REDUNDANT AND ON-LINE CORDIC: APPLICATION TO MATRIX TRIANGULARIZATION AND SVD

Milos Ercegovac
Tomas Lang

September 1987
CSD-870046

Redundant and On-line CORDIC: Application to Matrix Triangularization and SVD

Miloš D. Ercegovac and Tomas Lang
UCLA Computer Science Department
University of California, Los Angeles

Abstract

Several modifications to the CORDIC method of computing angles and performing rotations are presented: (i) the use of redundant (carry-free) addition instead of a conventional (carry-propagate) one; (ii) a representation of angles in a decomposed form to reduce area and communication bandwidth; (iii) the use of on-line addition (left-to-right, digit-serial addition) to replace shifters by delays; and (iv) the use of on-line multiplication, square root and division to compute scaling factors and perform the scaling operations. The modifications presented improve the speed and the area of CORDIC implementations. The proposed scheme uses efficiently floating-point representations. We discuss the application of the modified CORDIC method to matrix triangularization by Givens' rotations and to the computation of the single value decomposition (SVD).

1. Introduction

Many compute-intensive applications include matrix computations that involve the calculation of angles and their use in rotations. Examples are matrix triangularization and single value decomposition (SVD) [GOLU83]. To achieve adequate throughput, parallel structures have been proposed which are typically organized in linear, triangular, or square arrays [GENT81, CIMI81, AHME82a, LUK86, CAVA87]. The angle is computed in boundary or diagonal processors and broadcast to other processors for rotation. Several alternative implementations of the angle calculation and the rotations are possible. Of particular interest are the following two:

a) The *sine* and *cosine* of the angle are computed by means of a sequence of operations involving squaring, addition, multiplication, square root, and division. The rotation is then done by several multiplications and additions. The main advantages of this approach is that efficient implementations for the primitive operations are known and that redundancy can be used to improve the speed [ROB58, AVI61, ATKI75]. However, it requires various different modules and consists of several dependent computations. To reduce the delay introduced by this, it is possible to use the on-line approach, which allows the overlapping of these dependent operations; examples of this have been presented in [ERCE87a] and [ERCE87b].

b) Directly calculating the angle using a CORDIC operation [VOLD59, WALT71] and using the same approach for the rotation. This method has been applied to matrix triangularization in [AHME82a] and to the single value decomposition in [CAVA87]. It has as advantage that a small number of operations is required and that the same module can be used for both the angle

calculation and the rotation. However, the conventional implementation of the CORDIC module has two disadvantages: it is slow, because it involves recurrences including carry-propagate addition and variable shifting, and area-consuming because of the need for variable shifters and ROMs to store angle constants.

We present here a modification of the implementation of the CORDIC modules to improve the speed and reduce the number of shifters. Moreover, we design special CORDIC-based modules for the different operations, such as angle calculation, rotation, and two-sided rotation. These special-purpose CORDIC modules have advantages in speed and area with respect to the general-purpose modules developed by [VOLD59] and [WALT71] and used by [AHME82a] and [CAVA87].

The main improvements developed in this paper are:

i) Modification of the standard CORDIC module for the calculation of the angle $\tan^{-1}(a/b)$ so that redundant addition is used. This results in a significantly faster operation than that obtained with carry-propagate adders. Since the use of redundant addition makes the scaling factor variable, we develop an implementation of the computation of this variable factor.

ii) The angle is transmitted in decomposed form for use in the rotations. This reduces the communication bandwidth and eliminates the need of the angle recurrences in both CORDIC modules.

iii) Implementation of the rotation modules using on-line additions [ERCE84, IRWI87]. This replaces the area-consuming shifters by more area-efficient delays.

Both ii) and iii) above were already introduced for the nonredudundant CORDIC case in [ERCE87b].

To achieve good numerical properties and simplify the use of the system in a variety of environments, it is convenient to perform the computations using a floating-point representation. In [ERCE87a] this representation was used to implement the on-line computation of Givens' sin/cos rotations factors. In [AHME82b], the use of floating-point representation in CORDIC computations is discussed. However, the approach suggested there produces a reduction in speed because of the use of floating-point additions. Here we consider simplifications to this scheme to eliminate the overhead.

In Section 2 we present the redundant CORDIC module for the calculation of the angle. This redundant calculation makes it necessary to compute a variable scaling factor (in contrast with the constant scaling factor used in conventional CORDIC); the calculation of this factor is presented in Section 3. In Section 4 we develop the on-line CORDIC module for rotation. Sections 5 and 6 present the application of these techniques to the matrix triangularization and single value decomposition computations. Finally, we conclude with speed comparisons with the other approaches that have been presented.

2. Redundant CORDIC for angle calculation

The CORDIC scheme [VOLD59, WALT71] can be used to compute the angle θ, such that

$$\theta = \tan^{-1}(\frac{a}{b})$$

This calculation is done by the following set of recurrences

$$x_a[j+1] = x_a[j] + \sigma_j 2^{-j} y_a[j]$$

$$y_a[j+1] = y_a[j] - \sigma_j 2^{-j} x_a[j]$$

$$z_a[j+1] = z_a[j] + \sigma_j \tan^{-1}(2^{-j})$$

with

$$x_a[0] = b, \quad y_a[0] = a, \quad z_a[0] = 0$$

and the results being

$$x_a[n] = K(a^2 + b^2)^{1/2}, \qquad \theta = z_a[n]$$

where

$$K = \prod_{j=0}^{n-1}(1 + \sigma_j^2 2^{-2j})^{1/2}$$

In conventional CORDIC, the value of σ_j is obtained as

$$\sigma_j = \begin{cases} 1 & \text{if } y_a[j] \geq 0 \\ -1 & \text{if } y_a[j] < 0 \end{cases}$$

As mentioned in Section 1, this CORDIC operation is relatively slow because each of the n steps requires a carry-propagate addition and a variable shift. Moreover, it uses two shifters which are quite area-consumming. We now present the two following modifications to improve the implementation.

i) Elimination of one of the shifters by transforming the recurrences as follows. Let

$$w[j] = 2^j y_a[j]$$

Then the recurrences are transformed to

$$x_a[j+1] = x_a[j] + \sigma_j 2^{-2j} w[j]$$

$$w[j+1] = 2(w[j] - \sigma_j x_a[j])$$

$$z_a[j+1] = z_a[j] + \sigma_j \tan^{-1}(2^{-j})$$

$$\sigma_j = \begin{cases} 1 & \textbf{if } w[j] \geq 0 \\ -1 & \textbf{if } w[j] < 0 \end{cases}$$

This transformation leaves just one shifter for the x_a recurrence. Moreover, note that this form shows that the value of $x[j]$ does not change after $j=n/2$, that is, $x[n] = x[n/2]$ to the implementation precision.

ii) Replacing the carry-propagate addition by a redundant addition (carry-save or signed-digit). This approach has been used previously in the implementation of other operations, such as division and square root [ROBE58, METZ65, AVI61, ATKI75]. This requires that the determination of σ_j uses an estimate of $w[j]$ instead of its fully assimilated value. To make this possible, it is necessary to produce a redundant representation of θ in terms of the σ_j's. This is achieved by allowing σ_j to take values from the set $\{-1,0,1\}$ instead of from the set $\{-1,1\}$, which is the one used in conventional CORDIC.

The corresponding selection function for σ_j using this redundant set is developed in the Appendix A. To have the required overlap between the allowed selection intervals it is necessary to normalize $x[j]$. As shown in the next section, when floating-point representation is used it is possible to have $x[1]$ normalized, that is,

$$|x[1]| \geq 1/2$$

The specific selection function depends on the type of redundant representation used. As shown in Appendix A, for 2's complement carry-save representation the selection function is

$$\sigma_j = \begin{cases} 1 & \textbf{if } \hat{w} \geq 0 \\ 0 & \textbf{if } \hat{w} = -1/2 \\ -1 & \textbf{if } \hat{w} \leq -1 \end{cases}$$

where $\hat{w}$ is an estimate of $w[j]$ with a precision of 1 fractional bit.

The binary-level specification of this selection function and the corresponding switching expressions are given also in Appendix A.

Similarly, for signed-digit representation the selection function is

$$\sigma_j = \begin{cases} 1 & \textbf{if} \quad \hat{w} \geq 1/2 \\ 0 & \textbf{if} \quad \hat{w} = 0 \\ -1 & \textbf{if} \quad \hat{w} \leq -1/2 \end{cases}$$

where again $\hat{w}$ is computed using 1 fractional bit of w.

The implementation of the corresponding recurrences using the carry-save approach is shown in Figure 1. The step time corresponds to the shifter delay plus the 4-2 carry-save adder, since the selection function is overlapped with the shifting. This step time is significantly smaller than in the nonredundant case where carry-propagate addition is required. Depending on the technology, the speed up should be between 4 and 6.

The resulting angle θ can be produced in two alternative forms, as follows:

i) The angle in decomposed form is represented by the sequence of $\sigma_i's$. This form is used directly in the rotation for the triangularization case, as discussed in Section 5. In this case, it is not necessary to implement the angle recurrence z.

ii) The angle is represented by the carry-save form. This form is used in the SVD case, as shown in Section 6.

On-line implementation

An alternative implementation of the redundant CORDIC recurrences is to use on-line addition [ERCE84, IRWI87] instead of parallel redundant addition. In such an implementation, the recurrence is unfolded and on-line adders are used. The main advantage of this implementation is the replacement of the area-consuming shifters by more efficient delays, as shown in Figure 2a.

Since the determination of σ_j requires the three most significant sbits (signed binary digits) of $w[j]$ and this value is obtained by a multiplication by two, to have a fast implementation three sbits of w are produced per clock cycle and the on-line adder incorporates the multiplication by 2. The implementation of this on-line addition is shown in Figures 2b and 2c, where binary signed-digit adders are used [AVIZ61, KUNI87]. To feed these adders also three sbits of x have to be produced per clock cycle. In such a case, the shifting by 2^{-2j} is done by a combination of wiring and of delays, as shown in Figure 2d. Let us call $v[j]$ the vector formed by grouping the sbits of $w[j]$ in groups of three sbits. Moreover, let $v_i[j]$ be the *ith* group and $v_i^k[j]$ the *kth* sbit inside a group (k=0,1,2). Then, the delay for sbit $v^k[j]$, $k = 0,1,2$ is $d_k = 2j + k$. Consequently, the number of delay cells for sbit k, denoted by c_k is

