On-Line CORDIC Algorithms

Hai Xiang Lin
Hank J. Sips

Reprinted from
IEEE TRANSACTIONS ON COMPUTERS
Vol. 39, No. 8, August 1990

On-Line CORDIC Algorithms

HAI XIANG LIN, STUDENT MEMBER, IEEE, AND HENK J. SIPS, MEMBER, IEEE

Abstract—**The CORDIC algorithms provide in a fast way the calculations of a number of arithmetic basic functions. A CORDIC calculation takes $O(n)$ steps for a function, where n is the word length of the operands. The speed is limited by the carry propagation in the adders and the I/O throughput. Speed can be improved by introducing redundancy in the calculation circuit and I/O throughput by doing I/O transfers while calculating. The latter is characteristic for the class of so called on-line arithmetic. At the same time, the pin requirements are limited to a single digit per operand. This paper introduces a number of new algorithms to make an on-line CORDIC implementation. The on-line CORDIC algorithm takes $n + 6$ clock cycles to compute a CORDIC function. It is estimated that an implementation of the proposed algorithm is 6 to 7 times as fast as the traditional CORDIC approach for Givens' rotation, and 7 to 8 times as fast for SVD computation.**

Index Terms—**Computer arithmetic, CORDIC, digit pipelining, on-line algorithms.**

I. Introduction

IN ON-LINE computations, the input operands and results flow through arithmetic units in a digit by digit manner, starting with the most significant digit. An on-line algorithm is said to have an on-line delay of δ, if for the generation of the jth digit of the result, $(j + \delta)$ digits of the input operands are required [1]. Fig. 1 shows the principle of on-line computation and illustrates that digit on-line arithmetic units can be chained to obtain a fast throughput of digit-serial operands. In recent years, many algorithms for the on-line generation of the basic arithmetic functions: addition, multiplication, division, and square root have been considered [2]–[8].

The on-line generation of transcendental functions turns out to be more difficult. A traditional known fast method for calculating these functions is the CORDIC method [9], [10]. Operations of the type $a \cdot \cos\theta + b \cdot \sin\theta$, with $\theta = \tan^{-1}(y/x)$, occur frequently in rotation based algorithms for matrix triangularization, SVD (singular value decomposition), image processing, etc. In [11], Cavallaro and Luk describe a CORDIC implementation for solving the SVD problem. Their CORDIC-based 2×2 SVD-processor is twice as fast as one assembled from traditional hardware units (divider, square-rooter). In [12], Ercegovac and Lang describe a CORDIC implementation aimed at matrix triangularization and SVD with internal

Manuscript received November 2, 1989; revised March 5, 1990. This paper was presented in part at the 9th Symposium on Computer Arithmetic, Santa Monica, CA, September 6–8, 1989.

H. X. Lin is with the IBBC-TNO Netherlands Organization for Applied Scientific Research, Rijswijk, The Netherlands.

H. J. Sips is with the Department of Applied Physics, Delft University of Technology, Delft, The Netherlands.

IEEE Log Number 9036140.

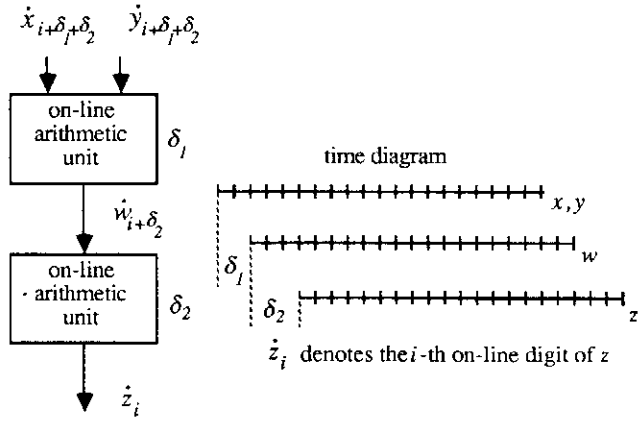

Fig. 1. Chained on-line computation.

on-line arithmetic. For SVD their approach is estimated to be 4 times as fast as the approach in [11], since the use of redundant adders allows for a faster clock period.

In this paper, we introduce an on-line CORDIC algorithm with an on-line delay δ of $6 \leq \delta \leq 8$ (independent of the word length n of the operands). It is also shown that by using the proposed scheme, Givens' rotation and the diagonalization of a 2×2 matrix in SVD computation can be computed in $n+13$ and $n+22$ clock cycles, respectively. The paper is organized as follows. In Section II, the arithmetic functions which can be calculated by the CORDIC algorithms are investigated on their on-line properties by using previously derived results. Section III states the modified CORDIC equations to allow for on-line input operands. In Section IV, the on-line delay characteristics are derived for $Z \rightarrow 0$ and $Y \rightarrow 0$. Thereafter, in Section V, the delays caused by the digitization of X and Y are derived. Section VI shows how the scaling factor K^{-1} can be generated. Section VII describes a proposed hardware organization of the on-line CORDIC processor. The internal data representation in relation to the accuracy of the computation results is considered. Finally, in Section VIII, the merits of the on-line CORDIC processor are discussed by means of the applicability to Givens' rotation and SVD.

II. On-Line Properties of Arithmetic Functions Suitable for CORDIC Generation

The CORDIC method is very efficient for the generation of transcendental functions. Besides that, also addition, multiplication, and square root can be implemented by using the same basic equations. Before going to the actual CORDIC algorithms, the basic on-line properties of the arithmetic functions under discussion are investigated. Independently of each other, Sips and Lin [13] and Duprat, Herreros, and Muller [14] have derived bounds on the on-line delays of arithmetic

functions. Moreover, in [13] it is shown that these bounds can be actually realized by using a table lookup implementation. This property is also used in this paper to generate the scaling factor K^{-1} (Section VI). The bounds can be stated as follows.

Let $F(X_i)$ be an arithmetic function on a set of N input numbers X_i, and let X_i be represented by a symmetric, radix-r redundant digit set $(-\eta, \cdots, -1, 0, 1, \cdots \eta)$, where η is the redundancy factor $(r/2 \leq \eta \leq r-1)$. The numbers X_i are recursively defined as

$$X_i = X_{i-1} + x_{i+\delta_j} \cdot r^{-i-\delta_j-1} \tag{1}$$

with

$$X_0 = \sum_{i=0}^{\delta_j - 1} x_i \cdot r^{-i-1} \qquad \text{where } j = \text{index}(x),\ 1 \leq j \leq N.$$

The elements δ_j, $1 \leq j \leq N$ of $\boldsymbol{\delta}$ represent the number of digits of the respective input operands which must be known to be able to generate the first digit of the result. From the definition of $\boldsymbol{\delta}$, it follows that the δ_j's might have different values. Therefore, we define $\delta_g = \max(\boldsymbol{\delta})$ as the *generation* on-line delay of $F(X_i)$.

Theorem 1 in [13] states that if $F(X)$ is continuous on $[X_A, X_B]$ and differentiable on (X_A, X_B), then $F(X)$ can be generated with an on-line delay of $\delta_g \geq \max(\boldsymbol{\delta})$ if

$$\Gamma(X_K) \leq r^{\varphi+1}\left\{1 - \frac{(r-1)}{2\eta}\right\} \tag{2}$$

where

$$\Gamma(X_K) = \max_{X_K}\{|\nabla F(X_K)| \cdot d\} \qquad \text{with } X_K \in [X_A, X_B]$$

$$d = (r^{-\delta_1}, r^{-\delta_2}, \cdots, r^{-\delta_N})$$

$$|\nabla F(X_K)| = (|\partial F(X_K)/\partial x_1|, \cdots, |\partial F(X_K)/\partial x_N|)$$

and φ is the position of the most significant (redundant) digit of the result.

In this paper, a few assumptions are made, mostly for reasons of simplicity. First, a fully redundant digit set is assumed, i.e, $\eta = r - 1$. In this case, (2) reduces to

$$\Gamma(X_K) \leq (1/2) \cdot r^{\varphi+1}. \tag{3}$$

Second, it is assumed that the input operands are normalized: $r^{-1} \leq |X| < 1$. For some input operands, representing for instance angles, it holds $|X| < \pi/2$. For quasi-normalized input operands, i.e., $r^{-2} \leq |X| < 1$, which can occur in chained on-line computations [13], the derivations can be done in a similar way.

From (3) the on-line delay δ can be derived as $\delta = \lceil \log_r (\text{Max } |\nabla F(X_K)|) + \log_r 2 \rceil - 1 - \varphi$. In [14], the derived bounds on the on-line delays are $\lceil \log_r (\text{Max } |\nabla F(X_K)|) \rceil \leq \delta \leq \lceil \log_r (\text{Max } |\nabla F(X_K)|) \rceil + 1$, under the assumption $F{:}[-1, 1] \rightarrow [-1, 1]$. For the interval assumption $F{:}[-1, 1] \rightarrow [-1, 1]$ it holds $\varphi = 0$. Therefore, the bound in [13] is equal or slightly sharper than the one in [14]. The delays δ are the generation delays. When the outputs must be within a certain range (i.e., normalized) one can either prescale the inputs or postnormalize the results. In [14], the inputs are prescaled to satisfy the output interval assumptions, whereas in [13] the results are postnormalized. The prescaling or postnormalization causes an extra on-line delay.

TABLE I
ON-LINE GENERATION DELAYS OF SEVERAL ARITHMETICAL FUNCTIONS, IF GENERATED BY THE TABLE LOOKUP SYSTEM

function	on-line delay δ	1-st digit φ	argument range
$sin(x)$, $cos(x)$	0	0	$\lvert x\rvert \leq \pi/2$
$ln(x)$	1	0	$r^{-1} \leq x \leq 1$
e^x	0 (r=2) 0 (r>3)	1 0	$x < ln2$
$x.sin\phi + y.cos\phi$	2 (r=2) 1 (r=3,4) 0 (r>4)	0 0 0	$r^{-1} \leq x,y < 1$
$tan^{-1}(y/x)$	1 (r=2) 1 (r=4) 0 (r>4)	1 0 0	
$(x^2+y^2)^{1/2}$	1 (r=2) 0 (r≥3)	0 0	$r^{-1} \leq x,y < 1$
$(x^2-y^2)^{1/2}$	variable		
$z+y/x$	2 (r=2) 1 (r=3) 0 (r≥4)	1 1 1	$r^{-1} \leq x,y,z < 1$

For the relevant functions in this paper, the on-line delays, derived according to (3), are listed in Table I. This table shows that very low on-line delays exist for these functions.

Moreover, it is shown in [13] that the above arithmetic functions can be realized by means of a table lookup implementation as shown in Fig. 2. The digits of the input operands provide the address for the table and two output digits suffice to generate the respective digits of the on-line result. For multiple input operands this soon becomes very impractical. For single argument functions, a 4 megabit address space facilitates the generation of 18–22 bit results, depending on the on-line delay factor. So with a simple single memory chip already reasonable accurate single argument functions can be realized.

