/2}$ is required. The shifts at each iteration are obtained from leading-zero encoders that encode the $\sqrt{w/2}$ most significant bits of the $x$ and $y$ variables. The costs involved in these operations are:

**Shifter:** The area complexity [8] of the shifter grows as $O(w \times \text{Maximum Shift required}) = O(w \times \sqrt{w}) = O(w^{1.5})$.

**Control Logic:** The size of the control logic grows as $O((\text{Maximum Shift required})^2) = O(\sqrt{w}^2) = O(w)$.

**Time Penalty:** Since the shifting is performed as part of the CORDIC iterations, the only time penalty incurred is a decrease in the clock rate due to the extra propagation delay caused by the presence of an additional shifter in the CORDIC data path.

## 4 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. The complexity of a floating-point implementation of the entire system can be avoided 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 does not affect the latency. We believe this scheme can be extended to the other modes of CORDIC.

## Acknowledgements

This work was supported in part by the National Science Foundation under Research Initiation Award MIP-8909498.

## References

[1] J. R. Cavallaro and F. T. Luk. CORDIC Arithmetic for an SVD Processor. *Journal of Parallel and Distributed Computing*, 5(3):271–290, June 1988.

[2] J. R. Cavallaro and F. T. Luk. Floating-Point CORDIC for Matrix Computations. *IEEE Int. Conf. on Computer Design*, pages 40–42, October 1988.

[3] M. D. Ercegovac and T. Lang. Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD. *IEEE Trans. Computers*, 39(6):725–740, June 1990.

[4] X. Hu, R. G. Harber, and S. C. Bass. Expanding the Range of the CORDIC Algorithm. *IEEE Trans. on Computers*, pages 13–21, Jan. 1991.

[5] Y. H. Hu. The Quantization Effects of the CORDIC Algorithm. *Intl. Conf. on Accoustics, Speech and Signal Processing*, pages 1822–1825, April, 1988.

[6] K. Kota. Architectural, Numerical and Implementation Issues in the VLSI Design of an Integrated CORDIC SVD Processor. Master's thesis, Rice University, Department of Electrical and Computer Engineering, May 1991.

[7] T. Y. Sung, Y. H. Hu, and H. J. Yu. Doubly Pipelined CORDIC Array for Digital Signal Processing Algorithms. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 2:1169–1172, April 1986.

[8] J. D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, Rockville, MD, 1984.

[9] J. S. Walther. A Unified Algorithm for Elementary Functions. *AFIPS Spring Joint Computer Conf.*, pages 379–385, 1971.

[10] B. Yang and J. F. Böhme. Reducing the Computations of the SVD Array Given by Brent and Luk. *SPIE Advanced Algorithms and Architectures for Signal Processing*, 1152:92–102, August, 1989.