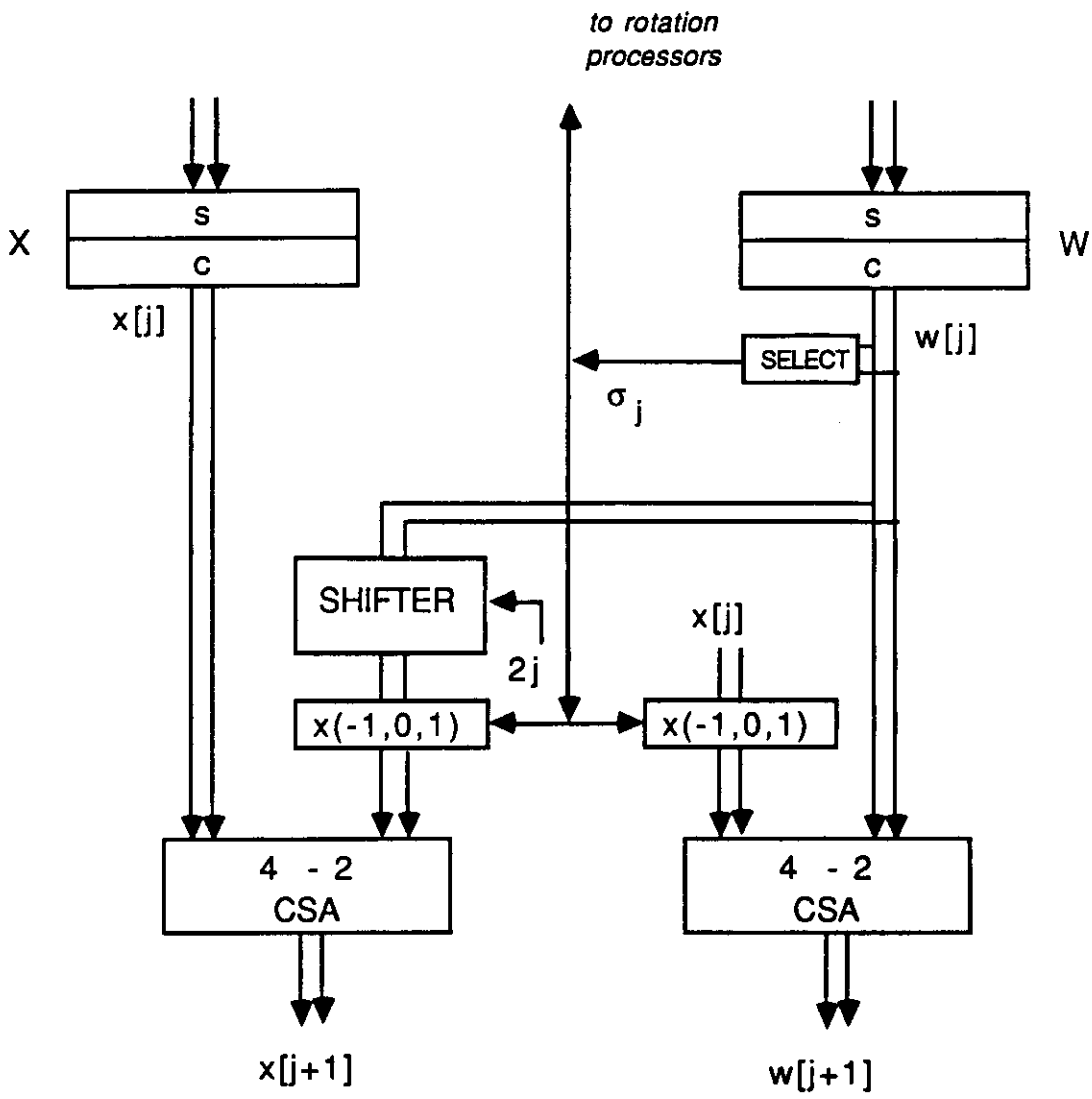


Figure 1: Implementation of x,w Recurrences for Angle Calculation

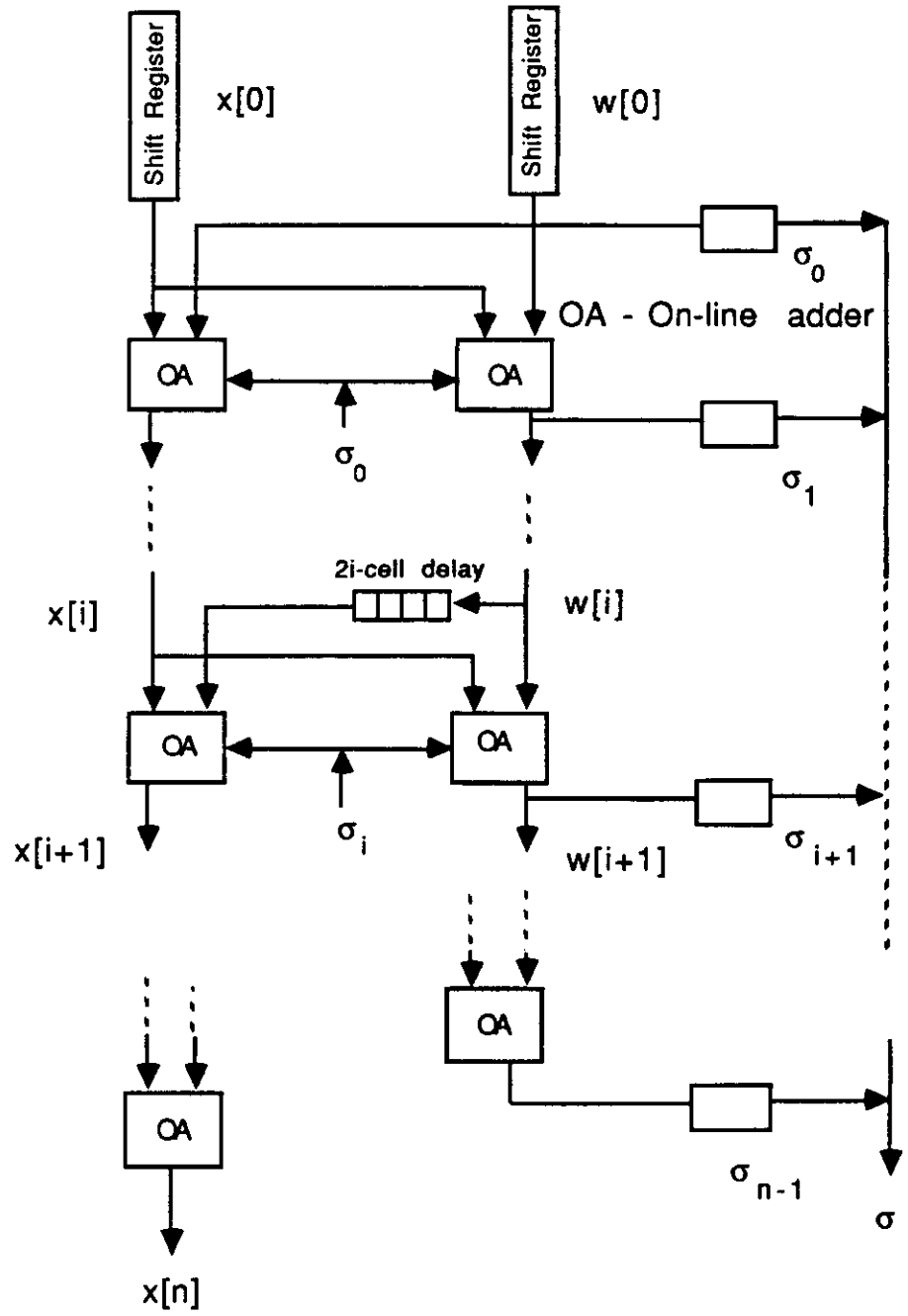


Figure 2a. On-Line Scheme
for Angle Computation

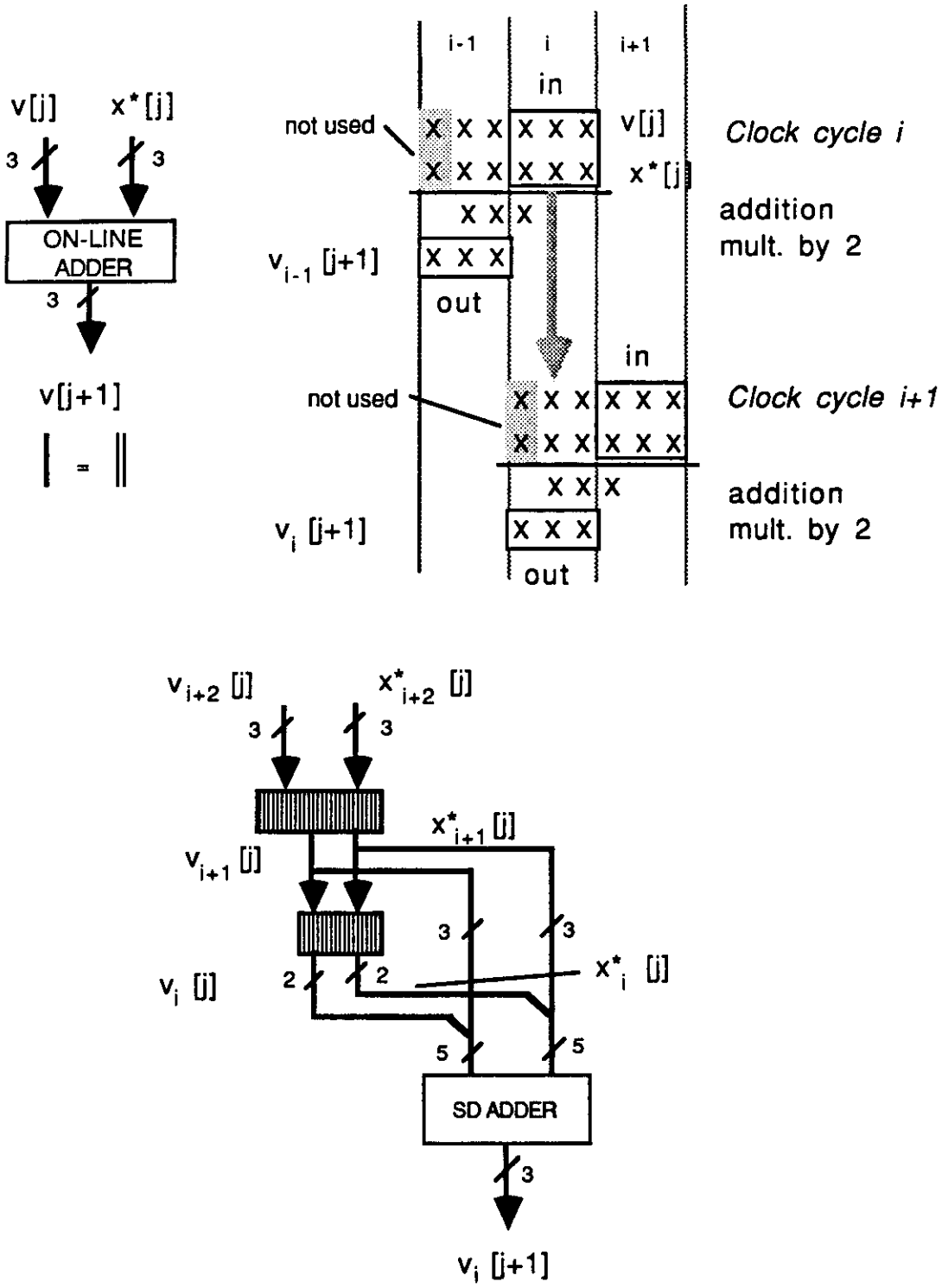


Figure 2b: On-Line Adder Organization

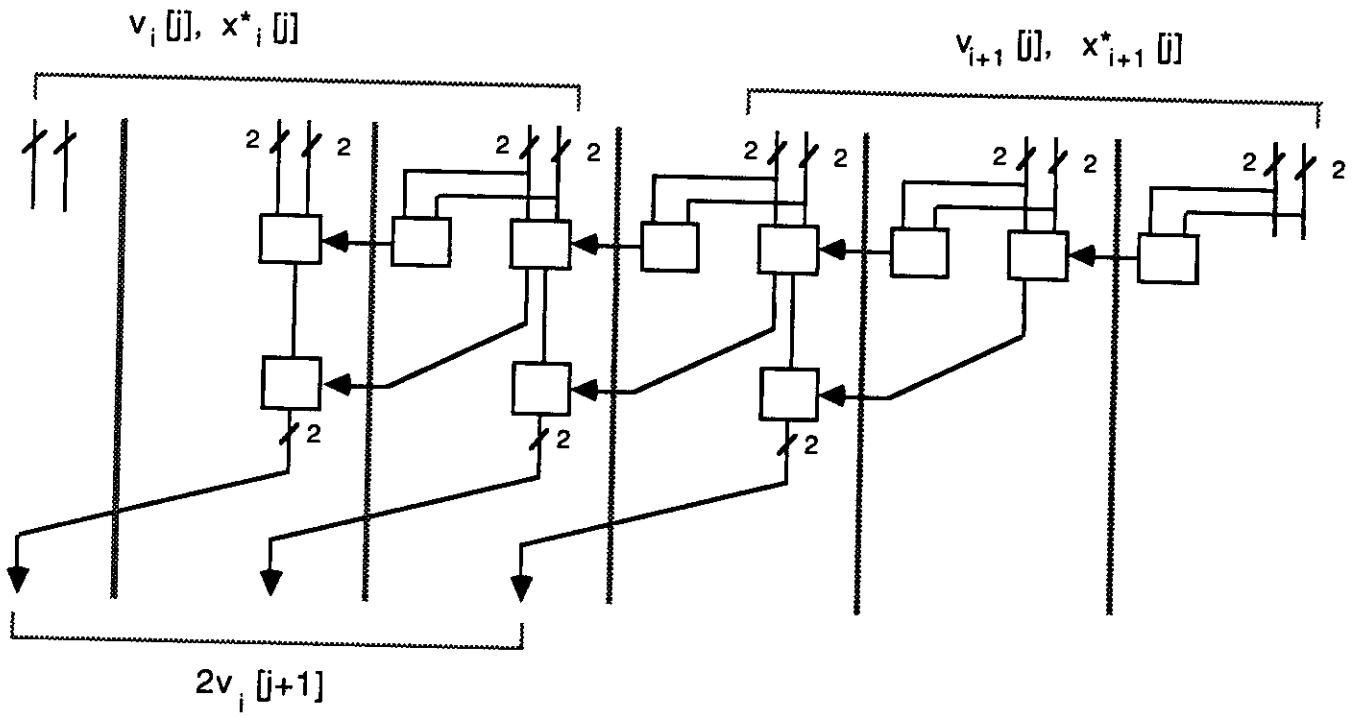


Figure 2c: Signed-Digit Adder

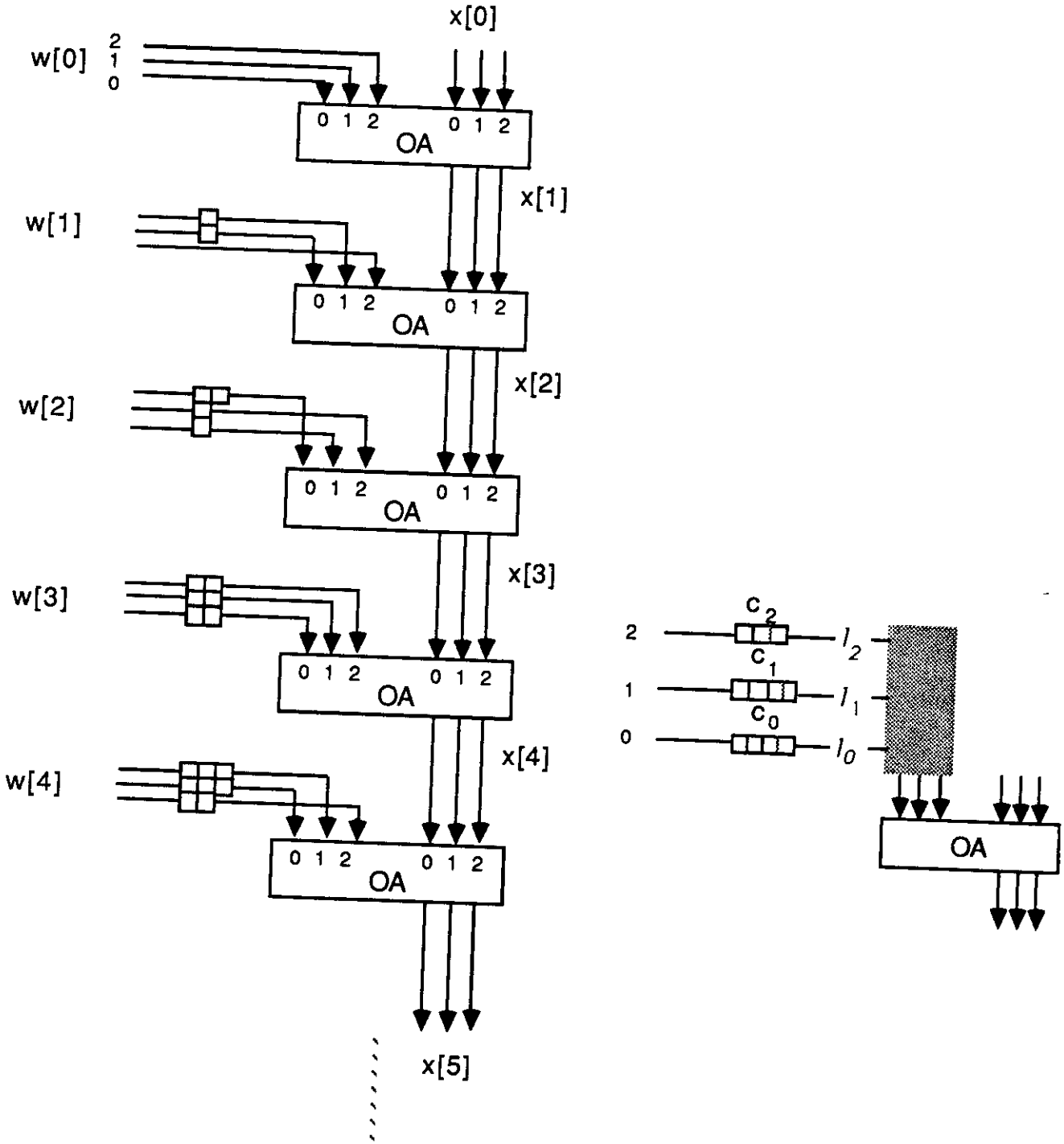


Figure 2d: Scheme for Shifting

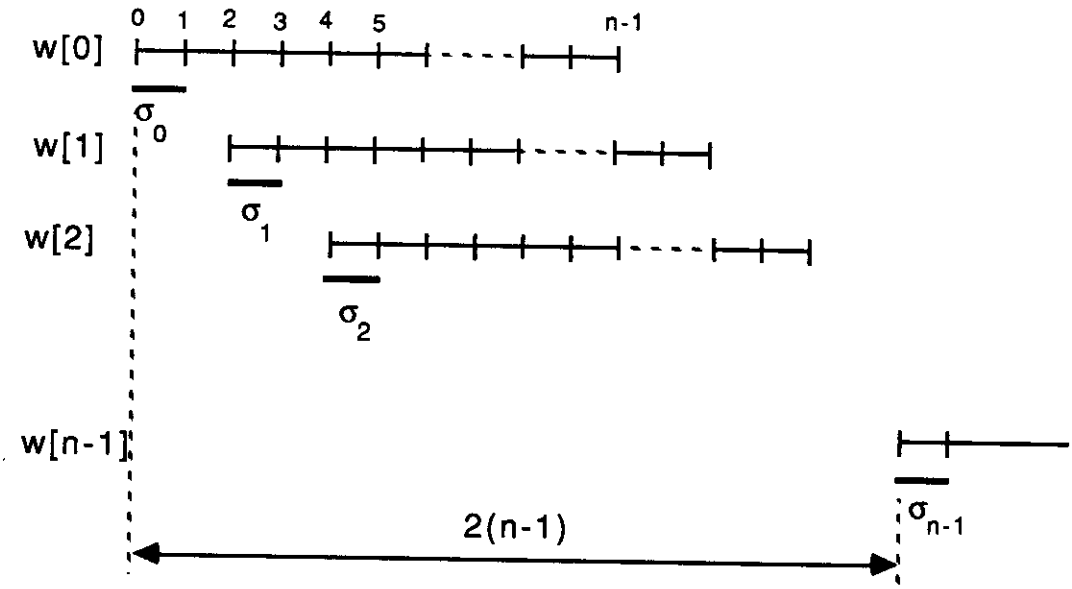


Figure 2e: Timing of On-Line Recurrence for Angle Computation

$$c_k = \left\lfloor \frac{d_k}{3} \right\rfloor$$

and the number of the adder input to which sbit k is connected, denoted l_k, is

$$l_k = d_k \ mod\ 3$$

The resulting on-line delay of addition is one clock cycle, that is two groups of three sbits of the operands are needed before the first group of three sbits of the result is produced. Consequently, the delay between the initiation of $w[j]$ and of $w[j+1]$ is two clock cycles. The timing is shown in Figure 2e. Because of this on-line delay, this scheme requires more clock cycles than the redundant parallel method. However, since no variable shifter is required, the clock period can be smaller in the on-line case. Moreover, since the rotations are performed with on-line recurrences, the match between both components might be better.

Floating-point representation

We now consider the modifications required for the described implementations when floating-point representations are used. In [AHME82b] a floating-point CORDIC is described. However, it uses floating-point adders to implement the recurrences. This is not attractive because the alignment requirements make the requrence step slower. We now develop a simpler alternative adapted to the angle calculation.

Assume that the initial values for the circular CORDIC are represented in normalized floating point as

$$x[0] = A_x \cdot 2^{a_x} \quad 1/2 \le |A_x| < 1$$

$$w[0] = y[0] = A_y \cdot 2^{a_y} \quad 1/2 \le A_y < 1$$

As indicated in [Ahme82b], it is possible to use directly this representation to perform the CORDIC recurrences if we use floating-point additions. However, we can eliminate this by aligning just in the first iteration, as follows. From the recurrence we get,

$$x[1] = x[0] + \sigma_0 w[0]$$

$$w[1] = 2(w[0] - \sigma_0 x[0])$$

It is always possible to select $\sigma_0 \neq 0$ because the sign of $w[0]$ is known. Consequently, a floating-point addition and subtraction will produce

$$x[1] = A^{+} \cdot 2^{a}$$

$$w[1] = 2A^{-} \cdot 2^{a}$$

where $a = max(a_x, a_y)$ and $x[1]$ is a normalized fraction (as required by the selection function).

After this first step, the iterations are performed using the fractions A^{+} and A^{-}. The resulting angle is correct, since it depends only on $x[0]/y[0]$. Note that the angle is not in a floating-point representation; its range is

$$\theta_{min} = \tan^{-1}(2^{-(n-1)}) \approx 2^{-(n-1)}, \quad \theta_{max} = \sum_{i=0}^{n-1} \tan^{-1}(2^{-i})$$

Since the maximum is larger than $\pi/2$, this produces no problem. The minimum value depends on n, but should be adequate for most applications.

In the triangularization application it is also necessary to compute the new diagonal element, which is $x[n]/K$. When the previously described method for floating-point representation is used, it is necessary to multiply the resulting $x[n]$ by 2^{a}.

3. Scale factor calculation