III. The On-Line CORDIC Equations

The CORDIC method is based on vector rotations [9], [10], where a new vector $P_{i+1} = (X_{i+1}, Y_{i+1})$ is obtained from $P_i = (X_i, Y_i)$ according to

$$X_{i+1} = X_i + m\rho_i Y_i 2^{-i} \tag{4a}$$

$$Y_{i+1} = Y_i - \rho_i X_i 2^{-i} \tag{4b}$$

$$Z_{i+1} = Z_i - \rho_i \theta_i \tag{4c}$$

where $m \in \{-1, 0, 1\}$ is the parameter for the coordinate system, and $\rho_i \in \{-1, 1\}$. Although the equations be formulated for a general radix, we will restrict the discussion to $r = 2$, for reasons of convenience. Various arithmetic functions can be calculated by forcing Z or Y to zero, by choosing the appropriate values of ρ_i in each iteration cycle. The values of θ_i are found by $\theta_i = m^{-1/2} \tan^{-1} [m^{1/2} \cdot 2^{-i}]$. The results calculated

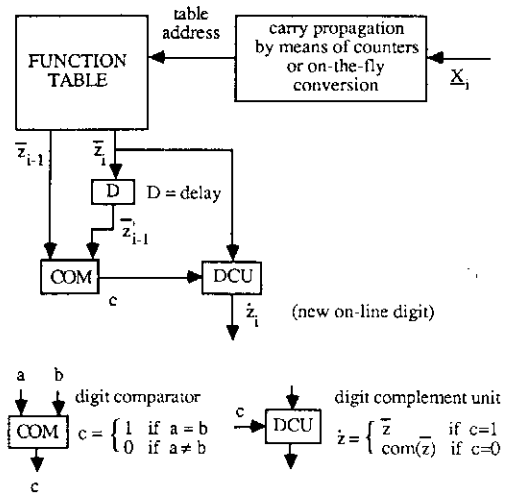

Fig. 2. Block diagram of an on-line table lookup system.

in (4a) and (4b) have to be corrected through multiplication by the following scaling factor

$$K^{-1} = \prod_{i=i_m}^{n-1} (1 + m\phi_i^2)^{-1/2} \qquad \text{with } \phi_i = \rho_i 2^{-i} \tag{5}$$

where $i_m = 0$ for $m = 1$ and $i_m = 1$ for $m = -1$ ($K^{-1} = 1$ for $m = 0$).

For the traditional CORDIC algorithms this scaling factor is constant and can be calculated in advance. Some other schemes use a slight modification of the iteration scheme in order to simplify the factor (e.g., to force the factor to be a power of 2), such that a multiplication is not necessary and the corrections can be done by simple additions in each iteration [15], [16].

A number of authors have published on-line versions of some of the CORDIC algorithms. Owens [8] uses a simplified CORDIC scheme to calculate $\sin(x)$ with an on-line delay of $\delta = 3$ for $r = 8$. However, the result is unscaled [(2)], so it cannot be used directly for further calculations. Ercegovac and Lang [12] describe two chained CORDIC schemes to perform Givens' rotation, i.e., $a \cdot \sin(\theta) + b \cdot \cos(\theta)$, with $\theta = \tan^{-1}(x/y)$. The first CORDIC scheme computes the angle which is transmitted in decomposed form, i.e., a sequence of ρ_i values, to the CORDIC module which computes the rotation. The total latency time of their implementation is $3n + 3$ clock cycles.

In the on-line case, the CORDIC equations of (4) are transformed to the following set of equations

$$X'_{i+1} = X'_i + m\rho_i Y'_i \cdot 2^{-i} + \Delta X_{i+1}(x_{i+\delta}, y_{i+\delta}) \tag{6a}$$

$$Y'_{i+1} = Y'_i - \rho_i X'_i \cdot 2^{-i} + \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta}) \tag{6b}$$

$$Z'_{i+1} = Z'_i - \rho_i \theta_i + z_{i+\delta} \cdot 2^{-(i+\delta)} \tag{6c}$$

where $x_{i+\delta}$, $y_{i+\delta}$, and $z_{i+\delta}$ are the new input digits of X, Y, and Z, respectively, at the ith iteration, X'_0, Y'_0, and Z'_0 are set to X_0, Y_0, and Z_0, respectively, according to (1). The terms ΔX_{i+1} and ΔY_{i+1} are the correction terms to correct the error due to the absence of $x_{i+\delta}$, $y_{i+\delta}$, and $z_{i+\delta}$ in the iterations $0, 1, \cdots, i-1$. The calculation of ΔX_{i+1} and ΔY_{i+1} is considered in Appendix A.

IV. On-Line Characteristics for $Z \to 0$ and $Y \to 0$

A. $Z \to 0$

To compute arithmetic functions like $x \cdot \cos(z) - y \cdot \sin(z)$ ($m = 1$), or $x \cdot \cosh(z) + y \cdot \sinh(z)$ ($m = -1$), Z'_i is forced to zero. If we define $W_i = 2^i \cdot Z'_i$, (6c) can be rewritten as

$$W_{i+1} = 2(W_i + z_{i+\delta} \cdot 2^{-\delta} - \rho_i \cdot \alpha_i) \tag{7}$$

where $\alpha_i = 2^i \cdot \theta_i$ can be directly read from the constant angles table. The value of ρ_i is determined by $W_i + z_{i+\delta} \cdot 2^{-\delta}$. Since full carry propagation in an addition will be avoided, only the most significant L digits of $W_i + z_{i+\delta} \cdot 2^{-\delta}$, denoted as W_i^*, are inspected. It holds that

$$|W_i + z_{i+\delta} \cdot 2^{-\delta} - W_i^*| < 2^{-(L-\varphi-1)} \tag{8}$$

where φ is the position of the most significant digit of W_i. Combining (7) and (8) it follows that

$$|W_{i+1}| < 2 \cdot |W_i^* - \rho_i \cdot \alpha_i| + 2^{-(L-\varphi-2)}. \tag{9}$$

We must find an appropriate selection of ρ_i such that Z'_{i+1} converges. Suppose a selection function of the form

$$\rho_i = \begin{cases} 1 & \text{if } W_i^* \geq b \\ 0 & \text{if } |W_i^*| < b \\ -1 & \text{if } W_i^* \leq -b \end{cases} \tag{10}$$

is chosen, where b is a positive constant ($0 < b \leq 1$). For the value of b a simple value, like some power of two, is to be preferred to simplify the selection process. The choice of b must be such that Z'_{i+1} converges to zero, i.e., W_{i+1} is bounded by a fixed positive constant.

First, we consider the case of circular rotation, i.e., $m = 1$. It can be shown that to ensure the convergence of Z'_{i+1} it requires $\delta = 2$. Furthermore, with $\delta = 2$, $b = 1/2$, and $L = 5$ ($\varphi = 0$), it holds $|W_i| < 2$ for all i. The proof of this is lengthy, but straightforward, so it suffices to give the procedure. W_0 consists of the first two digits of the input Z, therefore it holds $|W_0| \leq 1.5$. By considering all possible combinations of the first four most significant input digits, it follows that $|W_1| \leq 1.93$ and $|W_2| \leq 1.51$. Now $\theta_i = \tan^{-1}[2^{-i}]$, so by using a Taylor series-expansion which is truncated and rounded up- and downwards, it holds that

$$2^{-i} - (1/3)2^{-3i} + (1/35)2^{-5i} \leq \theta_i \leq 2^{-i} - (1/3)2^{-3i} + (1/5)2^{-5i}$$

or

$$1 - (1/3)2^{-2i} + (1/35)2^{-4i} \leq \alpha_i \leq 1 - (1/3)2^{-2i} + (1/5)2^{-4i}.$$

By using this, it can be shown by induction that

$$2 \cdot |W_i^* - \rho_i \cdot \alpha_i| + 2^{-(L-2)} \leq 1.51, \quad \text{if } |W_i| \leq 1.51 \text{ for } i \geq 2.$$

From the definition $W_i = 2^i \cdot Z_i'$ and $|W_i| < 2$, it follows that $Z_{n+1}' < 2^{-(n-1)}$. Thus, the convergence of the angle rotation of (4c) is guaranteed for $m = 1$.

For $m = -1$ (hyperbolic rotation), a similar analysis as for $m = 1$ can be made. It can be shown that using the same selection function [(10)] it holds $|W_i| < 1.5$ ($|W_0| < 1.0$), for $\delta = 2$, $b = 1/2$, and $L = 4$.

B. $Y \rightarrow 0$

To compute the functions $z + \tan^{-1}(y/x)$ and $(x^2 + y^2)^{1/2}$ (when $m = 1$), or $z + \tanh^{-1}(y/x)$ and $(x^2 - y^2)^{1/2}$ (when $m = -1$), Y has to be forced to zero. For (6b), it holds that the convergence is guaranteed if $|Y_{i+1}| \leq |X_i| \cdot 2^{-i}$ (X_i is bounded, see [10]). The equation to be forced to zero is (6b), i.e.,

$$Y_{i+1}' = Y_i' - \rho_i \cdot X_i' \cdot 2^{-i} + \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta}). \quad (11)$$

The choice of ρ_i is determined by the value of Y_i'. Let $V_i = 2^i \cdot Y_i'$, and assume again that only the L most significant digits of V_i are used for the selection and denote them as V_i^*. Then it holds that $V_i = V_i^* + \epsilon(V)$, with $\epsilon(V)$ being the truncation error and $|\epsilon(V)| \leq 2^{-(L-1)}$.

The selection function for ρ_i is defined as

$$\rho_i = \begin{cases} 1 & \text{if } V_i^* \geq b \\ 0 & \text{if } |V_i^*| < b \\ -1 & \text{if } V_i^* \leq -b \end{cases} \quad (12)$$

where b is a positive constant ($0 < b \leq 1$). The selection of ρ_i can simply be done by comparing the L most significant digits of V_i^* with b. The value of b must be chosen such that the convergence criterion $|Y_{i+1}| \leq |X_i| \cdot 2^{-i}$ is satisfied. For the circular CORDIC, i.e., $m = 1$, two cases can be considered:

1) $\rho_i = 0$: This implies that $|V_i^*| < b$ or $|Y_i'| < b \cdot 2^{-i} + 2^{-(i+L-1)}$. Since it holds (see Appendix B)

$$\left| \sum_{k=i+\delta}^{n} \Delta Y_{i+1}(x_k, y_k) \right| < 2.65 \cdot 2^{-(i+\delta+1)} \quad (13)$$

therefore,

$$|Y_{i+1}| = \left| Y_i' + \sum_{k=i+\delta}^{n} \Delta Y_{i+1}(x_k, y_k) \right| \leq b \cdot 2^{-i} + 2^{-(i+L-1)} + 2.65 \cdot 2^{-(i+\delta+1)}. \quad (14)$$

Since $X_i \geq 1/2$ [12] for $i \geq 1$, from (14), the condition $|Y_{i+1}| \leq X_i \cdot 2^{-i}$ is satisfied if

$$b \cdot 2^{-i} + 2^{-(i+L-1)} + 2.65 \cdot 2^{-(i+\delta+1)} \leq (1/2) \cdot 2^{-i}.$$

Therefore, the values of the parameters b, δ, and L must be chosen to satisfy the following inequality

$$b \leq 1/2 - 2^{-(L-1)} - 2.65 \cdot 2^{-(\delta+1)}. \quad (15)$$

2) $\rho_i = \pm 1$: Assume that $\rho_i = +1$, i.e., $V^* \geq b$ (the same result follows for $V^* \leq -b$). With the substitution of

$$X_i' = X_i - \sum_{k=i+\delta}^{n} \Delta X_i(x_k, y_k)$$

into (6b), it follows

$$Y_{i+1} = Y_i' - X_i \cdot 2^{-i} + \sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)]. \quad (16)$$

It can be shown that

$$\left| \sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)] \right| \leq 3 \cdot 2^{-(i+\delta)} \quad \text{for } i \geq 0$$

(see Appendix B). Thus, from (16) it holds

$$Y_{i+1} \geq b \cdot 2^{-i} - 2^{-(i+L-1)} - X_i \cdot 2^{-i} - 3 \cdot 2^{-(i+\delta)}$$

and

$$Y_{i+1} \leq b \cdot 2^{-i} + 2^{-(i+L-1)} - X_i \cdot 2^{-i} + 3 \cdot 2^{-(i+\delta)}.$$

Therefore, it holds $|Y_{i+1}| \leq X_i \cdot 2^{-i}$, if

$$b \geq 2^{-(L-1)} + 3 \cdot 2^{-\delta} \quad (17)$$

and

$$b \leq 1 - 2^{-(L-1)} - 3 \cdot 2^{-\delta}. \quad (18)$$

From the analysis in 1) and 2) [(15), (17), and (18)], it can be concluded that the choice of $b = 1/4$, $\delta = 4$, and $L = 5$ satisfies the convergence criterion.