For the computation of the new diagonal element in the triangularization case and for the rotation in both triangularization and SVD, it is necessary to compensate by the scale factor introduced by the inexact rotation of CORDIC [WALT71]. The value of this scaling factor is

$$K = \prod_{j=0}^{n-1} (1 + |\sigma_j| 2^{-2j})^{1/2}$$

Since in the redundant case the set of values of σ_i is $\{-1,0,1\}$ (in contrast with conventional CORDIC where it is $\{-1,1\}$), the value of K is not constant. Consequently, its value has to be computed and the compensation has to be done by actual division, since other methods, such as the one proposed in [DELO83], depend on the fact that the scaling factor is constant. We now describe an on-line algorithm for the computation of K. The algorithm has two steps:

i) Compute

$$P = \prod_{j=0}^{n-1} (1 + |\sigma_j| 2^{-2j})$$

by the recurrence

$$P[j+1] = P[j] + |\sigma_j| \cdot 2^{-2j} P[j]$$

with

$$P[0] = 1 \quad \text{and} \quad P = P[n] = P[n/2]$$

We use an on-line implementation, which unfolds the recurrence and uses shift registers for the delay, as shown in Figure 3. Note that only $n/2$ stages are needed.

ii) Compute $K = P^{1/2}$ by an on-line square-root algorithm [ERCE87a].

4. On-line CORDIC rotation

The rotation of the vector M by the angle θ is defined by

$$R[\theta] \begin{bmatrix} m_1 \\ m_2 \end{bmatrix}$$

where

$$R[\theta] = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

If the angle is known in its decomposed form, such that

$$\theta = \sum_{j=0}^{n-1} \sigma_j \tan^{-1}(2^{-j})$$

the rotation can be performed by a (partial) CORDIC operation, consisting of the recurrences

$$x_r[j+1] = x_r[j] + \sigma_j 2^{-j} y_r[j]$$

$$y_r[j+1] = y_r[j] - \sigma_j 2^{-j} x_r[j]$$

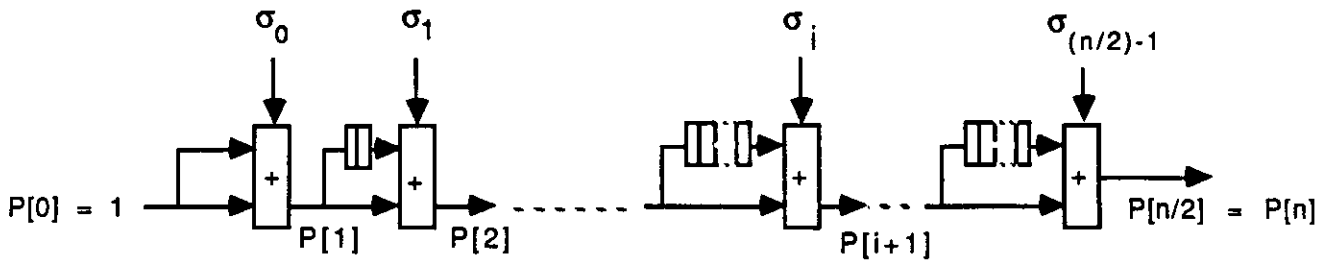


Figure 3: Implementation of Scaling-Factor Recurrence

with the initial conditions

$$x_r[0] = m_1 \quad y_r[0] = m_2$$

After n steps, the result is

$$\begin{bmatrix} x_r[n] \\ y_r[n] \end{bmatrix} = KR[\theta] \begin{bmatrix} m_1 \\ m_2 \end{bmatrix}$$

where

$$K = \prod_{i=0}^{n-1} (1 + |\sigma_i| \cdot 2^{-2i})^{1/2}$$

The CORDIC operation is partial because it uses the angle produced by another CORDIC operation in decomposed form. Consequently, no angle recurrence is needed. Moreover, the $\sigma's$ are passed in series (most significant first) so that the rotation can be overlapped with the angle calculation.

To keep up with the fast recurrence step obtained in the computation of the angle when redundant additions are used, the rotation CORDIC has also to use redundant addition. Since in this case there is no computation of $\sigma's$, a suitable implementation uses an on-line version. In this implementation, the recurrence is unfolded and on-line adders are used, as shown in Figure 4a. The main advantage of this scheme with respect to the parallel form is that the area-consuming shifters are replaced by more efficient delays. To obtain an on-line delay of 1 clock cycle, the additions should be radix 4. In this case, the interval between initiations of consecutive iterations in the on-line circular CORDIC is of 2 clock cycles.

As indicated before, the CORDIC operation produces a rotation multiplied by the scaling factor K. This scaling factor is computed as indicated in Section 3. The correction by this factor is performed by two on-line divisions, as shown in Figure 4a. The result of these divisions is in signed-digit form; to convert them to conventional form a on-the-fly conversion is used, as presented in [ERCE87c]. The timing of this on-line rotation processor is shown in Figure 4b.