For $m = -1$, a similar analysis can be made. For arbitrary values of the inputs no fixed on-line delay can be derived, due to the characteristics of the resulting arithmetic function. However, for special cases a better result can be given. As an example, for the function $\ln(w)$, it holds $\ln(w) = 2 \cdot \tanh(Y/X)$ with $X = w + 1$ and $Y = w - 1$, restricting the ranges of the operands to $1.5 \leq X < 2.0$ and $-0.5 \leq Y < 0.0$ for $0.5 \leq w < 1.0$. Now it holds $X_i \geq K \cdot \text{sqrt}(2) = 1.13$,

$$\left| \sum_{k=i+\delta}^{n} \Delta Y_{i+1}(x_k, y_k) \right| < (2.41) \cdot 2^{-(i+\delta)}$$

and

$$\left| \sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)] \right| < 5 \cdot 2^{-(i+\delta)}.$$

So the following equations can be derived:

$$b \leq 1.13 - 2^{-(L-1)} - 2.41 \cdot 2^{-\delta} \quad (19)$$

$$b \geq 2^{-(L-1)} + 5 \cdot 2^{-\delta} \tag{20}$$

$$b \leq 2.26 - 2^{-(L-1)} - 5 \cdot 2^{-\delta}. \tag{21}$$

Therefore, with $b = 1/2 + 1/4$, $\delta = 3$, and $L = 5$ the convergence criterion is satisfied.

V. Digitizing the Iteration Results X_i', Y_i', and Z_i'

The values of X_i', Y_i', and Z_i' are the results of the CORDIC calculation. These values are stored in redundant form and are driven to their final value in an iterative way and do not directly give the $(i - \delta)$th digit at the ith iteration. In order to produce an on-line result, these values must be transformed into an incremental representation where each iteration a new digit of the result is produced. This procedure is called digitization (e.g., see [7]).

A. The Digitization of X_i' and Y_i'

Let $\dot{Y}_{i-d}$ (the same consideration holds for $\dot{X}_{i-d}$) be the on-line digitized result of Y_{i+1}' and $R_{i+1} = 2^{i+1} \cdot (Y_{i+1}' - \dot{Y}_{i-d})$ be the (scaled) residue (d is the on-line delay for digitization). Suppose $\dot{y}_0$ has a positional weight of $2^{\varphi+d}$ after d initial shifts (not to be confused with the real weight 2^{φ}), then by using a symmetrical rounding function up to the $(\varphi + d)$th digit we obtain

$$\dot{y}_{i-d} = \text{Round}_{\varphi+d}(R_i) \qquad i \geq d,$$

$$\dot{y}_{i-d} = 0 \qquad 0 \leq i < d$$

$$R_{i+1} = 2(R_i + D(Y_i') - 2^{\varphi+d} \cdot \dot{y}_{i-d}), \quad \text{with } R_0 = Y_0' \tag{22}$$

where $D(Y_i') = 2^i \cdot (Y_{i+1}' - Y_i') = -\rho_i \cdot X_i' + 2^i \cdot \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})$. It can be ensured that

$$|R_i - 2^{\varphi+d} \cdot \dot{y}_{i-d}| < (1/2)2^{d+\varphi} + 2^{d+\varphi-L+1}, \qquad (L \geq 2). \tag{23}$$

What remains is that the digitizer delay d and the truncation length L must be chosen such that $\dot{y}_{i-d}$ is a single digit.

For $m = 1$, it holds that $|X_i'| \leq \text{sqrt}(2) \cdot K \leq 2.34$, for $|X| < 1$ and $|Y| < 1$, and $|\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})| \leq 2.65 \cdot 2^{-(i+\delta+1)}$ (see Appendix B). With $\delta = 2$, it follows $|D(Y_i')| \leq 2.34 + 2.65 \cdot 2^{-3} < 2.67$. Therefore, it holds with $\varphi = 1$ (remember that φ denotes the position of the most significant digit) that

$$|2(R_i + D(Y_i') - 2^{\varphi+d} \cdot \dot{y}_{i-d})| < 2^{d+1} + 2^{d-L+3} + 2.67 \cdot 2 \tag{24}$$

$\dot{y}_{i+1-d}$ is a single digit, if it holds that

$$|R_{i+1}| = |2(R_i + D(Y_i') - 2^{\varphi+d} \cdot \dot{y}_{i-d})| < 1.5 \cdot 2^{d+1} - 2^{d-L+2}.$$

Therefore, the on-line condition becomes

$$(2.67 \cdot 2^{-d} + 3 \cdot 2^{-(L-1)}) \leq 1/2.$$

This inequality is satisfied by choosing $d = 3$ and $L = 5$.

For $m = -1$, it holds that $|X_i'| \leq 2.72$, and $|\Delta Y_{i+1}'(x_{i+\delta}, y_{i+\delta})| \leq 2.41 \cdot 2^{-(i+\delta)}$. Equation (24) is transformed into (i starts from 1, unlike the case of $m = 1$ where i starts from 0, therefore it holds $R_{i+1} = 2^i \cdot (Y_{i+1}' - \dot{Y}_{i-d})$,

$$|2(R_i + D(Y_i') - 2^{\varphi+d} \cdot \dot{y}_{i-d})| < (1/2) \cdot 2^{d+2} + 2^{d-L+3} + 3.32 \cdot 2 \tag{25}$$

$\dot{y}_{i-d}$ is a single digit, if it holds that $|R_{i+1}| = |2(R_i + D(Y_i') - 2^{\varphi+d} \cdot \dot{y}_{i-d})| < 1.5 \cdot 2^{d+1} - 2^{d-L+2}$. Therefore, the on-line condition becomes $(3.32 \cdot 2^{-d} + 3.2^{-(L-1)}) \leq 1/2$. This inequality is satisfied by chosing $d = 3$ and $L = 6$.

B. The Digitization of Z_i'

In case of $Y_i' \to 0$, the result Z_i' needs to be digitized for on-line output. Let $R_i = 2^i \cdot (Z_i' - \dot{Z}_{i-1-d})$, we use a slightly different value for determining $\dot{z}_{i-d}$, which is,

$$\dot{z}_{i-d} = \text{Round}_{\varphi+d}(R_i + z_{i+\delta} \cdot 2^{-\delta}) \qquad i \geq d$$

$$\dot{z}_{i-d} = 0 \qquad 0 \leq i < d$$

$$R_{i+1} = 2(R_i + z_{i+\delta} \cdot 2^{-\delta} - \rho_i \cdot \alpha_i - 2^{\varphi+d} \cdot \dot{z}_{i-d}). \tag{26}$$

For $m = 1$, it holds $|\alpha_i| = 2^i \cdot |\tan^{-1}(2^{-i})| \leq 1$ for $i \geq 0$. With $\varphi = 1$ ($|Z_n'| \leq \pi/2$), it follows

$$|R_{i+1}| \leq |2(R_i + z_{i+\delta} \cdot 2^{-\delta} - 2^{\varphi+d} \cdot \dot{z}_{i-d})| + 2 \cdot |\alpha_i| \leq 2^{d+1} + 2^{d-L+3} + 2 \tag{27}$$

$\dot{z}_{i+1-d}$ is a single digit, if

$$|R_{i+1} + z_{i+1+\delta} \cdot 2^{-\delta})| \leq 1.5 \cdot 2^{d+1} - 2^{d-L+2}.$$

From (27), the on-line condition becomes ($\delta = 4$ for $Y \to 0$)

$$3 \cdot 2^{-L+2} + 2^{-d+1} + 2^{-d-4} \leq 1.$$

With $d = 2$ and $L = 5$ the on-line condition is satisfied.

For $m = -1$, it holds $|\alpha_i| = 2^i \cdot |\tanh^{-1}(2^{-i})| \leq (10/9)$ for $i \geq 1$. With $\varphi = 0$ ($|Z_n'| \leq \ln(2)$) and $Z = 0$, the on-line condition is satisfied if $3 \cdot 2^{-(L-2)} + (10/9) \cdot 2^{-d+1} + 2^{-d-4} \leq 1$. With $d = 2$ and $L = 5$ this condition is satisfied.

VI. On-Line Generation of the Scaling Factor K^{-1}

To obtain the final on-line output, the digitized results $\dot{X}_i$ and $\dot{Y}_i$ must be multiplied with the scaling factor K^{-1}. Unlike in conventional CORDIC, K^{-1} is not a constant since ρ_i can be zero in the on-line case, due to the use of noncarry propagating redundant adders. This implies that the scaling factor cannot be calculated in advance. A method for calculating K^{-1} has been described by Ercegovac and Lang [12]. They compute K^2 by the recurrence equation $K_{j+1}^2 = K_j^2 + m\phi_i^2 \cdot K_j^2$. In the implementation, the recurrence is unfolded by using

$n/2$ stages of on-line adders. These stages are followed by an on-line square-rooter and divider to obtain K^{-1}. Due to the unfolding, a relatively large on-line delay results, which in their implementation does not give an extra delay because it fits with the timing characteristics of the rest of the calculations. Another solution is to make the scaling factor a constant, by omitting the case $\rho_i = 0$. Techniques for this have been proposed by Takagi *et al.* [23]. One computes two different rotations at the same time, the other repeats some iterations. However, the proposed methods are not on-line and can only be applied for computing the sine and cosine functions.

Here a different method of calculating K^{-1} is shown. The method uses the table lookup approach, as described in Section II. This method has already been used in starting of an iterative method for calculating the reciprocal function [17]. If the number of input digits to the table is large, soon a very large table would be required. A method to reduce the table requirements is to partition the product terms of the scaling factor in groups (partial scaling factors) and to generate each partial scaling factor with the aid of a table lookup system. The partial scaling factors can then be combined to give the final function. Fig. 2 shows this principle of operation for K^{-1}. The following lemma states the on-line delay of the partial scaling function $K^{-1}(p, i_0)$.

Lemma: The partial scaling factor

$$K^{-1}(p, i_0) = \prod_{i=i_0}^{p} (1 + m \cdot \phi_i^2)^{-1/2}$$

with $m = \pm 1$ and $\phi_i = \rho_i \cdot 2^{-i}$ can be generated with an on-line delay of $\delta = 1 - 2i_0$, if $m = 1$ and $\delta = 2 - 2i_0$, if $m = -1$.

Proof: The on-line delay can be calculated by using (3) and by considering each of the ρ_i's as separate inputs. Details are given in Appendix C. □

Since $i_0 = 0$ for $m = 1$ and $i_0 = 1$ for $m = -1$, it follows that $K^{-1}(p, i_0)$ can be generated with an on-line delay of $\delta = 0$. Now consider the case of splitting K^{-1} into two subproducts

$$K^{-1} = \prod_{i=i_m}^{i_0-1} (1 + m \cdot \phi_i^2)^{-1/2} \cdot \prod_{i_0}^{n-1} (1 + m \cdot \phi_i^2)^{-1/2} \quad (28)$$

where $i_m = 0$, if $m = 1$, and $i_m = 1$, otherwise. From the results above it follows that a negative on-line delay results when generating the second partial scaling factor in (28) for $i_0 \geq 2$. This implies that the first $2i_0 - 1$ (or $2i_0 - 2$ for $m = -1$) digits of the second partial scaling factor are known before the first ρ_i becomes known. It can be shown that we have a number of the form $0.11\cdots11xx$ or $1.0\cdots0\bar{1}xx$ ($\bar{1}$ means -1) with $2i_0 - 1$ known digits for $m = 1$. The latter representation might be advantageous in a multiplier. For $m = -1$ the subproduct has the form $1.00\cdots00xx$ with $2i_0 - 2$ known digits.