Floating-point representation

The use of floating-point representation has similar characteristics as those discussed for the angle. Let the initial values be

$$x[0] = B_x \cdot 2^{b_x} \quad 1/2 \le B_x < 1$$

and

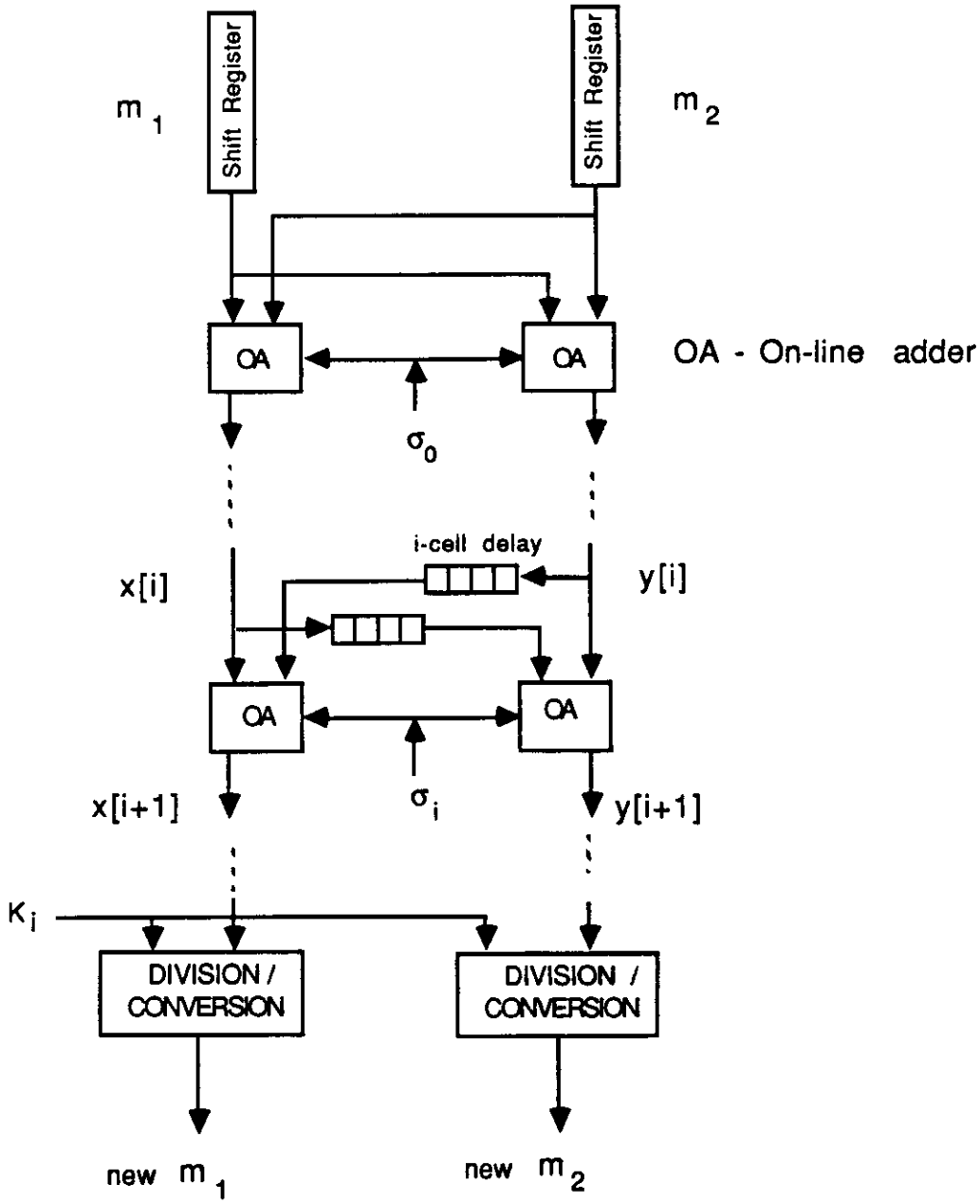


Figure 4a: x,y Recurrence
for Rotation

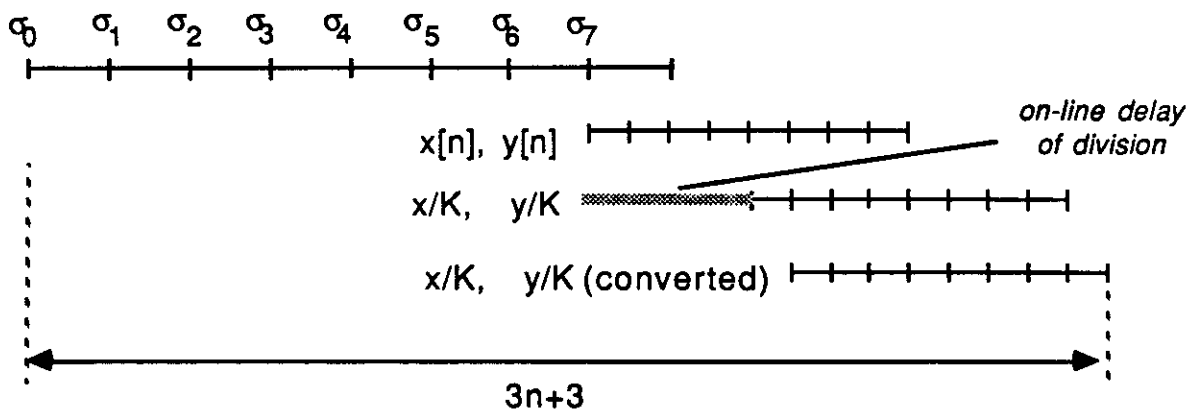


Figure 4b: Timing of the Rotation Processor

$$y[0] = B_y \cdot 2^{b_y} \quad 1/2 \le B_y < 1$$

As before, a floating-point addition and a subtraction produces

$$x[1] = B^+ \cdot 2^b$$

$$y[1] = B^- \cdot 2^b$$

The scaled values B^+ and B^- are used for the remaining iterations, producing $x'[n]$ and $y'[n]$, so that the final results are

$$x[n] = x'[n] \cdot 2^b$$

$$y[n] = y'[n] \cdot 2^b$$

5. Application to Matrix Triangularization

The triangularization of a matrix by Givens' rotations has been discussed in [GOL83]. Traditionally, this transformation was proposed for execution in a sequential computer. Recently, interest has evolved in the computation using parallel computers [SAME78] and concurrent structures of processing elements [GENT81], to improve the speed of execution. Due to the nature of the transformation, linear and triangular arrays have been proposed [CIMI81, AHME82a, GENT81]. If the basic operations performed by the processing elements are additions, multiplications, division, and square root, the boundary processors perform a more complex computation than the others and, therefore, determine the step time. Several proposals have been made to reduce this imbalance, such as the elimination of square root operations [GENT73], the use of on-line computation [CIMI81, ERCE87], and the utilization of the CORDIC approach [AHME82a]. We now develop an implementation using the redundant and on-line CORDIC scheme presented in the previous sections; this improves the speed by having a smaller step time, reduces the need of shifters, and allows the overlapping of the CORDIC rotation with the divisions required to compensate for the scaling factor.

Givens' rotations are used in the solution of linear equations of the form $Ax=b$ [AHME82a, GOLU83]. The algorithm triangularizes the $m \times m$ matrix A with a sequence of plane rotations. The following is a sequential description of the algorithm:

For r=1 to m

Begin

For $i=r+1$ to m

Begin

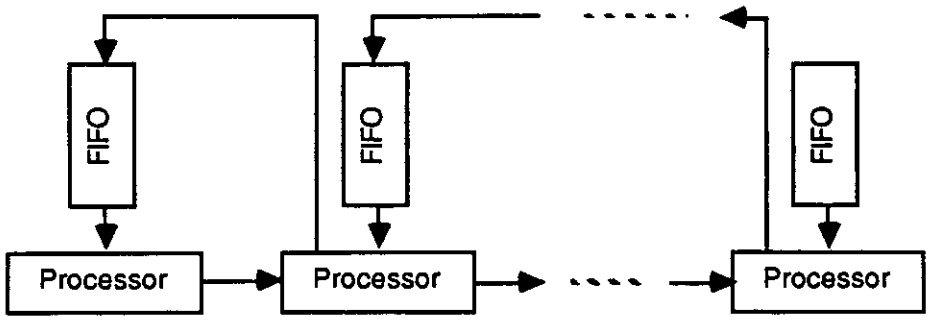


Figure 5: Linear Array of Processors
for Givens Rotations

$\theta_{ri} \leftarrow -\tan^{-1}(a_{ir}/a_{rr})$; $a_{rr} \leftarrow (a_{rr}^2 + a_{ir}^2)^{1/2}$

For $j=r+1$ to m

Begin

$$\begin{bmatrix} a_{rj} \\ a_{ij} \end{bmatrix} \leftarrow \begin{bmatrix} \cos\theta_{ri} & -\sin\theta_{ri} \\ \sin\theta_{ri} & \cos\theta_{ri} \end{bmatrix} \begin{bmatrix} a_{rj} \\ a_{ij} \end{bmatrix}$$

End

$$\begin{bmatrix} b_r \\ b_i \end{bmatrix} \leftarrow \begin{bmatrix} \cos\theta_{ri} & -\sin\theta_{ri} \\ \sin\theta_{ri} & \cos\theta_{ri} \end{bmatrix} \begin{bmatrix} b_r \\ b_i \end{bmatrix}$$

End
End

In [GENT81, CIMI81, AHME82a] linear and triangular arrays are proposed to perform the triangularization. For instance, in the linear array, shown in Figure 5, the angle θ_{ri} is computed in the leftmost processor and is transferred to the right while data is transferred both to the left and to the right. The leftmost processor computes the angle and all other processors compute the rotations. The scheme we propose is applicable to both types of arrays.

Implementation using CORDIC.

In [AHME82a] a method using CORDIC is proposed for both the computation of the angle and of the rotation. The justification given there is that "the rotation is performed in no more time than a bit-serial multiplication". However, this is not completely accurate because the CORDIC scheme needs a variable shift and a carry-propagate addition, while multiplication does not require such a shift and can be done with carry-save addition. Consequently, the step time in multiplication can be made significantly smaller than for CORDIC. Moreover, additional steps are required to compensate for the scaling factor introduced by the CORDIC rotation (the number of steps for scaling ranges from $2n$, if CORDIC divisions are used, to $n/4$ if repeated steps are used for compensation [DELO83]). Consequently, the time of the angle calculation is of n CORDIC steps followed by the rotation which, in the best case, corresponds to $1.25n$ CORDIC steps. In both cases, the step time is determined by the shifting and the carry-propagate addition.

More specifically, the algorithm suggested in [AHME82b] is as follows:

i) The leftmost processor computes the rotation angle and the new diagonal element by means of a circular CORDIC. To obtain the new element it is also necessary compensate for the scaling factor, which can be achieved by a division using a linear CORDIC or by the repetition of some CORDIC steps [DELO83]. The total delay is, therefore, between $1.25n$ and $2n$ CORDIC steps. The angle is transmitted to the rotation processors.

ii) The other processors perform the rotation by a circular CORDIC. They use the angle produced by the angle processor. The compensation for the scaling factor is performed in the same way as in i). The total delay of the rotation is, therefore, of between $1.25n$ and $3n$ CORDIC steps.

Problems with the CORDIC Implementation

From the previous discussion we can conclude that each iteration of the triangularization algorithm is quite slow, since it requires at least $2.25n$ CORDIC steps, and the delay of a step is determined by the variable shift and the carry-propagate adder.

Implementation Using Redundant and On-line CORDIC

We now show the implementation using the redundant and on-line CORDIC operations presented in Sections 2, 3, and 4. As in the scheme described in [AHME82a], there are two types of processors: the *angle processors* and the *rotation processors*. We now discuss our scheme for these processors and their interface.

The angle processor

The angle processor computes the angle θ_{ri} and the new diagonal element a_{rr} using the redundant CORDIC presented in Section 2. It also calculates the scale factor as described in Section 3. Moreover, an on-line division is used to correct by the scale factor to obtain the new a_{rr}. The result of the on-line division is in signed-digit form; it is converted to conventional form using the on-the-fly conversion presented in [ERCE87c].

Consequently, the angle processor consists of the CORDIC recurrences, the on-line recurrence for K^2, the on-line square-root unit, the on-line divider/conversion unit, as shown in Figure 6a.

The timing is shown in Figure 6b. Because of the shifter and the 4-to-2 carry-save adder, the step time of the CORDIC recurrence is larger than that of the other recurrences (for P, on-line square root, and on-line division). Therefore, we use a clock period corresponding to the smaller steps and assign two clock cycles to the CORDIC recurrence step.

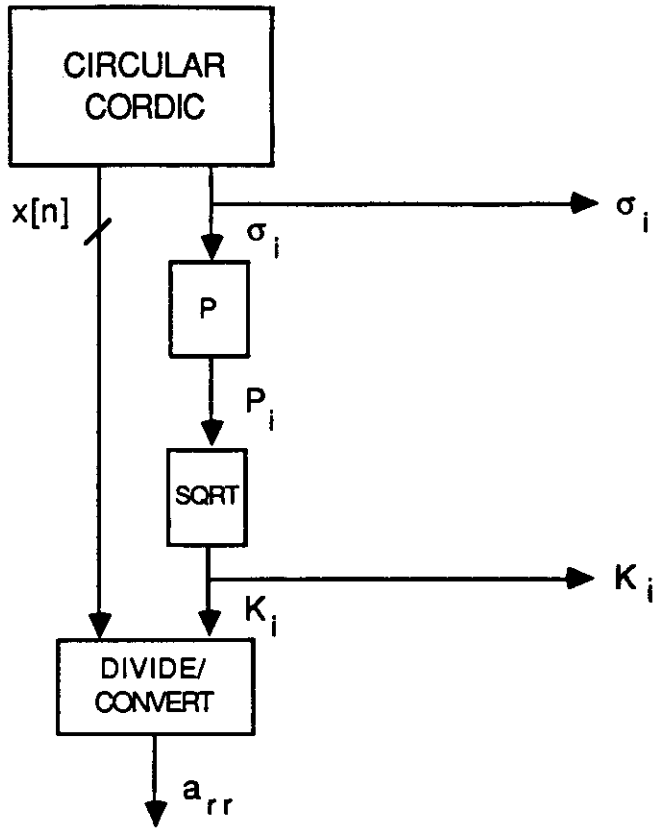


Figure 6a: Angle Processor Organization
for Triangularization

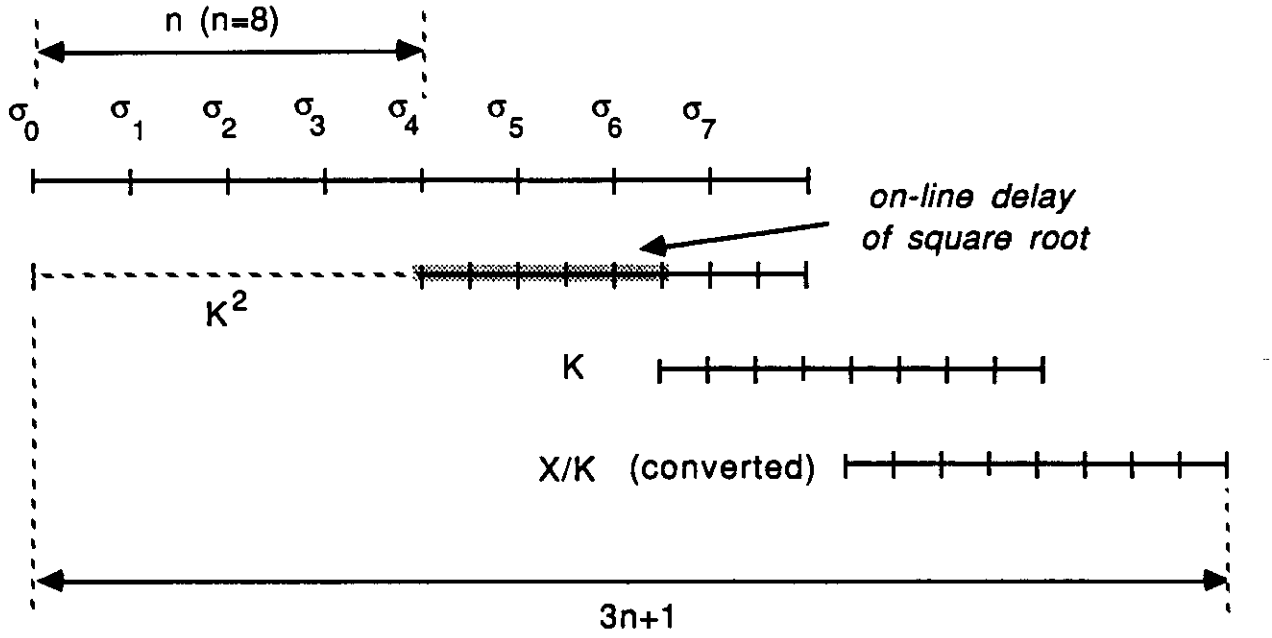


Figure 6b: Timing of the Angle Processor

On-line rotation processor

The rotation processor performs the rotation by means of an (partial) on-line circular CORDIC, as described in Section 4. The division by the scaling factor K is done by two on-line division units, which include the conversion to conventional representation.

Overall timing

The overall timing is shown in Figure 7. The time of one iteration of the triangularization process is $3n+3$ cycles (determined by the rotation). As indicated before, the step time of the angle processor is two clock cycles, while for the other recurrences it is one clock cycle.

The speed of the scheme presented here compares favorably with the one of using conventional CORDIC, as proposed in [AHME82a]. In that implementation, the number of clock cycles is $2.25dn$, where d is the number of clock cycles per CORDIC step (carry-propagate addition and shifting). For an estimated value of d=6 (to use the same clock period in both implementations), we conclude that the scheme presented here would be approximately 4.5 times faster.

Since this speed improvement is due to two factors, namely, the overlap between the angle calculation and the rotation and the use of redundant addition, we now separate the effect of these factors. If the original scheme proposed in [AHME82a] is modified so that overlapping is possible, the time would be reduced to $1.25dn$, with a speed-up of 1.8. The use of the redundant adder is responsible for the other 2.5 speed-up factor.

6. Application to SVD

The single value decomposition (SVD) is important in many matrix computations. For the definition of this transformation, the basic algorithms for its computation, and its applications the reader is directed to [GOLU83] and [LUK86]. Because of the computation-intensive nature of the algorithms, great interest has appeared on parallel arrays, as discussed in [BRENT85a], [BRENT85b], [LUK86], and [CAVA87]. The primitive operation in these cases is the diagonalization of a 2x2 matrix by the rotations $R(\theta_l)$ and $R(\theta_r)$ (using the notation in [CAVA87]), such that

$$R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} e & 0 \\ 0 & f \end{bmatrix}$$

where θ_l and θ_r are the left and right rotation angles, respectively. The corresponding rotation matrix is

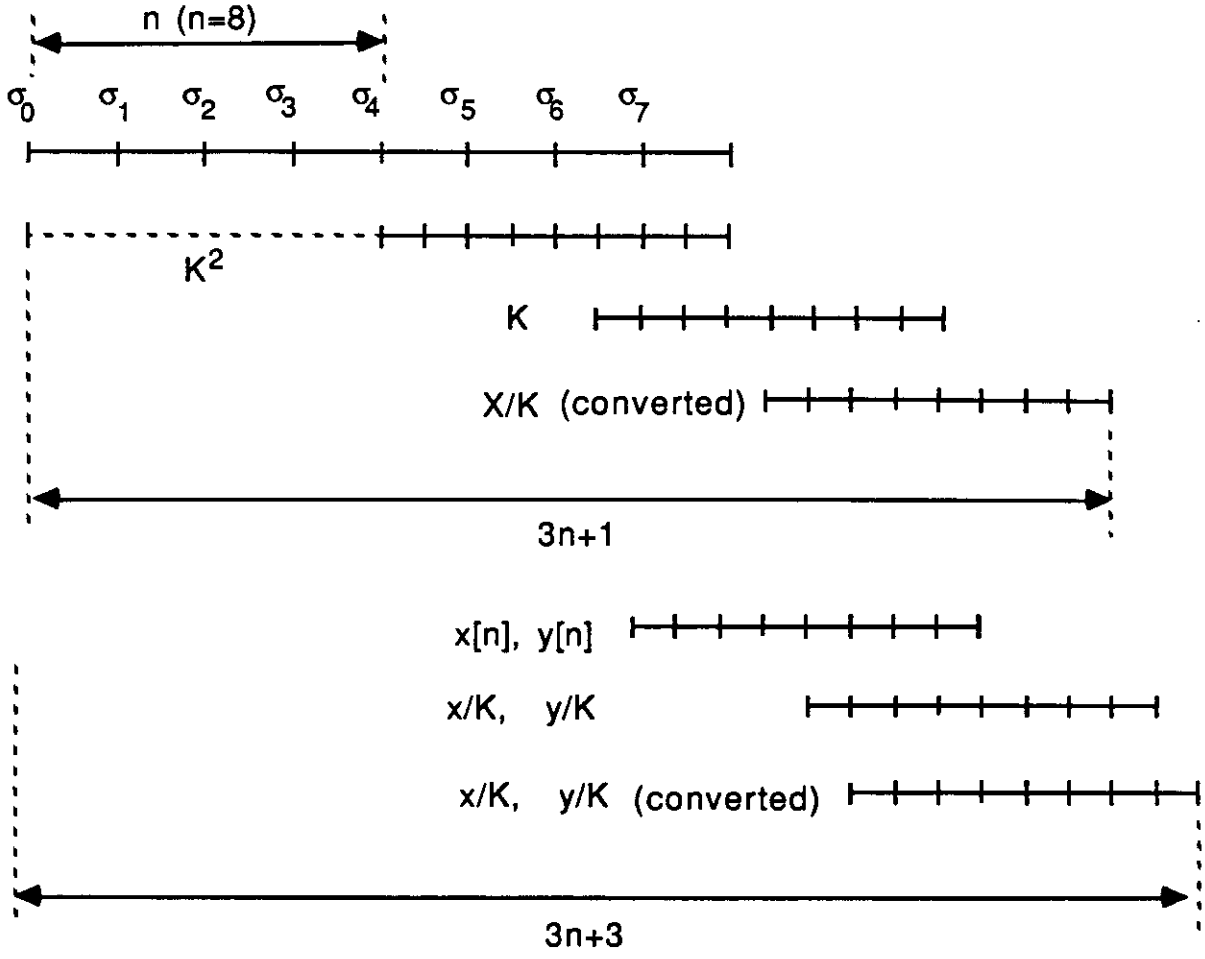


Figure 7: Overall Timing for Triangularization

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Several methods can be used to perform these rotations, in particular the two-step method and the direct two-angle method [BREN85]. Because of the use of CORDIC we consider here the latter method.

With respect to the implementation, two approaches are possible: a) the computation of $\cos\theta$ and $\sin\theta$ by a sequence of primitive operations, such as squaring, division, and square root, or b) the use of the CORDIC procedure for direct computation of the angles and of the rotations. An implementation using the first approach using the two-step method is reported in [BREN85] and of the second approach using the two-angle method in [CAVA87]. In [CAVA87] a comparison of these two implementations is made, in terms of area and time complexity.

In this paper, we illustrate the use of the redundant and on-line CORDIC schemes discussed in Sections 2,3 and 4 to improve the speed of execution of the two-angle algorithm.

Direct two-angle algorithm for SVD using CORDIC

The algorithm (Figure 8) first computes, by means of two concurrent CORDIC circular operations, the angles

$$\theta_s = \tan^{-1}\left(\frac{c+b}{d-a}\right)$$

$$\theta_d = \tan^{-1}\left(\frac{c-b}{d+a}\right)$$

Then, the two angles θ_l and θ_r are obtained as

$$\theta_l = \frac{(\theta_s - \theta_d)}{2}$$

$$\theta_r = \frac{(\theta_s + \theta_d)}{2}$$

Finally, the two-sided rotation is performed by the sequence of two CORDIC operations.