The partitioning according to (28) can be further performed in the same way. The final scaling factor is formed by multiplying the subproducts. A single partial scaling factor can be easily generated on-line by using a table with the bit pattern

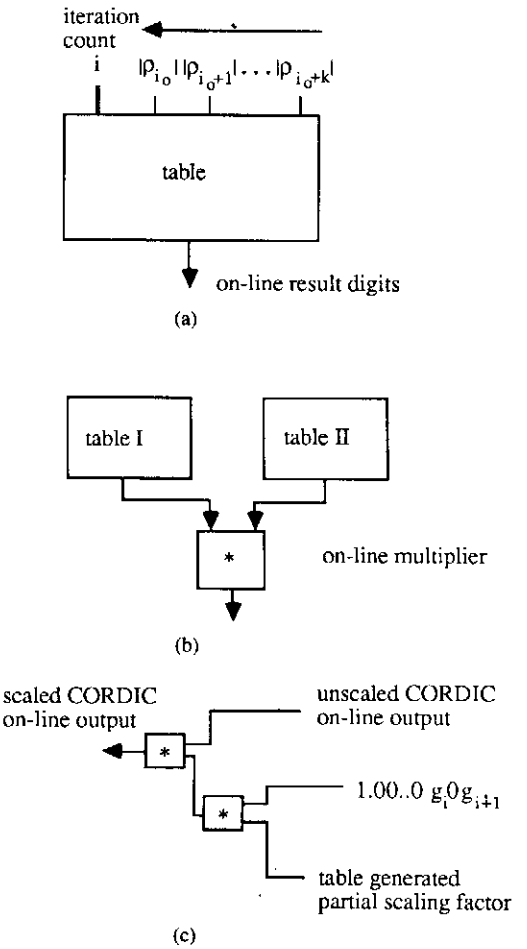

Fig. 3. (a) Table generation of (partial) scaling factor. (b) Generation of the scaling factor from the partial tables. (c) Scaling factor hardware organization.

$|\rho_{i0}||\rho_{i0+1}|\cdots|\rho_{i0+k}|$ as address, plus an iteration counter i [see Fig. 3(a)]. Subproducts are combined by using on-line multipliers [1], [2] [one for two subproducts, two for three subproducts, etc., see Fig. 3(b)].

For an accuracy of 2^{-n}, only $n/2$ product terms of (5) are required (see Appendix D). A further simplification can be obtained by observing that the higher $(n/4)$ half of the $n/2$ product terms ($i = n/4 + 1, \cdots, n/2$) can be reduced in complexity by deleting all cross products, because they are beyond the required accuracy, i.e., these $n/4$ product terms can be replaced by a single word of the form $1.00\cdots g_i 0 g_{i+1} 0\cdots$ with $g_i = m \cdot |\rho_i|$, $m \in \{-1, 0, 1\}$ (the proof is given in Appendix D). The digit g_i is inserted at the position with a weight of 2^{-2i-1}, for $i = n/4 + 1, \cdots, n/2$. The memory requirements are then determined by the partition sizes of the lower $n/4$ product terms ($i = i_m, \cdots, n/4$). The size of the memory is $2i \cdot 2^p$, where p is the number of bits in the address space, i is the iteration counter. There are two output digits of the table. However, the signs are known in advance, so only two output bits suffice.

Table II shows the relation between the required memory and the number of (partial factor) partitions as a function of the required precision and Fig. 3(c) shows the hardware organization. The memory requirements can be traded off with

TABLE II
MEMORY REQUIREMENTS OF THE PARTIAL TERMS OF THE SCALING FACTOR

n (bits)	partition I	memory size (bits)	partition II	memory size
16	(4)	512	(2,2)	256
24	(6)	4k	(3,3)	1k
32	(8)	16k	(4,4)	2k
48	(6,6)	16k	(4,4,4)	6k
64	(8,8)	64k	(5,5,6)	24k

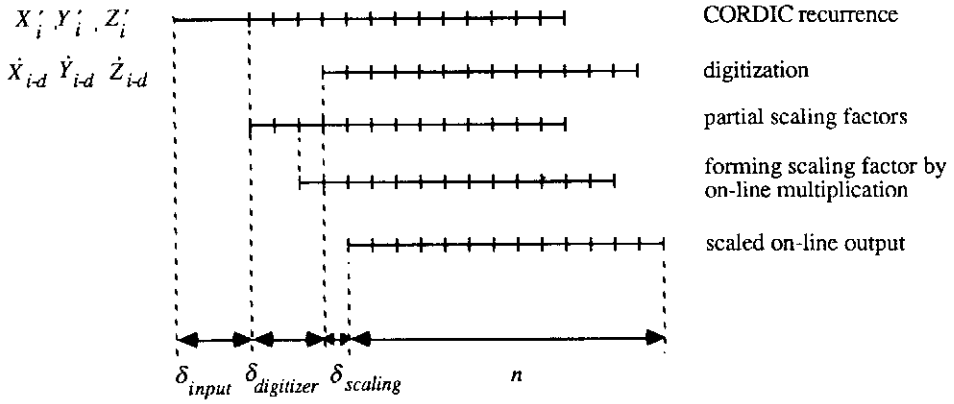

Fig. 4. Illustration of timing of on-line CORDIC implementation.

multipliers. To rescale one CORDIC output (X_i), 1, 2, or 3 on-line multipliers are required for 1, 2, or 3 table partitions. For the second CORDIC output (Y_i) another on-line multiplier is needed. Since the generation of the partial scaling factors can all start at the first iteration, the resulting on-line delay of the table lookup phase is $\delta = 0$. The total on-line delay of scaling factor is equal to the on-line delay of the on-line multiplier(s) for forming the partial scaling factors into one. The on-line delay of an on-line multiplier is 1 ([1], [2]). Therefore, for a partition of 2 or 3 partial tables, the on-line delay for forming the scaling factor is only 1 to 2. Since $\delta_{input} + \delta_{digitizer} > 4$, the only additional on-line delay to the on-line CORDIC algorithm is the on-line delay of the multiplication of the scaling factor with the digitized result (see Fig. 4), i.e., $\delta_{scaling} = 1$.

VII. IMPLEMENTATION ASPECTS OF THE ON-LINE CORDIC ALGORITHMS

The hardware modifications of an on-line CORDIC implementation as compared to the conventional CORDIC scheme are straightforward. Fig. 5 shows an overview of the hardware, whereas in Fig. 6 a more detailed design is given. The basic components in Fig. 5 are the normal CORDIC iterators (modules A and C). There are two (smaller) CORDIC-like iterators for the correction terms needed (modules B and E). The results of the iterations are digitized using modules C and D. The implementation of the X, Y iterator [Fig. 6(a)] contains a three-operand carry propagation free redundant adder (RA), due to the extra correction term. A signed-digit redundant adder (RA) has about the same area and time complexity as a carry save adder (CSA). The correction iterators [Fig. 6(b)] are normal CORDIC-like iterators. The scalings with 2^i

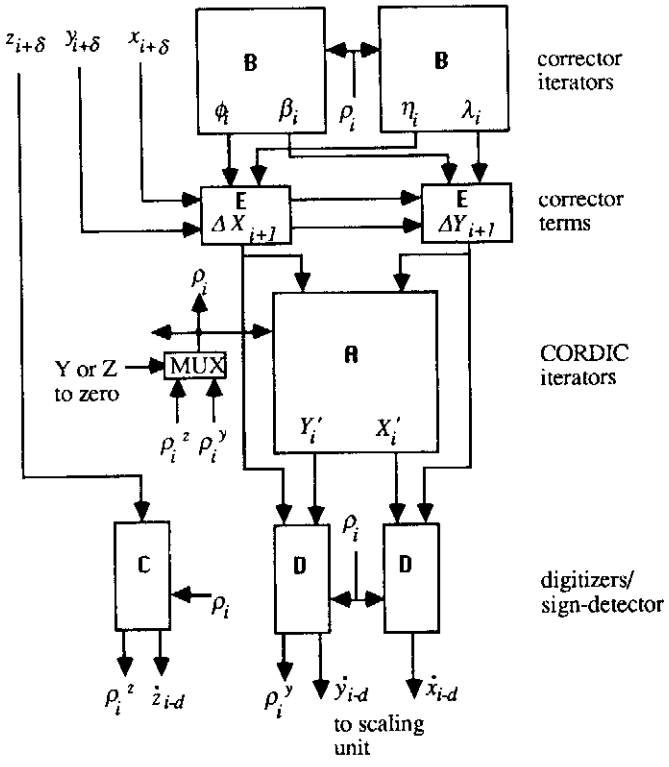

Fig. 5. Hardware overview.

as shown in Fig. 6 are fictive, they only indicate the relative position and do not affect the weight of the result.

A. Precision of the Various Variables in the Modules

In this section, the required precision of the various operands in the hardware modules is determined. This precision analysis is rather complex because of the interdependence

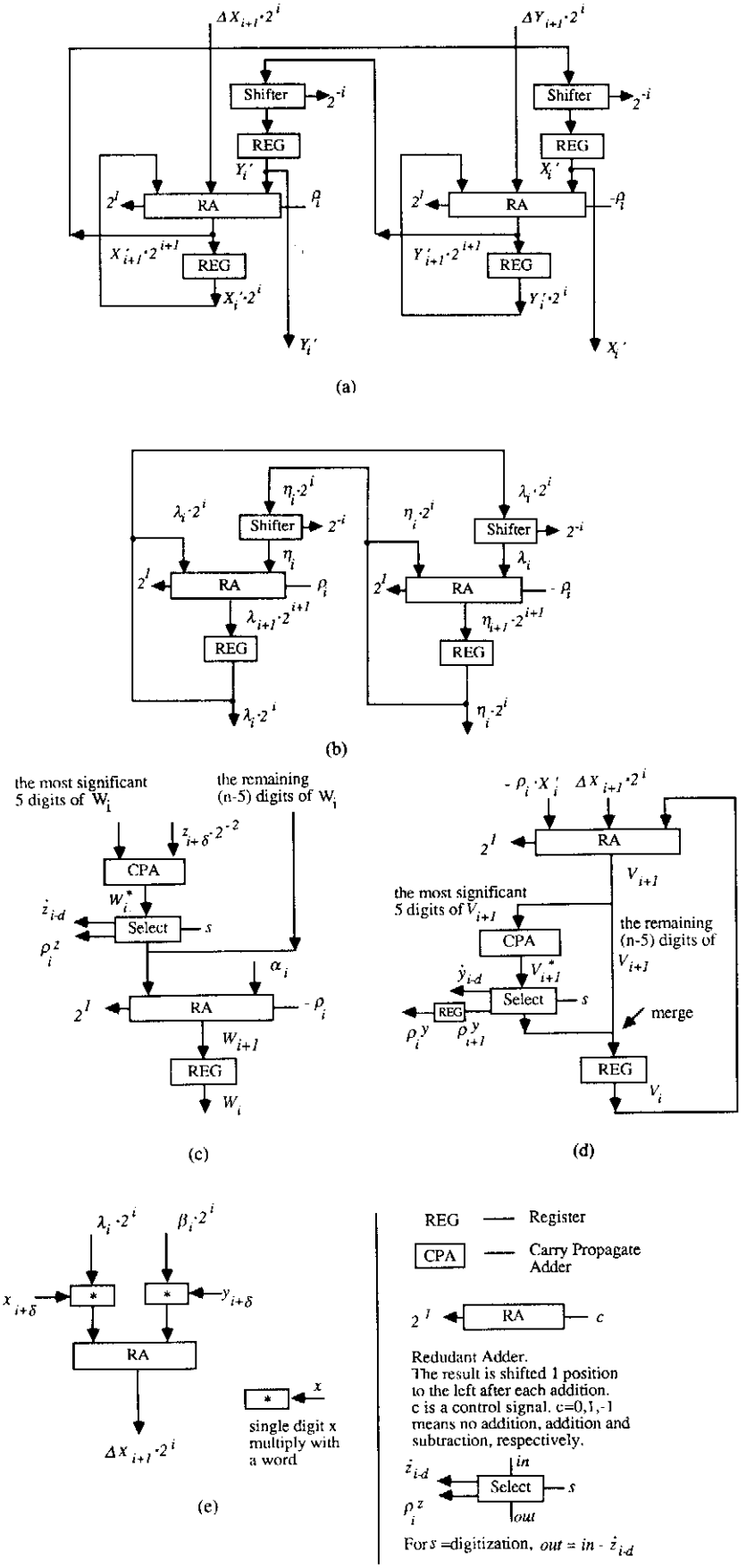

Fig. 6. Detailed hardware schemes. (a) A-module for X and Y recurrence. (b) B-module for λ, η and β, ϕ recurrence. (c) C-module for Z recurrence and digitization. (d) D-module for sign-detection and digitization of X and Y. (e) E-module for correction term.

relations of the operands. The analysis starts with the digitizer modules and works its way back to the corrector modules.