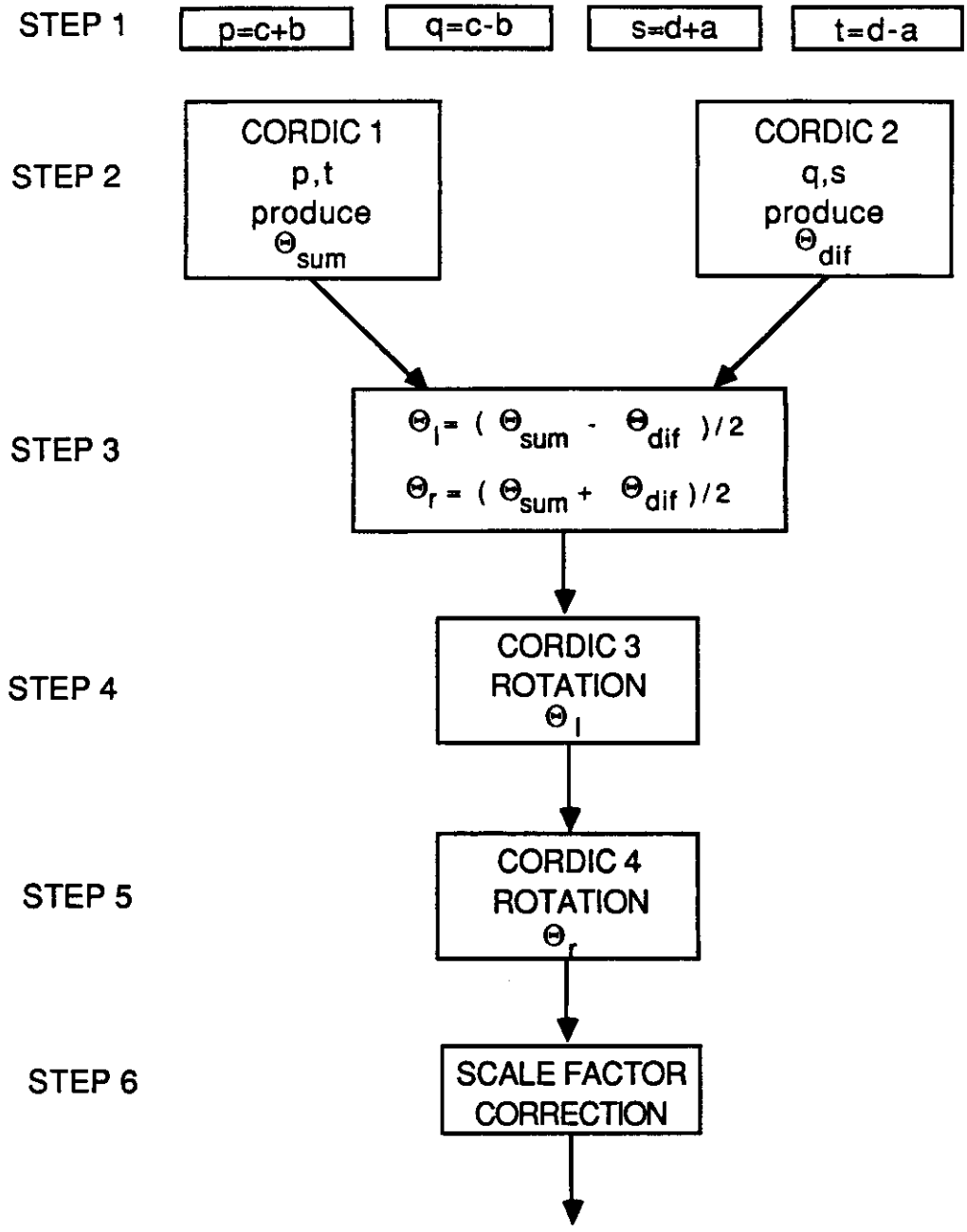


Figure 8: CORDIC Scheme for SVD

Implementation

In [CAVA87] this scheme is implemented quite directly (except for the correction factor, which is incorporated in the rotations). This results in a time of $3.25T_c$, each T_c corresponding approximately to n carry-propagate additions, and in an area essentially equal to two CORDIC modules. In [ERCE87b] we proposed modifications to the implementation that improve the speed by a factor of approximately 3.5. However, the speed is still basically dependent on the time to perform a carry-propagate addition of n bits. Here we use the redundant adder version presented in Section 2 to further improve the speed.

Computation of θ_s and θ_d

We present an implementation that uses redundant CORDIC operations to calculate the angles θ_s and θ_d, with the resulting improvement in speed because of the smaller addition time. This implementation has been discussed in Section 2. The resulting angles are represented by the $\sigma's$ as follows:

$$\theta_s = \sum_{i=0}^{n-1} \sigma_i^s \cdot \tan^{-1}(2^{-i})$$

$$\theta_d = \sum_{i=0}^{n-1} \sigma_i^d \cdot \tan^{-1}(2^{-i})$$

The $\sigma_i's$ are computed by the CORDIC unit shown in Figure 1. To compute both angles concurrently requires two CORDIC units. To reduce the amount of hardware, it is possible to pipeline one unit.

Computation of θ_l and θ_r

Then the angles θ_l and θ_r have to be computed. The simplest way to do this is to compute σ_i^l and σ_i^r directly from σ_i^s and σ_i^d. The corresponding relations are

$$\sigma_i^l = \frac{\sigma_i^s - \sigma_i^d}{2}, \qquad \sigma_i^r = \frac{\sigma_i^s + \sigma_i^d}{2}$$

resulting in the following table:

σ_i^s	σ_i^d	σ_i^l	σ_i^r
1	1	0	1
1	0	1/2	1/2
1	-1	1	0
0	1	1/2	-1/2
0	0	0	0
0	-1	-1/2	1/2
-1	1	-1	0
-1	0	-1/2	-1/2
-1	-1	0	-1

In contrast with the implementation discussed in [ERCE87b], it is not possible to use directly these values of σ_i^l and σ_i^r for the rotations because of the values $\pm 1/2$, which do not lead to a simple rotation step. Because of this, we have to use these values to compute another decomposition with the digit set $\{-1,0,1\}$. That is, we compute the sequences of γ_i^l and γ_i^r such that

$$\theta_l = \sum_{i=0}^{n-1} \sigma_i^l \cdot \tan^{-1}(2^{-i}) = \sum_{i=0}^{n-1} \gamma_i^l \cdot \tan^{-1}(2^{-i}) \quad \sigma_i^l = \{-1,-1/2,0,1/2,1\} \quad \gamma_i^l = \{-1,0,1\}$$

$$\theta_r = \sum_{i=0}^{n-1} \sigma_i^r \cdot \tan^{-1}(2^{-i}) = \sum_{i=0}^{n-1} \gamma_i^r \cdot \tan^{-1}(2^{-i}) \quad \sigma_i^r = \{-1,-1/2,0,1/2,1\} \quad \gamma_i^r = \{-1,0,1\}$$

We now describe the computation of the γ_i^l ; the computation of γ_i^r being identical (we will skip the superscript to simplify the notation).

We want to perform the computation on-line [ERCE84, IRWI87], in order to overlap it with the computation of the angles θ_s and θ_d, and with the rotation. To do this, we define the residual

$$z[j] = 2^j \left(\sum_{i=0}^{j+p} \sigma_i \cdot \tan^{-1}(2^{-i}) - \sum_{i=0}^{j} \gamma_i \cdot \tan^{-1}(2^{-i}) \right)$$

where p is the on-line delay. This results in the recurrence,

$$z[j+1] = 2(z[j] + \sigma_{j+p} \cdot 2^j \tan^{-1}(2^{-(j+p)}) - \gamma_j \cdot 2^j \tan^{-1}(2^{-j}))$$

with initial condition

$$z[0] = \sum_{i=0}^{p-1} \sigma_i \cdot 2^i \tan^{-1}(2^{-i})$$

To simplify the selection function, we decompose this recurrence into two by defining

$$w[j] = z[j] + \sigma_{j+p} \cdot 2^j \tan^{-1}(2^{-(j+p)})$$

so that

$$z[j+1] = 2(w[j] - \gamma_j \cdot 2^j \tan^{-1}(2^{-j}))$$

Note that the multiplication by 2^j is not achieved by shifting, rather the constants $2^j \tan^{-1}(2^{-j})$ are stored in the ROM (instead of $\tan^{-1}(2^{-j})$).

To use carry-save adders for these additions, it is necessary to perform the selection of γ using an estimate of w. In Appendix B we determine a selection function using two fractional bits of w and an on-line delay $p=2$. This function is

$$\gamma_i = \begin{cases} 1 & \textbf{if } \hat{w} \geq 1/2 \\ 0 & \textbf{if } -1/2 \leq \hat{w} \leq 1/4 \\ -1 & \textbf{if } \hat{w} \leq -3/4 \end{cases}$$

The implementation of this module and its timing is shown in Figure 9. Since the carry-save adders have a small delay, the delay of the ROMs might be the determining factor in the step time of this recurrence. To reduce this delay, the ROM can be replaced by a shift-register array.

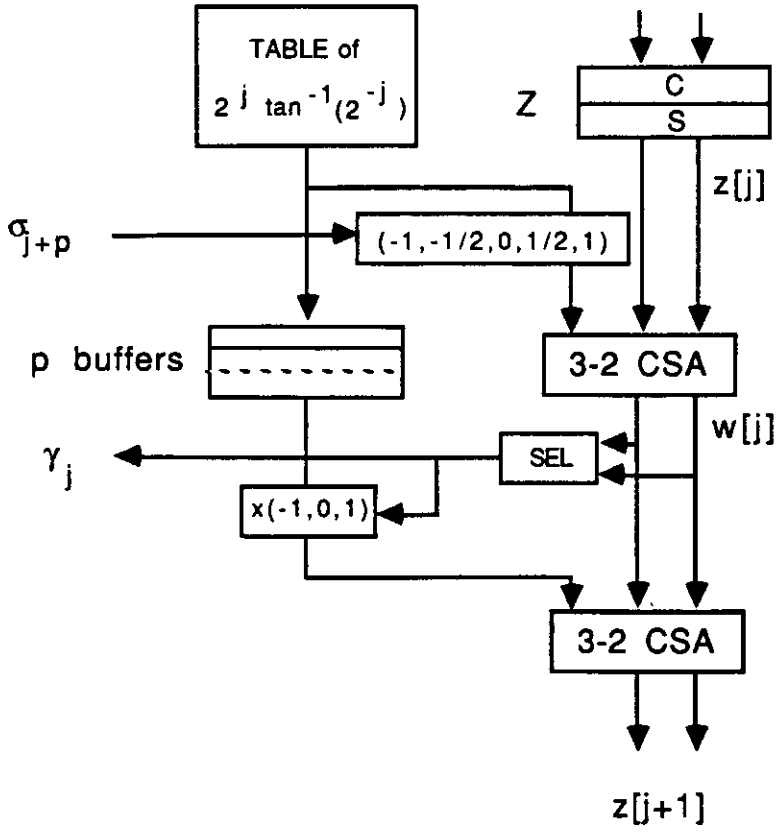
On-line two-sided rotation

The left-angle rotation is done by an on-line circular CORDIC operation, as discussed in Section 4.

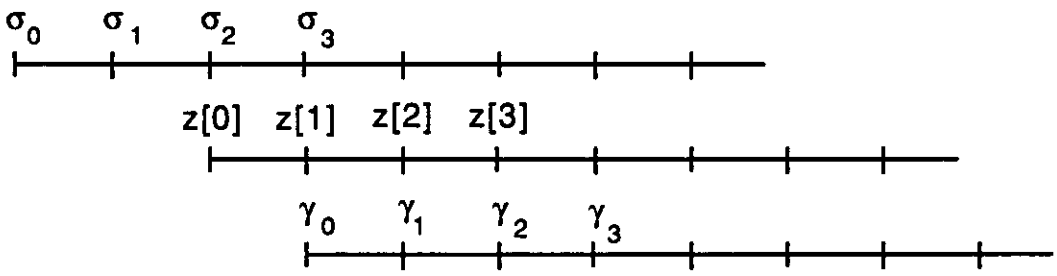
The right-angle rotation is also performed by an on-line circular CORDIC. The timing of Figure 10 shows that this right-angle rotation begins when all digits of the inputs have already entered the left-angle rotation unit; consequently, it is possible to use the same unit for both rotations.

Scale-factor correction

Each of the CORDIC rotations produces a modification of the magnitudes [WALT71] by the factor



(a)



(b)

Figure 9: Angle Computation for SVD
(a) - Implementation
(b) - Timing

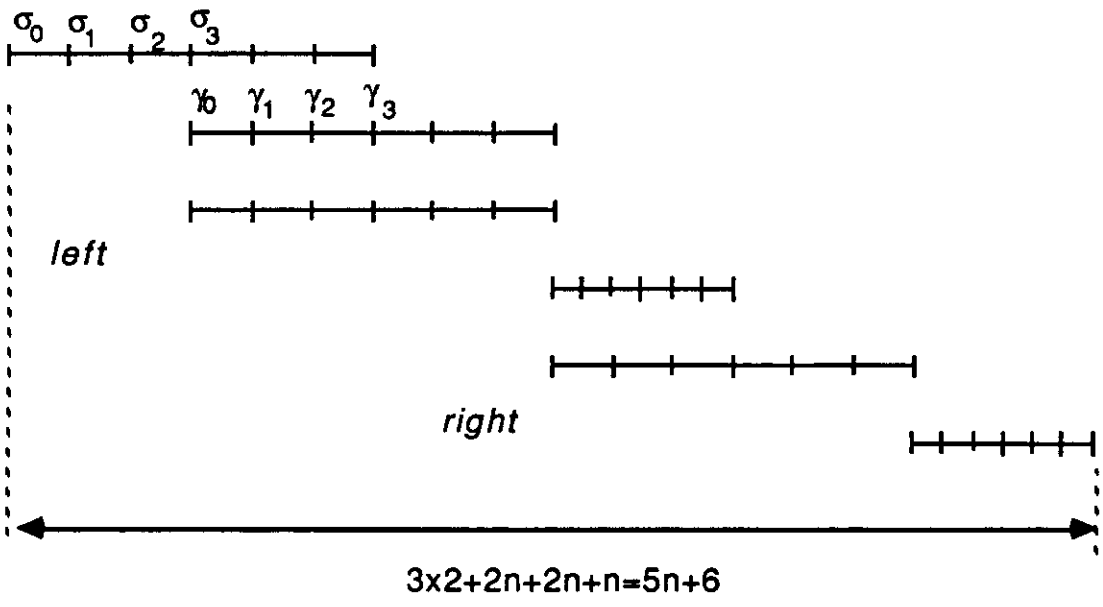


Figure 10: Timing of Two-Sided Rotation

$$K_l = \prod(1 + (\gamma_i^l)^2 2^{-2i})^{1/2}$$

and

$$K_r = \prod(1 + (\gamma_i^r)^2 2^{-2i})^{1/2}$$

It is necessary to correct for these factors. Instead of performing individual corrections, it is possible to perform just one correction using the factor

$$K = K_l \cdot K_r$$

In conventional CORDIC the factors are constant (independent of the actual values of the σ_i's) because the possible values of γ_i is the set {-1,1}. In contrast, in this case the digit sets of θ_l and θ_r are {-1,0,1}, so that the correction factors for each of these rotations are not constant and have to be computed for the specific angle value. From the definition,

$$K = K_l \cdot K_r = \prod(1 + (\gamma_i^l)^2 2^{-2i})^{1/2} \cdot \prod(1 + (\gamma_i^r)^2 2^{-2i})^{1/2}$$

There are several ways to calculate this scaling factor. We choose to follow the procedure discussed in Section 3. Since now there are $2n$ factors of the form $(1 + \sigma^2 \cdot 2^{-2i})$, the on-line implementation of $P = K^2$ would have $2n$ on-line adders and delays. However, in the same way as done for the two-sided rotation, it is possible to use twice the module discussed in Section 3, by feeding back the output.

As discussed in Section 3, an on-line square root unit is used to calculate $K = P^{1/2}$. Then on-line divisions perform the correction and the on-the-fly conversion to 2's complement representation [ERCE87c].

Overall SVD system

We now summarize the complete system. As shown in Figure 11, the diagonal processors contain the following components:

- a partial redundant CORDIC module to evaluate the angles θ_s and θ_d (in decomposed form). The main components of this module are a carry-save adder and a shifter. The module is pipelined with two stages, to compute both angles.

- an on-line module to compute the decomposition digits γ_j^l and γ_j^r of the angles θ_l and θ_r.

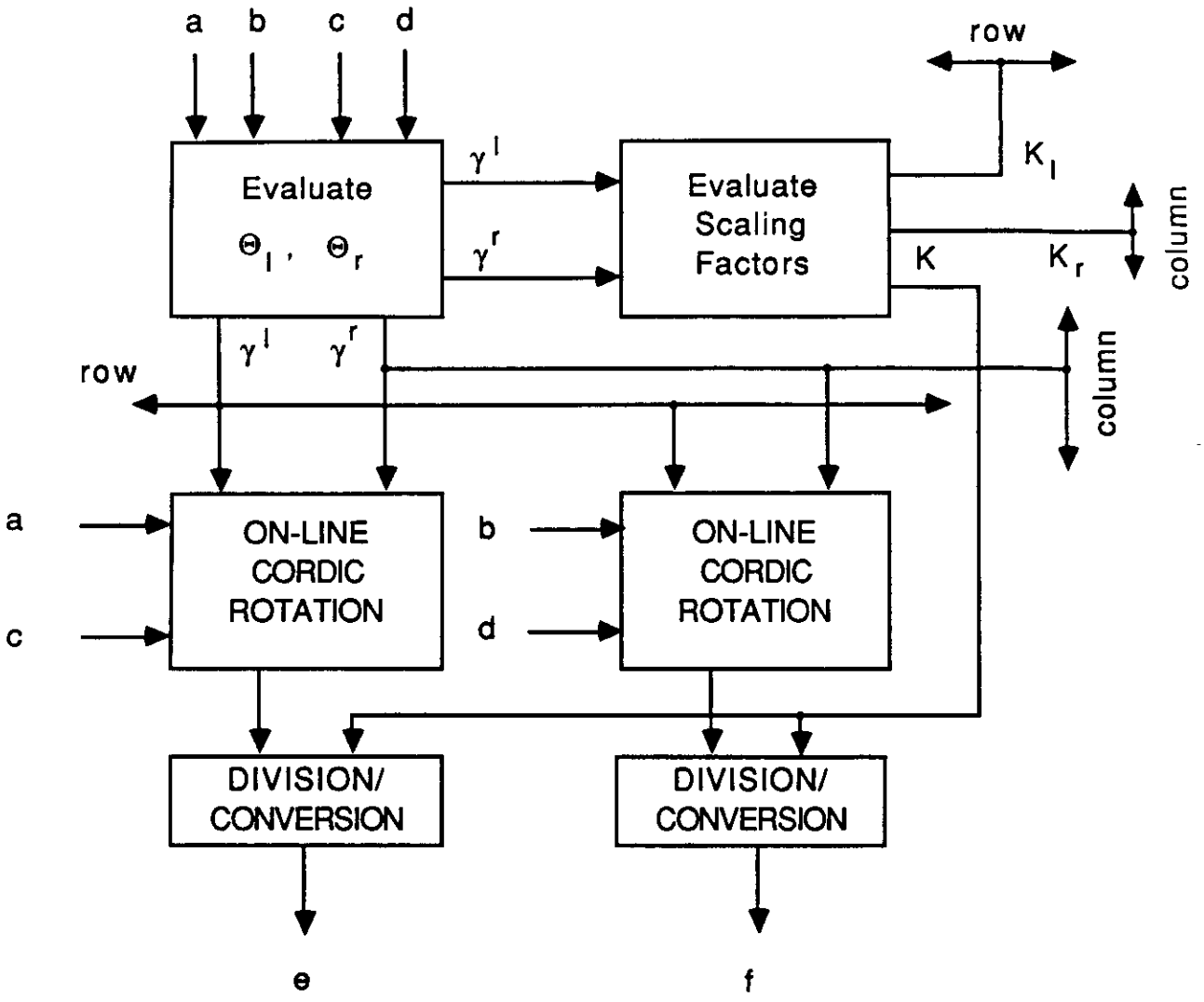


Figure11: Diagonal Processor Organization

- the on-line modules to compute K_l, K_r, and K.

- two partial on-line CORDIC modules to perform the rotations. Again, these modules do not require an angle recurrence, since the angle is produced in suitable decomposed form. The main components of these modules are n on-line adders and shift registers for delaying.

- an on-line multiplication module and two on-line division modules to perform the scaling correction. These dividers also convert to conventional representation.

The off-diagonal processors (Figure 12) contain the same rotation module as the diagonal processor, one multiplier, and four division modules.

Figure 13 shows the timing of the complete system. We estimate that the operation takes

$$T = 5n + 10 \quad clock\ cycles$$

In comparison, the implementation proposed in [CAVA87] takes $3.25nd$ cycles. For the typical value of d=6, the implementation proposed here is about 4 times faster. Moreover, it is 1.6 times faster than the nonredundant scheme proposed in [ERCE87b].

With respect to area, we cannot make a significant comparison without actual realization.

7. Conclusions

We have presented several modifications to the CORDIC method in order to improve speed and efficiency of its implementation. The main contributions are: (i) the introduction of redundant (carry-free) addition to replace time-consuming conventional additions; (ii) the use of on-line arithmetic to reduce the communication bandwidth, maximize the overlap between successive operations, and replace area-expensive shifters by delays; (iii) the use of angles in decomposed forms to eliminate angle accumulation recurrences. These modifications contribute to a speedup of about 4.5 with respect to a conventional CORDIC in the Givens' triangularization algorithms, and to a speedup of about 4 in the SVD case. No attempt has been made at this time to estimate the savings in the area since no VLSI realizations are done.

Acknowledgements This research has been supported in part by the ONR Contract N00014-85-K-0159 *On-Line Arithmetic Algorithms and Structures for VLSI.*

References

[AHME82a] H.M. Ahmed, J.M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," IEEE Computer, Vol.15, No. 1, Jan. 1982, pp. 65-82.

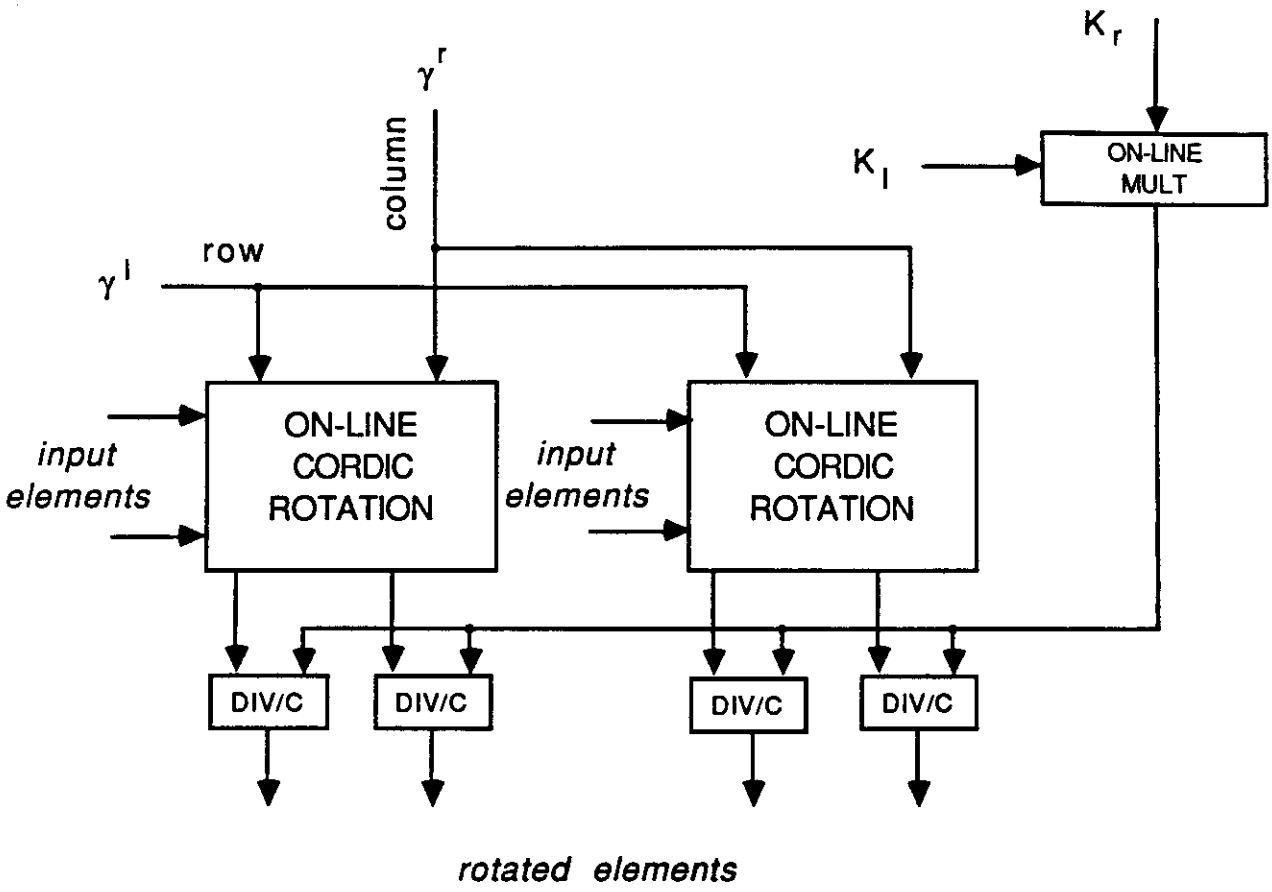


Figure 12: Off - Diagonal Processor Organization

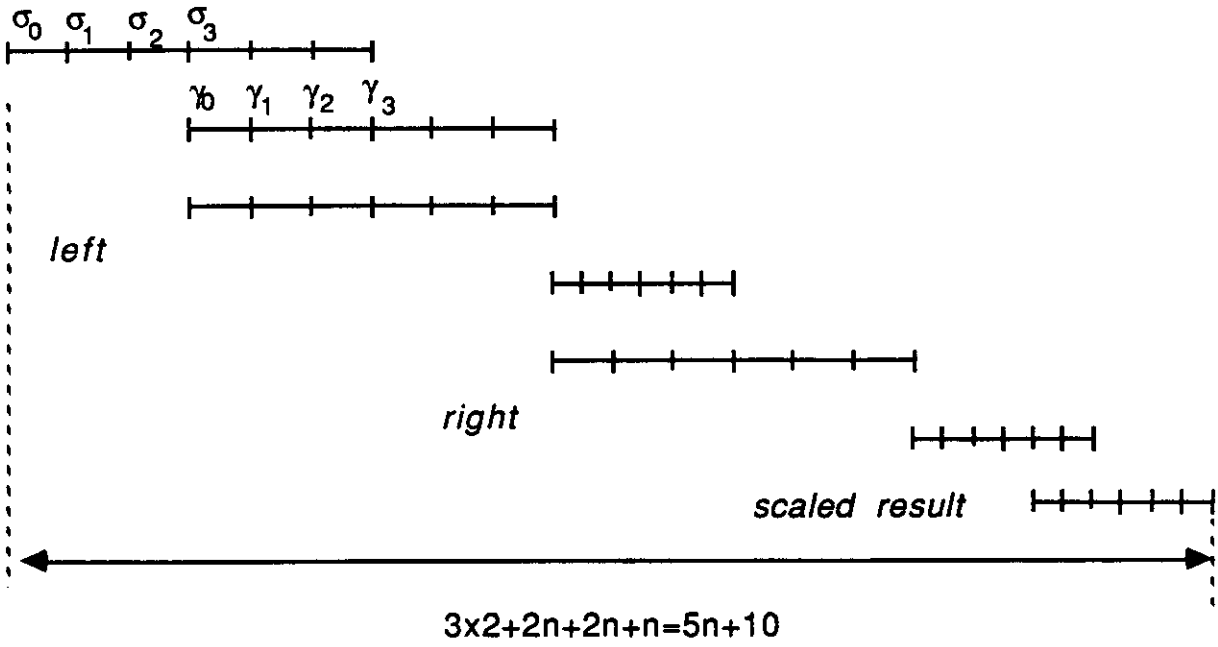


Figure 13: Timing of SVD

[AHME82b] H.M. Ahmed, "Signal Processing Algorithms and Architectures", Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, 1982.