For an n digit mantissa, the results of the on-line CORDIC computation have an accuracy of 2^{-n}, if the total rounding error of the CORDIC iterations is less than 2^{-n}. Therefore, in order to obtain results with an accuracy of 2^{-n} the last digits (the least significant digit) x_n, y_n, and z_n must be selected from the residues R_n with an error less than $(1/2)\cdot 2^{-n}$ (due to rounding). In the following, a brief analysis will be given about the internal data representation. There are two sources of errors during the iteration: 1) the truncation error due the limited length of the RA's; 2) errors caused by the barrel shifters due to limited number of digits in the rows or due to the fact that the maximum shift range is less than the number of shifts required. In the discussion, we use the superscript E to indicate the exact value of a variable, i.e., R_n^E indicates the exact value of R_n, and $\epsilon(R_n) = |R_n - R_n^E|$ denotes the error in R_n.

1) The Digitization Module: We require that $\epsilon(R_n) = |R_n - R_n^E| < (1/2)\cdot 2^{-n}$ for an accuracy of 2^{-n} of the digitization results. The equation for R_{i+1} [(22)] is (unscaled here)

$$R_{i+1} = R_i - \rho_i \cdot X_i' \cdot 2^{-i} + \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta}) - 2^{-i+\varphi+d} \cdot y_{i-d}. \quad (29)$$

Suppose that the data paths (RA's and REG) for the digitizer consist of $(n + q_R)$ digits. We want to determine the value of q_R. Assume further that the barrel shifter for $X_i'\cdot 2^{-i}$ is $n \times (n + q_X)$ digits, then some digits of $X_i'\cdot 2^{-i}$ will be lost (shifted away) when the number of digits in X_i' grows larger than $(n + q_X - i)$. It can be easily shown that the number of digits in X_i' at iteration i is $\delta + i(i-1)/2$. An error with a maximum of $2^{-(n+q_X)}$ is caused if it holds that $\delta + i(i-1)/2 > n + q_X - i$, or equivalently for $i \geq \sqrt{(2n)}$ with the assumption $q_X > \delta$. The maximum error in R_{i+1} caused by the addition of $X_i'\cdot 2^{-i}$ is $\max\{2^{-(n+q_X)}, 2^{-(n+q_R)}\}$. Similarly, with an RA in module E of $(n+q_{\Delta Y})$ digits, it holds that $\epsilon(\Delta Y_{i+1}) = 0$ for $i < \sqrt{(2n)}$ and $\epsilon(\Delta Y_{i+1}) = |\Delta Y_{i+1} - \Delta Y_{i+1}^E| < 2^{-(n+q_{\Delta Y})}$ for $i \geq \sqrt{(2n)}$ (as will be shown later on). Thus, for the error in R_{i+1}, it holds

$$\epsilon(R_{i+1}) < \epsilon(R_i) + \max\{2^{-(n+q_X)}, 2^{-(n+q_R)}\} + \max\{2^{-(n+q_{\Delta Y})}, 2^{-(n+q_R)}\} \quad (30)$$

with $\epsilon(R_i) = 0$ for $i < \sqrt{(2n)}$. Thus,

$$\epsilon(R_{i+1}) < (i - \sqrt{(2n)}) \cdot (\max\{2^{-(n+q_X)}, 2^{-(n+q_R)}\} + \max\{2^{-(n+q_{\Delta Y})}, 2^{-(n+q_R)}\}). \quad (31)$$

Assume that $q = q_R = q_{\Delta Y} = q_X$, then the condition $\epsilon(R_i) < (1/2)\cdot 2^{-n}$ (for $i = 0, 1, \cdots, n$) can be satisfied if $q_R \geq \lceil \log(2(n - \sqrt{(2n)}))\rceil + 1$.

For the digitizer of Z_i' the only error is caused by the truncation of the angle α_i [(7)]. Therefore, $q_R \geq \lceil \log(n)\rceil + 1$ is sufficient for the digitizer of Z_i'.

2) The CORDIC Iterator for X_i' and Y_i': From the above consideration 1), we require that $\epsilon(X_i'\cdot 2^{-i}) \leq 2^{-(n+q)}$ and $\epsilon(Y_i'\cdot 2^{-i}) \leq 2^{-(n+q)}$ for $i = 1, 2, \cdots, n$. Suppose that the RA's and REG's in the iterator consist of $(n + a)$ digits, and the shifters are $n \times (n + q)$. From (4a), it holds

$$\epsilon(X_{i+1}') < \epsilon(X_i') + \max\{\epsilon(Y_i'\cdot 2^{-i}), 2^{-(n+a)}\} + \max\{\epsilon(\Delta X_{i+1}), 2^{-(n+a)}\}. \quad (32)$$

With the knowledge that $\epsilon(X_i') = 0$ for $i < \sqrt{(2n)}$, $\epsilon(Y_i'\cdot 2^{-i}) < 2^{-(n+q)}$ and $\epsilon(\Delta X_{i+1}) < 2^{-(n+q)}$, it follows

$$\epsilon(X_{i+1}') < (i - \sqrt{(2n)} + 1)\cdot 2^{-n+1} \cdot \max\{2^{-q}, 2^{-a}\}. \quad (33)$$

Since only the errors in the terms $X_i'\cdot 2^{-i}$ and $Y_i'\cdot 2^{-i}$ directly influence the digitization results and $\epsilon(Y_i'\cdot 2^{-i}) \leq \max\{\epsilon(Y_i')\cdot 2^{-i}, 2^{-(n+q)}\}$, the errors in $\epsilon(X_i')$ and $\epsilon(Y_i')$ may be larger than $2^{-(n+q)}$. We require $\epsilon(X_i') \leq 2^{-(n+q-i)}$ for $1 \leq i \leq n$ to ensure $\epsilon(X_i'\cdot 2^{-i}) \leq 2^{-(n+q)}$, i.e.,

$$(i - \sqrt{(2n)} + 1)\cdot 2^{-n+1} \cdot \max\{2^{-q}, 2^{-a}\} \leq 2^{-(n+q-i-1)}, \quad \text{for } i \geq \sqrt{(2n)}. \quad (34)$$

Since the term at the right-hand side of (34) increases faster than the term at the left-hand side, the condition is satisfied if (34) is valid for $i = \sqrt{(2n)}$. This leads to $a \geq q - \sqrt{(2n)}$, i.e., $a \geq \lceil \log(2(n - \sqrt{(2n)}))\rceil - \sqrt{(2n)} + 1$.

3) The Corrector Module: In the considerations 1) and 2), we require that $\epsilon(\Delta X_{i+1}) < 2^{-(n+q)}$ and $\epsilon(\Delta Y_{i+1}) < 2^{-(n+q)}$, for $1 \leq i \leq n$. Suppose the RA's and REG's in the corrector consist of $(n + c)$ digits, the value of c being determined in the following analysis.

Since $\Delta X_{i+1} = \lambda_i\cdot x_{i+\delta} + \beta_i\cdot y_{i+\delta}$ (see Appendix A), it holds $\epsilon(\Delta X_{i+1}) < (\epsilon(\lambda_i) + \epsilon(\beta_i))\cdot 2^{-(i+\delta)}$. Therefore, $\epsilon(\lambda_i) < (1/2)\cdot 2^{-(n+q-i-\delta)}$ and $\epsilon(\beta_i) < (1/2)\cdot 2^{-(n+q-i-\delta)}$ are sufficient conditions for $\epsilon(\Delta X_{i+1}) < 2^{-(n+q)}$. Similarly, $\epsilon(\phi_i) < (1/2)\cdot 2^{-(n+q-i-\delta)}$ and $\epsilon(\lambda_i) < (1/2)\cdot 2^{-(n+q-i-\delta)}$ are conditions for $\epsilon(\Delta Y_{i+1}) < 2^{-(n+q)}$. We want to determine the values of c, s, and t such that these conditions are satisfied. In the following, we consider the case for $\epsilon(\lambda_i)$, the other cases ($\epsilon(\beta_i)$, etc.) can be derived in the same way. From (A3), it follows that

$$\epsilon(\lambda_{i+1}) < \epsilon(\lambda_i) + \max\{\epsilon(\eta_i\cdot 2^{-i}), 2^{-(n+c)}\} \quad (35)$$

(for $\epsilon(\beta_i)$, $\epsilon(\eta_i)$, and $\epsilon(\phi_i)$ similar equations as (35) yield).

Assume that the barrel shifters in the corrector have s rows each of t digits ($s \times t$), i.e., the maximum shift range is 2^{-s}. Since the number of digits of (exact) λ_i and η_i at iteration i is $1 + i(i-1)/2$, the digits with a weight less than 2^{-t} will be lost when $i(i-1)/2 + i > t$, i.e., $i \geq \sqrt{(2t)}$. Thus,

$$\epsilon(\eta_i\cdot 2^{-i}) = \begin{cases} 0, & i < \sqrt{2t} \\ \max\{2^{-t}, \epsilon(\eta_i)\cdot 2^{-i}\}, & \sqrt{2t} \leq i \leq s \\ \max\{2^{-s}, \epsilon(\eta_i)\cdot 2^{-i}\}, & i > s \end{cases} \quad (36)$$

(assuming $\sqrt{(2t)} < s < t$). $\epsilon(\eta_i\cdot 2^{-i})$ denotes the error caused by the shifter, while $\epsilon(\eta_i)$ denotes the error in η_i. Therefore,

for $\epsilon(\lambda_i) < (1/2)\cdot 2^{-(n+q-i-\delta)}$ it is sufficient that

$$(i - \sqrt{2(n+c)} + 1) \cdot 2^{-(n+c)} \leq 2^{-(n+q-i-\delta+1)}$$

and

$$\sum_{k=\sqrt{2i}}^{i} \epsilon(\eta_k \cdot 2^{-k}) \leq 2^{-(n+q-i-\delta+1)}. \tag{37}$$

The first equation in (37) leads to the condition $c \geq \lceil \log(2(n - \sqrt{(2n)}))\rceil - \sqrt{(2(n+c))}$ ($\delta \geq 2$). The second equation in (37) leads to the conditions $t \geq n + \lceil \log(2(n - \sqrt{(2n)}))\rceil - \sqrt{(2t)}$ and $s \geq (n + \lceil \log(2(n - \sqrt{(2n)}))\rceil)/2$.