[ATKI75] D. E. Atkins, "Introduction to the Role of Redundancy in Computer Arithmetic," Computer, June 1975, pp.74-75.

[AVIZ61] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," IEEE Trans. Elec. Computers, Vo. EC-10, September 1961, pp.389-400.

[BREN85a] R.P. Brent, F.T. Luk, and C.F. Van Loan, "Computation of the Singular Value Decomposition Using Mesh-Connected Processors," Journal of VLSI and Computer Systems, vol. 1, no. 3, pp.242-270, 1985.

[BREN85b] R.P. Brent and F.T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," SIAM J. Sci. Statist. Comput., vol. 6, pp. 69-84, 1985.

[CAVA87] J.R. Cavallaro and F.T. Luk, "CORDIC Arithmetic for an SVD Processor," Proc. 8th Symposium on Computer Arithmetic, pp. 113-120, 1987. [CIMI81] L. Ciminiera, A. Serra, and A. Valenzano, "Fast and Accurate Matrix Triangularization using an Iterative Array," Proceedings 5th. Symposium on Computer Arithmetic, 1981, pp. 215-221

[DELO83] J.M. Delosme, "VLSI Implementation of Rotations in Pseudo-Eucledian Spaces," Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 2, pp. 927-930.

[ERCE84] M.D. Ercegovac, "On-Line Arithmetic: An Overview," Proc. SPIE Real-Time Signal Processing, 495 (VII), pp. 86-93, August 1984.

[ERCE87a] M.D. Ercegovac and T. Lang, "On-Line Scheme for computing Rotation Factors," Proc. 8th Symposium on Computer Arithmetic, pp. 196-203, 1987.

[ERCE87b] M.D. Ercegovac and T. Lang, "On-Line Schemes for Computing Rotation Factors for SVD," Proc. SPIE 826, August 1987.

[ERCE87c] M.D. Ercegovac and T. Lang, "On-the Fly Coversion of Redundant into Conventional Representations," IEEE Transactions on Computers, vol. C-36, pp. 895-897, July 1987.

[GENT73] W. M. Gentleman, "Least Squares computations by Givens Transformations Without Square Roots," J. Institute Maths. Applics., Vol. 2, 1973, pp. 329-336.

[GENT81] W.M. Gentleman and H.T. Kung, "Matrix Triangularization by Systolic Arrays," Proc. SPIE: Real-Time Signal Processing IV (1981), pp. 19-26.

[GOLU83] G.H. Golub and C.F. Van Loan, "Matrix Computations," The John Hopkins University Press, Baltimore, 1983.

[IRWI87] M.J. Irwin and R.M. Owens, "Digit-Pipelined Arithmetic as Illustrated by the Paste-Up System: A Tutorial," IEEE Computer, April 1987, pp. 61-73.

[KUNI87] S. Kuninobu, et. al., "Design of High-Speed MOS Multiplier and Divider Using Redundant Representation," Proc. 8th Symposium on Computer Arithmetic, pp. 80-86, 1987.

[LUK86] F.T. Luk, "Architectures for Computing Eigenvalues and SVDs," Proc. SPIE Highly Parallel Signal Processing Architectures, vol. 614, 1986.

[METZ65] G. Metze, "Minimal Square Rooting," IEEE Trans. Elec. Computers, Vol.EC-14, No.2, 1965, pp.181-185.

[ROBE58] J.E. Robertson, "A New Class of Digital Division Methods," IEEE Trans. Elec. Computers, Vol. EC-7, September 1958, pp.218-222.

[SAME78] A. Sameh and D.J. Kuck, "On Stable Parallel Linear Solvers", JACM 25, No.1, pp.81-91, 1978.

[VOLD59] J. Volder, "The CORDIC Trigonometric Computing Technique," IRE Trans. Electronic Computers, EC-8, no. 3, pp. 330-334, Sept. 1959.

[WALT71] J.S. Walther, "A Unified Algorithm for Elememtary Functions," AFIPS Spring Joint Computer Conf., pp. 379-385, 1971.

Appendix A

We now develop the selection function for σ_j using the digit-set $\{-1,0,1\}$ and an estimate of $w[j]$. For this, we first obtain the selection intervals $[L_k,U_k]$ of $w[j] = 2^j y[j]$ so that $\sigma_j = k$ can be selected. The intervals for $k=\pm 1$ are the same as for the standard CORDIC [WALT71], that is,

$$L_1 = U_{-1} = 0$$

and

$$U_1[j] = -L_{-1}[j] = 2^j A$$

where

$$\tan^{-1}\left(\frac{A}{x[j]}\right) = \sum_{i=j+1}^{n-1} \tan^{-1}(2^{-i}) + \tan^{-1}(2^{-(n-1)})$$

It is also shown in [WALT71] that the value

$$|y[j+1]| = x[j]2^{-j},$$

which results in the standard CORDIC when $y[j] = 0$, is acceptable for convergence. Therefore, we obtain

$$U_1[j+1] = -L_{-1}[j+1] \geq 2x[j]$$

For the modified redundant CORDIC we need to determine U_0 and L_0. >From the previous expression, it is possible to select $\sigma_j = 0$ whenever the resulting

$$|w[j+1]| \leq 2x[j]$$

Consequently, from the recurrence

$$w[j+1] = 2(w[j] - \sigma_j x[j])$$

we obtain

$$U_0[j] = -L_0[j] = x[j]$$

Using these intervals we now determine a selection function that is suitable for using an estimate $\hat{w}[j]$ of $w[j]$. To be able to do this, it is necessary to have an overlap between the intervals for -1 and 0 and for 0 and 1. This can be achieved by normalizing $x[j]$, that is making

$$x[j] \geq 1/2$$

>From the description of the operation for floating-point representation, it can be seen that it is possible to have $x[1]$ normalized. Moreover, from the recurrence for $x[j]$ we can deduce that

$$x[j+1] = x[j] + \sigma_j w[j]2^{-2j} \geq x[j]$$

Consequently,

$$x[j] \geq x[1] \geq 1/2 \quad \text{for } j \geq 1$$

We now obtain a suitable selection function. Since the actual relation between $\hat{w}$ and w depends whether carry-save or signed-digit redundant addition is used, we treat these two cases separately.

i) Carry-save case

In this case the values are represented in 2's complement, so that the relation between $\hat{w}$ and w is

$$w[j] - 2^{-t+1} \le \hat{w}[j] \le w[j]$$

where the estimate is computed by assimilating t fractional bits of w.

If $M(k)$ $[m(k)]$ is the largest [smallest] value of $\hat{w}[j]$ for which a quotient value of k is chosen, we have

$$m(k) \le \hat{w} \le M(k) \;\rightarrow\; \sigma_j = k$$

$$m(k+1) = M(k) + 2^{-t} \quad (\textit{since } t \textit{ fractional bits are used in selection})$$

$$m(k) \ge L_k; \quad M(k) \le U_k - 2^{-t+1}$$

These relations are illustrated in Figure A1. Introducing

$$U_0 = -L_0 = 1/2 \quad (\textit{to make the selection independent of } x[j])$$

and

$$L_1 = U_{-1} = 0$$

results in the following inequalities:

$$M(0) \le 1/2 - 2^{-t+1}$$

$$M(0) \ge -2^{-t} \quad (\textit{since } m(1) = M(0) + 2^{-t} \ge 0)$$

$$m(0) \ge -1/2$$

$$m(0) \le -2^{-t} \quad (\textit{since } M(-1) = m(0) - 2^{-t} \le -2^{-t+1})$$

This results in

$$max(-1/2, -2^{-t}) \le M(0) \le 1/2 - 2^{-t+1}$$

$$-1/2 \le m(0) \le -2^{-t}$$

These expressions are satisfied for $t=1$ and $M(0) = m(0) = -1/2$. The resulting selection function is

$$\sigma_i = \begin{cases} 1 & \textbf{if} \quad \hat{w} \geq 0 \\ 0 & \textbf{if} \quad \hat{w} = -1/2 \\ -1 & \textbf{if} \quad \hat{w} \leq -1 \end{cases}$$

ii) Signed-digit case

In this case, the relation between $\hat{w}$ and w is

$$w[j] - 2^{-t} \leq \hat{w}[j] \leq w[j] + 2^{-t}$$

where $\hat{w}$ is computed using t fractional bits of w.

Using the same notation as in case i) we get,

$$m(k) \leq \hat{w} \leq M(k) \quad \rightarrow \quad \sigma_j = k$$

$$m(k+1) = M(k) + 2^{-t} \quad (\textit{since } t \textit{ fractional bits are used in selection})$$

$$m(k) \geq L_k + 2^{-t}; \quad M(k) \leq U_k - 2^{-t}$$

These relations are illustrated in Figure A1.b. Again, as in case i) introducing

$$U_0 = -L_0 = 1/2 \quad (\textit{to make the selection independent of } x[j])$$

and

$$L_1 = U_{-1} = 0$$

results in the following inequalities:

$$M(0) \leq 1/2 - 2^{-t}$$

$$M(0) \geq -2^{-t} \quad (\textit{since } m(1) = M(0) + 2^{-t} \geq 0)$$

$$m(0) \geq -1/2 + 2^{-t}$$

$$m(0) \leq -2^{-t} \quad (\textit{since } M(-1) = m(0) - 2^{-t} \leq -2^{-t+1})$$

These expressions are satisfied for $t=1$ and $M(0)=m(0)=0$. The resulting selection function is

$$\sigma_i = \begin{cases} 1 & \textbf{if} \quad \hat{w} \geq 1/2 \\ 0 & \textbf{if} \quad \hat{w} = 0 \\ -1 & \textbf{if} \quad \hat{w} \leq -1/2 \end{cases}$$

Appendix B

We determine now the selection function of the on-line determination of the component γ_j of θ_l. The corresponding recurrences are

$$z[j+1] = 2(w[j] - \gamma_j 2^j \tan^{-1}(2^{-j}))$$

$$w[j] = z[j] + \sigma_{j+p} 2^j \tan^{-1}(2^{-(j+p)})$$

We now determine the intervals $[L_k, U_k]$ of $w[j]$ so that $z[j+1]$ remains bounded when chosing $\gamma_j = k$. From the recurences we obtain,

$$U_1 - 2^j \tan^{-1}(2^{-(j+p)}) = 2(U_k - k \cdot 2^j \tan^{-1}(2^{-j}))$$

$$-U_1 + 2^j \tan^{-1}(2^{-j+p)}) = 2(L_k - k \cdot 2^j \tan^{-1}(2^{-j}))$$

Making $k=1$ in the first we get,

$$U_1 = 2 \cdot 2^j \tan^{-1}(2^{-j}) - 2^j \tan^{-1}(2^{-(j+p)})$$

Consequently,

$$L_1 = 2^j \tan^{-1}(2^{-(j+p)})$$

$$U_0 = 2^j \tan^{-1}(2^{-j}) - 2^j \tan^{-1}(2^{-(j+p)})$$

$$L_0 = -U_0$$

$$U_{-1} = -L_1$$

$$L_{-1} = -U_1$$

Since these intervals depend on j and we want a selection function independent of j, we determine the corresponding bounds. We get

$$L_1 \leq 2^{-p}$$

$$U_0 \geq \pi/4 - \tan^{-1}(2^{-p})$$

To get a positive overlap ($U_0 > L_1$) we need $p \geq 2$. Using $p=2$, we get

$$L_1 \leq 2^{-2}$$

$$U_0 \geq 2^{-1}$$

Since the overlap is of 2^{-2}, it is possible to use an estimate of w with two fractional bits. A suitable selection function is

$$\gamma_j = \begin{cases} 1 & \textbf{if } \hat{w} \geq 1/2 \\ 0 & \textbf{if } -1/2 \leq \hat{w} \leq 1/4 \\ -1 & \textbf{if } \hat{w} \leq -3/4 \end{cases}$$