B. Time and Area Complexity

From Figs. 5 and 6 it can be observed that the critical path (i.e., the longest circuit flow propagation time) of the on-line CORDIC implementation is either one shifter, one redundant adder, and one register, or one 5-digit carry propagate adder, one selection, one redundant adder, and one register. Since the small 5-digit carry propagate adder can be implemented using carry look-ahead logic and the selection function in module C and D has only a delay of 2 to 3 gates, the barrel shifter in the CORDIC iterator (module A) is the dominant factor of propagation delay for $n \geq 16$. Therefore, for $n \geq 16$ the critical path is 1 $n \times (n+q)$ barrel shifter, one redundant adder, and one register.

Like in the case of conventional CORDIC implementations [18], the two barrel shifters in the CORDIC iterator (module A) consume a major part of total area. The four barrel shifters in the corrector (module B) are approximately $(n/2) \times n$, so these four barrel shifters consume roughly the same area as two $n \times n$ barrel shifters in the CORDIC iterator. For the scaling factor a ROM is needed plus a number of on-line multipliers. ROM's are smaller than shifters. For 24-bit, the ROM size is approximately equal to that of the angles table. For 48-bit, about six times more space is needed. On the other hand, there is no need for a fast carry propagating adder. By using the same technology as described in [22], we have estimated that the area of the on-line CORDIC implementation is approximately twice that of a conventional CORDIC implementation.

VIII. Application to Givens' Rotation and SVD and Discussion

From the previous sections it follows that all applicable CORDIC operations can be made on-line. The total latency per function is given by

$$T_{\text{tot}} = n + \delta_{\text{input}} + \delta_{\text{digitizer}} + \delta_{\text{scaling}}.$$

The results $\delta_{\text{total}} = \delta_{\text{input}} + \delta_{\text{digitizer}} + \delta_{\text{scaling}}$ have been summarized in Table III. In this table, the columns denote the on-line inputs. So XYZ in column 1 denotes that X, Y, and Z are input on-line, while XY in column 2 denotes that only X and Y are input on-line. (Notice that $\delta_{\text{scaling}} = 0$ for Z since no rescaling is required.)

The proposed on-line CORDIC algorithm attains speed improvement by introducing redundancy in the calculation circuit. Another advantage is that on-line algorithms reduce the I/O requirement and the total latency of chained processing. For example, in applications such as Givens' rotation for matrix triangularization and the transformation for singular value decomposition (SVD), the angle must be computed first and then rotations are performed. In the following, we compare for Givens' rotation and SVD the implementation using the proposed on-line CORDIC processor with some other known implementations.

TABLE III
On-Line Delays of Several CORDIC Functions

function	on-line arguments		
	X,Y,Z	X,Y	Z
	on-line delays		
$x.sin\phi+y.cos\phi$	6	4	6
$x.sinh\phi+y.cosh\phi$	6	4	5
$tan^{-1}(y/x)$	6	6	2
$(x^2+y^2)^{1/2}$	8	8	4
$ln(w), (x=w+1, y=w-1)$	6	6	2

Application to Givens' Rotation: Givens' rotations occur as basic operations in the triangularization of a matrix [19]. In [20], Ahmed, Delosme, and Morf describe an implementation for Givens' rotation using conventional CORDIC algorithms. They used two types of processors: the angle processor and the rotation processor. Using this scheme the total delay of the Givens' rotation is between $2.25n$ and $3n$ CORDIC steps. The step time (clock cycle) of their scheme is mainly determined by the carry-propagate adders and the shifters.

Ercegovac and Lang [12] describe a CORDIC implementation of Givens' rotation using redundant adders and overlapping of the angle- and rotation calculation. The total latency of this scheme is $3n+3$ CORDIC steps. They have estimated that their scheme will be approximately 4.5 times faster than the scheme in [20], because of a 6 times faster clock.

To compute the Givens' rotation, two on-line CORDIC processors as in Fig. 5 can be used: one for the computation of the angle and the other for the rotation. Then the total latency is $(n + 13)$ CORDIC steps[1] (of course, the same strategy as described in [12] by using decomposed sequence of the angle, $\{\rho_0, \rho_1, \cdots, \rho_n\}$, for rotation can be applied in a dedicated Givens' rotation implementation. This will reduce the on-line delay and some hardware). We notice that the computation latency in terms of number of CORDIC steps of our implementation and the implementations described in [12] and [20] are not fully comparable. The latter two implementations assume that the full word of the input operands is available before the computation starts, while our on-line implementation assumes the input operands are fed into the CORDIC processor one digit/bit at a time during the computation. To make a rough comparison, we add n CORDIC steps to the latency time of the implementation described in [12]. The total latency time then becomes $4n+3$ CORDIC steps for Givens' rotation. The step time in our implementation is determined by a shifter and a redundant adder, so it is equivalent to 2

[1] Since the first digit of the angle computed by the first on-line CORDIC processor has a weight of 2^1 (due to the digitization), instead of 2^0, the on-line delay of the second processor is to be increased by 1.

CORDIC steps of the scheme in [12] (their scheme for angle computation has the same critical path as ours, to which a step time has been counted as two basic time steps). The speed ratio of the implementation in [12] and the proposed on-line CORDIC implementation is therefore $(4n+3)/(2n+25)$. For $n = 24$ (32-bit floating point) and $n = 48$ (64-bit floating point), the application of the proposed on-line CORDIC implementation for Givens' rotation is approximately 1.3 times and 1.6 times faster, respectively. Therefore, the proposed implementation is approximately 6.5 times faster than the conventional CORDIC implementation in [20].

Application to SVD Computation: Singular value decomposition (SVD) is an important algorithm in many matrix computations [11], [19], [21]. In [21], Brent, Luk, and Van Loan describe a systolic square array of simple 2×2 processors to compute the SVD of a large matrix. The primitive operation in SVD is the diagonalization of a 2×2 matrix. Using the proposed on-line CORDIC processor to compute such a diagonalization, the computation proceeds as follows: 1) compute the two angles using two concurrent CORDIC processors: $\theta_s = \tan^{-1}[(c+b)/(d-a)]$ and $\theta_d = \tan^{-1}[(c-b)/(d+a)]$; 2) compute $\theta_l = (\theta_s - \theta_d)/2$ and $\theta_r = (\theta_s + \theta_d)/2$; 3) perform the operations of the type $x\cdot\cos(\theta) + y\cdot\sin(\theta)$ and $x\cdot\cos(\theta) - y\cdot\sin(\theta)$, for θ_l and θ_r, using two pairs of chained on-line CORDIC processors. Fig. 7 shows a configuration of the implementation using the proposed on-line CORDIC processors. The total latency of such a SVD computation is $n+22$ CORDIC steps (including the on-line delay of the operations $c+b$ and $(\theta_s - \theta_d)/2$, etc.).

Cavallaro and Luk describe in [11] a CORDIC implementation for the diagonalization of a 2×2 matrix in SVD applications using two CORDIC modules. The total latency time of their scheme is $3.25n$ CORDIC steps, where the step time is determined by the ripple carry-propagate addition of n bits. Ercegovac and Lang [12] describe an implementation using redundant addition and internal on-line arithmetic to maximize the overlap between successive operations. These modifications contribute to a speed up of about 4 compared to the scheme in [11]. The total latency time of their scheme is $5n + 10$ CORDIC steps, but the step time (clock cycle) is about 6 times faster than the scheme in [11]. The speed ratio of the scheme using the proposed on-line CORDIC processor (Fig. 7) and the scheme in [12] is $(6n+10)/(2n+44)$ (for the comparison, n input steps have been added to the latency time of the scheme in [12]). Thus, the scheme using the proposed on-line CORDIC processors is 1.7 times and 2.1 times faster for $n = 24$ and $n = 48$, respectively. Based on the estimation given in [12], we conclude that the scheme in Fig. 7 is 6.8 to 8.4 times faster than the scheme in [11] using conventional CORDIC implementation. More speedup in SVD application is obtained than in the computation of Givens' rotation by using the proposed on-line CORDIC method. This demonstrates the effectiveness of on-line chaining in computing complicated functions, since the total latency time of the computation only increases by a small constant on-line delay time for each successive calculation. It is expected that higher speedups can be obtained when the on-line CORDIC processors are organized properly into a systolic structure (e.g., the systolic scheme in [21] with some modifications) for the SVD of a large matrix. Further investigations of this important application have yet to be done.

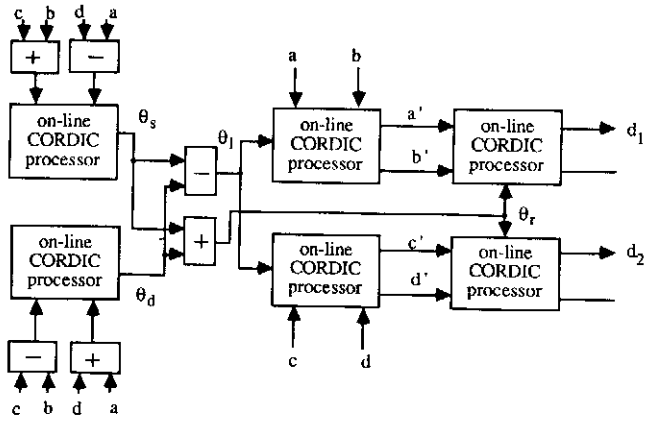

Fig. 7. An on-line configuration for SVD computation using CORDIC processors.

The hardware complexity of the scheme described in [11] is essentially equal to two conventional CORDIC modules, while the scheme in Fig. 7 requires six on-line CORDIC processors. Since the hardware complexity of one on-line CORDIC processor is approximately two conventional CORDIC modules, the hardware complexity of the scheme using the proposed on-line CORDIC processors is about 6 times of that of the scheme described in [11]. The choice between these two schemes will depend on the practical situation, since there is a tradeoff between the time and the hardware complexity. The hardware complexity of the implementation in [12] is unknown to the authors, further evaluations are required. The hardware complexity of Fig. 7 can also be reduced by performing the rotation of the angle θ_r using the first pair of CORDIC processors. Now the feedback of a' and b' for the rotation of θ_r must wait (using an external register) until the angle computation has been completed at step $(n+9)$. The total latency will be increased to $2n+15$ CORDIC steps (if the first $\delta = 4$ digits of a' and b' are accumulated inside the processor using a register, it can be reduced to $2n+11$ CORDIC steps). The speedup as compared to the scheme in [11] is decreased to 5; however, the hardware complexity has also been reduced to about 4 times that of the scheme described in [11]. One may even think of using only two on-line CORDIC processors; in that case an even higher speedup and hardware complexity ratio can be obtained. However, then one cannot benefit from the advantage of on-line chaining to achieve higher speed.

Appendix A

Calculation of the Correction Terms

In this Appendix a way is shown to calculate the correction terms $\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})$ of (4). $\Delta X_k(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_k(x_{i+\delta}, y_{i+\delta})$ are the errors in X'_k and Y'_k with respect to X_k and Y_k. They represent the errors due to the absence of the digits $x_{i+\delta}$ and $y_{i+\delta}$ from the *first* to the $(k-1)$th iterations $(k \le i+1)$ in the CORDIC recurrence iterations [(4)]. The CORDIC recurrence equations are linear equations with respect to the digits $x_{i+\delta}$ and $y_{i+\delta}$

$(i = 0, 1, \cdots, n-\delta)$; therefore, the errors at iteration k due to the absence of $x_{i+\delta}$'s and $y_{i+\delta}$'s can be decomposed into independent terms $\Delta X_k(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_k(x_{i+\delta}, y_{i+\delta})$ for $i = 0, 1, \cdots, n-\delta$. Since the digits $x_{i+\delta}$ and $y_{i+\delta}$ become known at the ith iteration, the errors made by the absence of $x_{i+\delta}$ and $y_{i+\delta}$ are corrected by adding $\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})$ to X'_{i+1} and Y'_{i+1}, respectively.

In the following, we denote $\Delta X_k(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_k(x_{i+\delta}, y_{i+\delta})$ as ΔX_k and ΔY_k, respectively. The error in X_{k+1} and Y_{k+1} (assuming that only $x_{i+\delta}$ and $y_{i+\delta}$ are absent for fixed i) can be written as

$$\begin{aligned} X_{k+1} &= X'_k + \Delta X_k + m\rho_k(Y'_k + \Delta Y_k)\cdot 2^{-k} \\ &= X'_k + m\rho_k Y'_k \cdot 2^{-k} + \Delta X_k + m\rho_k \Delta Y_k \cdot 2^{-k} \\ &= X'_{k+1} + \Delta X_{k+1} \\ Y_{k+1} &= Y'_k + \Delta Y_k - \rho_k(X'_k + \Delta X_k)\cdot 2^{-k} \\ &= Y'_k - \rho_k X'_k \cdot 2^{-k} + \Delta Y_k - \rho_k \Delta X_k \cdot 2^{-k} \\ &= Y'_{k+1} + \Delta Y_{k+1}. \end{aligned} \tag{A1}$$

The initial value of X_0 and Y_0 in the conventional CORDIC recurrences are the entire input operands, so in on-line CORDIC recurrences the initial errors are $x_\delta x_{1+\delta} \cdots x_n$ and $y_\delta y_{1+\delta} \cdots y_n$. The errors with respect to the digits $x_{i+\delta}$ and $y_{i+\delta}$ are $\Delta X_0(x_{i+\delta}, y_{i+\delta}) = x_{i+\delta}$ and $\Delta Y_0(x_{i+\delta}, y_{i+\delta}) = y_{i+\delta}$.

The term ΔX_k (with respect to $x_{i+\delta}$ and $y_{i+\delta}$) consists of two parts: one part depends on the digit $x_{i+\delta}$ and the other part on $y_{i+\delta}$. The same holds for ΔY_k. Denote $\Delta X_k = \lambda_k(x_{i+\delta}) + \beta_k(y_{i+\delta})$ and $\Delta Y_k = \eta_k(x_{i+\delta}) + \phi_k(y_{i+\delta})$, where $\lambda_k(x_{i+\delta})$ and $\eta_k(x_{i+\delta})$ are the terms dependent on $x_{i+\delta}$ and $\beta_k(y_{i+\delta})$ and $\phi_k(y_{i+\delta})$ are the terms dependent on $y_{i+\delta}$. By observing that $\lambda_0(x_{i+\delta}) = \lambda_0 \cdot x_{i+\delta}$, $\beta_0(y_{i+\delta}) = \beta_0 \cdot y_{i+\delta}$, $\eta_0(x_{i+\delta}) = \eta_0 \cdot x_{i+\delta}$, and $\phi_0(y_{i+\delta}) = \phi_0 \cdot y_{i+\delta}$, with $\lambda_0 = 1$, $\beta_0 = 0$, $\eta_0 = 0$, and $\phi_1 = 1$, we can separate the values of $x_{k+\delta}$ and $y_{k+\delta}$ from the terms λ_k, β_k, η_k, and ϕ_k. The following relations hold

$$\begin{aligned} \lambda_{k+1} &= \lambda_k + m\rho_k\eta_k \cdot 2^{-k} && \text{with } \lambda_0 = 1 \\ \beta_{k+1} &= \beta_k + m\rho_k\phi_k \cdot 2^{-k} && \text{with } \beta_0 = 0 \\ \eta_{k+1} &= \eta_k - \rho_k\lambda_k \cdot 2^{-k} && \text{with } \eta_0 = 0 \\ \phi_{k+1} &= \phi_k - \rho_k\beta_k \cdot 2^{-k} && \text{with } \phi_0 = 1. \end{aligned} \tag{A2}$$

From (A2), we can calculate λ_k, β_k, η_k, and ϕ_k $(k = 0, 1, \cdots, i)$ independently from $x_{i+\delta}$ and $y_{i+\delta}$. The correction terms at iteration i can then be evaluated according to

$$\begin{aligned} \Delta X_{i+1}(x_{i+\delta}, y_{i+\delta}) &= \lambda_i \cdot x_{i+\delta} + \beta_i \cdot y_{i+\delta} \\ \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta}) &= \eta_i \cdot x_{i+\delta} + \phi_i \cdot y_{i+\delta}. \end{aligned} \tag{A3}$$

The equations (A2) are two sets CORDIC-like equations.

Note: In the above consideration, we assume that the index k starts from 0, which is the case of the circular CORDIC $(m = 1)$. For the hyperbolic case $(m = -1)$, the index k begins at 1. However, the results remain further unchanged.

Appendix B

Estimation of the Upper Bounds on ΔX_{i+1} and ΔY_{i+1}

The following relations hold for the correction terms $\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})$ and $\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})$,

$$\begin{aligned} \Delta X_{i+1}(x_{i+\delta}, y_{i+\delta}) &= \Delta X_i(x_{i+\delta}, y_{i+\delta}) \\ &\quad + m\rho_i \cdot \Delta Y_i(x_{i+\delta}, y_{i+\delta}) \cdot 2^{-i} \\ \Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta}) &= \Delta Y_i(x_{i+\delta}, y_{i+\delta}) \\ &\quad - \rho_i \cdot \Delta X_i(x_{i+\delta}, y_{i+\delta}) \cdot 2^{-i}. \end{aligned} \tag{B1}$$

For $m = 1$, with $\Delta X_0(x_{i+\delta}, y_{i+\delta}) = x_{i+\delta}$ and $\Delta Y_0(x_{i+\delta}, y_{i+\delta}) = y_{i+\delta}$ ($x_{i+\delta}$ and $y_{i+\delta}$ are digits with a weight of $2^{-(i+\delta+1)}$, since i starts from 0), it can be verified (e.g., simply iterating the recurrence (B1) with a computer) that

$$\begin{aligned} |\Delta X_4(x_{i+\delta}, y_{i+\delta})| &\le 2.32 \cdot 2^{-(i+\delta+1)} \\ |\Delta Y_4(x_{i+\delta}, y_{i+\delta})| &\le 2.32 \cdot 2^{-(i+\delta+1)}. \end{aligned} \tag{B2}$$

Therefore, from (B1) and (B2) it holds

$$\begin{aligned} |\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})| &\le 2.32 \cdot 2^{-(i+\delta+1)} \cdot \prod_{j=4}^{i}(1 + 2^{-j}) \\ |\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})| &\le 2.32 \cdot 2^{-(i+\delta+1)} \cdot \prod_{j=4}^{i}(1 + 2^{-j}). \end{aligned} \tag{B3}$$

Since

$$\begin{aligned} \prod_{j=4}^{i}(1 + 2^{-j}) &< \prod_{j=4}^{\infty}(1 + 2^{-j}) \\ &< \prod_{j=4}^{9}\left(\frac{1 + 2^{-j}}{1 + (1/j^2\pi^2)}\right) \cdot \sinh(1) \\ &\quad \cdot \frac{1}{\prod_{j=1}^{3}(1 + (1/j^2\pi^2))} \le 1.14 \end{aligned}$$

$$\left(\sinh(x) = x \cdot \prod_{j=1}^{\infty}(1 + x^2/(j^2\pi^2)), \quad \text{and for } j \ge 10,\right.$$

$$\left.(1 + 2^{-j}) < (1 + 1/(j^2\pi^2))\right)$$

it follows

$$|\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})| \le 2.65 \cdot 2^{-(i+\delta+1)}$$

$$|\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})| \le 2.65 \cdot 2^{-(i+\delta+1)}. \quad \text{(B4)}$$

For $m = -1$, it can be verified that $|\Delta X_4(x_{i+\delta}, y_{i+\delta})| \le 2.11 \cdot 2^{-(i+\delta)}$ and $|\Delta Y_4(x_{i+\delta}, y_{i+\delta})| \le 2.11 \cdot 2^{-(i+\delta)}$ ($x_{i+\delta}$ and $y_{i+\delta}$ have the weight of $2^{-(i+\delta)}$, since i starts from 1) and

$$|\Delta X_{i+1}(x_{i+\delta}, y_{i+\delta})| \le 2.41 \cdot 2^{-(i+\delta)}$$

$$|\Delta Y_{i+1}(x_{i+\delta}, y_{i+\delta})| \le 2.41 \cdot 2^{-(i+\delta)}. \quad \text{(B5)}$$

Similar analysis can be made for

$$\sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)].$$

The procedure is to calculate the above formula for several small values of i and then use (B4) and (B4) to estimate the upper bound. The following upper bounds can be obtained:

$$\left| \sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)] \right| \le 3 \cdot 2^{-(i+\delta)}$$

for $m = 1$

$$\left| \sum_{k=i+\delta}^{n} [\Delta X_i(x_k, y_k) \cdot 2^{-i} + \Delta Y_{i+1}(x_k, y_k)] \right| \le 5 \cdot 2^{-(i+\delta)}$$

for $m = -1$.

Appendix C

On-Line Property of the Scaling Factor K^{-1}

In this Appendix, the on-line delay of the scaling factor K^{-1} is determined:

$$K^{-1} = \sum_{i=i_0}^{k} (1 + m \cdot \phi_i^2)^{-1/2} \quad \text{(C1)}$$

where $\phi_i = \rho_i \cdot 2^{-i}$. In order to determine the on-line delay of (C1), the on-line condition stated in (2) will be evaluated. The terms ϕ_i are considered as the input variables X of the table. The absolute value of the differential quotient of $(1 + \phi_i^2)^{-1/2}$ is

$$|\partial(1 + \phi_i^2)^{-1/2} / \partial \phi_i| = |-\phi_i (1 + \phi_i^2)^{-3/2}|. \quad \text{(C2)}$$

It follows that every element g_i of the vector $|\nabla K^{-1}(\phi_i)|$ is of the form

$$g_i = \left| -\phi_i (1 + m \cdot \phi_i^2)^{-1} \cdot \prod_{i=i_0}^{k} (1 + m \cdot \phi_i^2)^{-1/2} \right|. \quad \text{(C3)}$$

A. Circular Case ($m = 1$)

For the circular case it holds that

$$\left| \prod_{i=i_0}^{k} (1 + \phi_i^2)^{-1/2} \right| \le 1 \quad \text{(C4)}$$

therefore,

$$g_i \le 2^{-i} (1 + 2^{-2i})^{-1}. \quad \text{(C5)}$$

It holds that $0,607 \cdots \le K^{-1} \le 1$, yielding $\varphi = 0$ for the position of the most significant digit of K^{-1}. Therefore, from (3) the following inequality must be satisfied:

$$\sum_{i=i_0}^{k} g_i \cdot 2^{-\delta_i} \le 1 \quad \text{(C6)}$$

where δ_i is the on-line delay of the variable ϕ_i. It holds that $\phi_i = \rho_i \cdot 2^{-i}$. This means that this term is known at the ith iteration; before that all digits are zero. Thus, we can define the on-line delay of ϕ_i relative to ϕ_{i0} as $\delta_i = \delta_0 + i$. It also holds that $(1 + 2^{-2i})^{-1} < 1$, transforming (C6)

$$\sum_{i=i_0}^{k} 2^{-\delta_0 - 2 \cdot i} < 2^{-\delta_0 - 2 \cdot i_0 + 1} \le 1. \quad \text{(C7)}$$

Equation (C7) is satisfied if $\delta_0 = 1 - 2i_0$. Therefore, if ϕ_{i0} is known, $2i_0 - 1$ digits of the result can be generated in advance. For the case $i_0 = 0$ a better bound can be achieved, yielding with (C6) $\delta_0 = 0$.

B. Hyperbolic Case ($m = -1$)

For the hyperbolic case it holds that

$$\prod_{i=i_0}^{k} (1 - 2^{-2i})^{-1/2} < \prod_{i=i_0}^{k} (1 + 2^{-2i}) < 1 + 2^{-2i_0} + 2^{-2i_0 - 1} + \cdots + 2^{-2i_0 - k} < 1 + 2^{-2i_0 + 1} \quad \text{(C8)}$$

with $i_0 \ge 1$. From this, it follows that

$$g_i < 2^{-i} (1 - 2^{-2i})^{-1} \cdot (1 + 2^{-2i_0 + 1}) \le 2^{-i+1}. \quad \text{(C9)}$$

For the hyperbolic case it holds that $1 \le K^{-1} \le 1,207 \cdots$. Therefore, also in this case we can take $\varphi = 0$ for the position of the most significant digit of K^{-1}. From (C9) and (C6) it follows

$$\sum_{i=i_0}^{k} 2^{-\delta_0 - 2i + 1} < 2^{-\delta_0 - 2i + 2} \le 1. \quad \text{(C10)}$$

Equation (C10) is satisfied if $\delta_0 = 2 - 2i_0$. Therefore, $2i_0 - 2$ digits of the result can be generated in advance prior ϕ_{i0} is known. For the case $i_0 = 1$, the bound yields with (C6) $\delta_0 = 0$.

APPENDIX D

ACCURACY ISSUES CONCERNING THE GENERATION OF THE SCALING FACTOR

In this Appendix, it will be shown that for an accuracy of 2^{-n} in the scaled results, only $n/2$ product terms of (5) are required. A simple method is described to generate the higher half ($n/4$) of the $n/2$ product terms. Using this simple method, the memory size of the lookup table for the on-line generation of the scaling factor can be reduced to a square root of the original size.

Lemma: Let

$$P_m(k, i_0) = \prod_{i=i_0}^{k} (1 + m \cdot |\rho_i| \cdot 2^{-2i})^{-1/2},$$

$$S_m(k, i_0) = 1 - m \cdot \left(\sum_{i=i_0}^{k} |\rho_i| \cdot 2^{-2i-1} \right), \quad \text{and}$$

$$D(k, i_0) = 2^{-4i_0} \cdot \sum_{i=i_0}^{k} |\rho_i| \cdot 2^{-2(i-i_0)},$$

with $m \in \{-1, 1\}$ and $r_i| \in \{0, 1\}$.

It holds

$$|P_1(k, i_0) - S_1(k, i_0)| \le (3/8) \cdot D(k, i_0), \quad \text{for } 0 \le i_0 \le k \quad \text{(D1.a)}$$

$$|P_{-1}(k, i_0) - S_{-1}(k, i_0)| \le D(k, i_0), \quad \text{for } 1 \le i_0 \le k. \quad \text{(D1.b)}$$

$P_1(k, i_0)$ and $P_{-1}(k, i_0)$ are (partial) scaling factors for $m = 1$ and $m = -1$, respectively. $S_m(k, i_0)$ is the approximation to $P_m(k, i_0)$ and $D(k, i_0)$ is the error bound of the approximation.

Proof: First, we consider the case of $m = 1$, i.e., (D1.a). It can be shown that it holds $1 - x/2 \le (1 + x)^{-1/2} \le 1 - x/2 + (3/8) \cdot x^2$ for $0 \le x \le 1$. Using this fact it follows that

$$1 - |\rho_i| \cdot 2^{-2i-1} \le (1 + |\rho_i| \cdot 2^{-2i})^{-1/2} \le 1 - |\rho_i| \cdot 2^{-2i-1} + (3/8) \cdot |\rho_i| \cdot 2^{-4i}. \quad \text{(D2)}$$

Equation (D2) shows that (D1.a) is true for $k = i_0$. Now, assume that (D1.a) is true for $k = j$ ($j \ge i_0$), we want to show that (D1.a) is true for $k = j + 1$. Under the assumption and with the fact that $P_1(k, j) \le 1$, it holds

$$\begin{aligned} P_1(j+1, i_0) &\ge P_1(j, i_0) \cdot (1 - |\rho_{j+1}| \cdot 2^{-2(j+1)-1}) \\ &\ge S_1(j, i_0) - (3/8) \cdot D(j, i_0) - |\rho_{j+1}| \cdot 2^{-2(j+1)-1} \\ &\ge S_1(j+1, i_0) - (3/8) \cdot D(j+1, i_0) \end{aligned} \quad \text{(D3)}$$

and

$$\begin{aligned} P_1(j+1, i_0) &\le P_1(j, i_0) \cdot (1 - |\rho_{j+1}| \cdot 2^{-2(j+1)-1} + (3/8) \cdot |\rho_{j+1}| \cdot 2^{-4(j+1)}) \\ &\le S_1(j, i_0) + (3/8) \cdot D(j, i_0) + |\rho_{j+1}| \cdot 2^{-2(j+1)-1} + (3/8) \cdot |\rho_{j+1}| \cdot 2^{-4(j+1)} \\ &\le S_1(j+1, i_0) + (3/8) \cdot D(j+1, i_0). \end{aligned} \quad \text{(D4)}$$

From (D3) and (D4), it is shown that (D1.a) is true for $k = j + 1$. Therefore, by induction it is shown that (D1.a) is true for all $k \ge i_0 \ge 0$.

For $m = -1$, the proof of (D1.b) can be done in the same way as for (D1.a). It can be shown that it holds $1 + x/2 \le (1 - x)^{-1/2} \le 1 + x/2 + x^2$ for $0 \le x \le 1/4$. So, $1 + |\rho_i| \cdot 2^{-2i-1} \le (1 - |\rho_i| \cdot 2^{-2i})^{-1/2} \le 1 + |\rho_i| \cdot 2^{-2i-1} + |\rho_i| \cdot 2^{-4i}$ for $i \ge 1$. Similar to the case for $m = 1$, (D1.b) can be proven by induction.

From the lemma, it follows that $P_1(n-1, n/2) \ge S_1(n-1, n/2) - (3/8) \cdot D(n-1, n/2) > 1 - 2^{-n-1}$ and $P_1(n-1, n/2) \le 1$, therefore, $|P_1(n-1, n/2) - 1| < 2^{-n-1}$. Similarly, it can be shown that $|P_{-1}(n-1, n/2) - 1| < 2^{-n-2}$. This means that for an accuracy of 2^{-n} in the scaled results, only the lower $n/2$ product terms in (5) are required.

Furthermore, from the lemma it follows that $|P_1(n/2 - 1, n/4) - S_1(n/2 - 1, n/4)| \le (3/8) \cdot D(n/2 - 1, n/4) < 2^{-n-1}$ and $|P_{-1}(n/2, n/4 + 1) - S_{-1}(n/2, n/4 + 1)| \le D(n/2, n/4 + 1) < 2^{-n-3}$. This means that the $n/4$ product terms $P_1(n/2 - 1, n/4)$ and $P_{-1}(n/2, n/4 + 1)$ can be replaced by $S_1(n/2 - 1, n/4)$ and $S_{-1}(n/2, n/4 + 1)$, respectively. Therefore, the scaling factor can be computed by multiplying the word of the form $1.000 \cdots g_i 0 g_{i+1} 0 \cdots$ (with $g_i = m \cdot |\rho_i|$, $m \in \{-1, 0, 1\}$) with the number $P_1(n/4 - 1, 0)$ (or $P_{-1}(n/4, 1)$) generated from the lookup table.

REFERENCES

[1] M. D. Ercegovac, "An online arithmetic: An overview," *SPIE*, vol. 495, Real Time Signal Processing VII, pp. 86–93, 1984.

[2] K. D. Trivedi and M. D. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. C-27, no. 7, pp. 681–687, July 1977.

[3] P. K.-G. Tu and M. D. Ercegovac, "A radix-4 on-line division algorithm," in *Proc. 8th Symp. Comput. Arithmetic*, Como, Italy, 1987, pp. 181–187.

[4] K. D. Trivedi and J. G. Rusnak, "High radix on-line division," in *Proc. 4th Symp. Comput. Arithmetic*, 1978, pp. 164–174.

[5] V. G. Oklobdzija and M. D. Ercegovac, "An on-line square root algorithm," *IEEE Trans. Comput.*, vol. C-31, no. 1, pp. 70–75, Jan. 1982.

[6] M. J. Irwin, "A pipelined processing unit for on-line division," in *Proc. 5th Symp. Comput. Arithmetic*, 1978, pp. 24–30.

[7] R. M. Owens, "Compound algorithms for digit online arithmetic," in *Proc. 5th Symp. Comput. Arithmetic*, 1981, pp. 64–71.

[8] R. M. Owens, "Digit online algorithms for pipelined architectures," Ph.D. dissertation, Dep. Comput. Sci., The Pennsylvania State Uiv., 1980.

[9] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sept. 1959.

[10] J. S. Walther, "A unified algorithm for elementary functions," in *Proc. Spring Joint Comput. Conf.*, 1971, pp. 379–385.

[11] 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.

[12] M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: Ap-

plication to matrix triangularization and SVD," UCLA Dep. Comput. Sci., Tech. Rep., CSD-870046, Sept. 1987.
[13] H. J. Sips and H. X. Lin, "A new model for on-line arithmetic with an application to the reciprocal calculation," *J. Parallel Distributed Comput.*, pp. 218–230, 1990.
[14] J. Duprat, Y. Herreros, and J. M. Muller, "Some results about on-line computation of functions," in *Proc. 9th Symp. Comput. Arithmetic*, Santa Monica, CA, Sept. 1989, pp. 112–118.
[15] H. M. Ahmed, "Signal processing algorithms and architectures," Ph.D. dissertation, Dep. Elec. Eng., Stanford Univ., 1982.
[16] J. C. Bu, E. F. A. Deprettre, and F. de Lange, "On the optimization of pipelined silicon CORDIC algorithm," in *Proc. EUSIPCO-86, Signal Processing III: Theories and Applications*, I. T. Young *et al.* Eds., 1986, pp. 1227–1230.
[17] H. X. Lin and H. J. Sips, "A novel floating-point on-line division algorithm," in *Proc. 8th Symp. Comput. Arithmetic*, Como, Italy, 1987, pp. 188–195.
[18] G. L. Haviland and A. A. Tuszynsky, "A CORDIC arithmetic processor chip," *IEEE Trans. Comput.*, vol. C-29, no. 2, pp. 68–79, Feb. 1980.
[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: John Hopkins Univ. Press, 1983.
[20] H. M. Ahmed, J. M. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing," *IEEE Comput. Mag.*, vol. 15, no. 1, pp. 65–86, Jan. 1982.
[21] 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. 2, pp. 242–270, 1985.
[22] A. A. J. de Lange, A. J. van der Hoeven, E. F. Deprettere, and J. Bu, "An optimal floating point pipeline CMOS CORDIC processor," in *Proc. 1988 Int. Symp. Circuits Syst.*, Helsinki, Finland, 1988, pp. 2043–2047.
[23] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," *IEEE Trans. Comput.*, to be published.

Hai Xiang Lin (S'90) was born in Guangxi, China, on September 23, 1961. He received the M.Sc. degree from the Faculty of Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands.

For his work he received the "Delftse Hogeschool Fonds" award. Since then he has been working as a graduate student at IBBC-TNO Netherlands Organization of Applied Scientific Research towards his Ph.D., where he is currently involved in a project on parallelizing finite element analysis software. His research interests include parallel computer systems architectures, parallel algorithms, and computer arithmetic.

Henk J. Sips (A'78) was born in Amsterdam, The Netherlands, on October 14, 1950. He received the M.Sc. degree in 1976 in electrical engineering and the Ph.D. degree in 1984 from Delft University of Technology, Delft, The Netherlands.

Currently he is an Associate Professor in Physics Informatics at the Delft University of Technology and a Professor in Computer Science at Brabant University. He is also head of the computer systems architecture group of the Institute of Applied Computer Science, Delft, The Netherlands. His research interests include computer architecture, parallel programming systems, parallel algorithms, and computer arithmetic.