ON-LINE SCHEMES FOR COMPUTING ROTATION ANGLES FOR SVDS

Milos D. Ercegovac
Tomas Lang

August 1987
CSD-870043

On-Line Schemes for Computing Rotation Angles for SVDs

Miloš D. Ercegovac and Tomas Lang

UCLA Computer Science Department
University of California
Los Angeles, CA90024

Abstract

Two floating-point radix-2 schemes using on-line arithmetic for implementing the direct two-angle method for SVDs are presented. The first scheme is an on-line variant of the cosine/sine approach and is the fastest of the schemes considered: it performs the 2×2 SVD step in about $2n$ clock cycles. However, it requires a relatively large number of modules; this number is reduced when some modules are reused, resulting in a time of $3n$ clock cycles. The number of modules of this on-line version is still larger than that of the conventional one, but this is compensated by the smaller number of bit-slices per module and by the digit-serial communication among modules. The corresponding speed-up ratios are of 5 and 3 with respect to a conventional arithmetic implementation. The second scheme uses an on-line CORDIC approach and performs the 2×2 SVD in about $7n$ clock cycles and is advantageous because it is more time-area efficient. It results in a speed-up of about 2.5 with respect to the conventional CORDIC implementation.

1. Introduction

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 exists in using parallel arrays of processing elements, as discussed in [BREN85a], [BREN85b], [LUK86], and [CAVA87]. The primitive operation required for the parallel computation 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

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

Several methods can be used to compute the angles for these rotations, in particular the two-step method and the direct two-angle method [BREN85]. Here we will use the latter since it results in a more effective implementation when using on-line schemes. Briefly, this direct two-angle method consists in first finding the angles θ_s (sum) and θ_d (difference) as

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

Then the angles θ_l and θ_r are obtained as

$$\theta_l = \frac{\theta_s - \theta_d}{2} \quad \theta_r = \frac{\theta_s + \theta_d}{2}$$

For the realization of this method the following two approaches are possible:

a) The values of $\cos\theta$ and $\sin\theta$ are computed by a sequence of primitive operations involving squaring, addition, multiplication, division, and square root. The rotation is then done by several multiplications and additions. The main advantage of this approach is that efficient implementations for the primitive operations are known and that redundancy can be used to improve their speed. However, it requires various different modules and consists of several dependent computations. Examples of this approach are presented in [BREN85].

b) The CORDIC procedure [VOLD59, WALT71] is used for direct computation of the angles and of the rotations. The advantage of this approach is that a small number of operations is required and that the same module can be used for both the angle calculations and the rotation. However, the conventional implementation of the CORDIC module has two disadvantages: it is slow, because it involves recurrences that include carry-propagate addition, and area-consuming because of the need for variable shifters. An implementation of the direct two-angle method using the CORDIC procedure is presented in [CAVA87].

In this paper, we explore the two before-mentioned approaches with the objective of using on-line techniques [ERCE84] to improve the speed and reduce the area. Section 2 considers the sin/cos-based implementation and Section 3 presents the CORDIC-based implementation. Section 4 makes comparisons between these methods.

To make the speed comparisons meaningful a suitable measure has to be used. In some studies the comparisons are done in terms of the number of addition-like steps, which appear as basic components in the iterations for operations such as multiplication, division, and CORDIC. However, this is not an adequate measure since the time for addition depends on the type of addition performed. More specifically, the time of carry-propagate addition is several times larger

than that of redundant (carry-save or signed-digit) addition. It might be claimed that this does not change the validity of measuring in terms of additions, since for a particular implementation the corresponding type of addition would be used. However, this is not correct since not all algorithms can be directly transformed from one using carry-propagate addition into one using redundant additions. Furthermore, when fast redundant additions are used, other terms which were neglected when carry-propagate additions are considered become important. As a consequence, to make more meaningful comparisons, we define a basic clock cycle and estimate the time of the various operations in terms of this clock cycle.

The implementations we describe are for floating-point representations since this format provides better numerical characteristics and results in a system which is easier to use in a variety of environments than the fixed-point alternative. We use the characteristics of the algorithm to reduce the additional overhead introduced by the floating-point representation.

2. On-line implementation of the sin/cos-based approach

We now present an on-line implementation using the sin/cos-based method. As described in [BREN85] the algorithm is as follows:

$p_1 = d - a \quad q_1 = c + b$

{ Compute $c_s = \cos(\frac{\theta_s}{2})$ and $s_s = \sin(\frac{\theta_s}{2})$ }

if $|p_1| \le \varepsilon |q_1|$ **then**
begin
$c_s = 1;$
$s_s = 0$
end
else

begin

$\rho_1 = \frac{p_1}{q_1};$

$t_1 = \frac{1}{|\rho_1| + (1 + \rho_1^2)^{1/2}};$

$c_s = \frac{1}{(1 + t_1^2)^{1/2}};$

$s_s = sign(\rho_1) \cdot t_1 \cdot c_s$

end

$p_2 = a + d \quad q_2 = c - b$

{ Compute $c_d = \cos(\frac{\theta_d}{2})$ and $s_d = \sin(\frac{\theta_d}{2})$ }

if $|p_2| \leq \varepsilon |q_2|$ **then**

begin
$c_d = 1;$
$s_d = 0$
end

else

begin

$$\rho_2 = \frac{p_2}{q_2};$$

$$t_2 = \frac{1}{|\rho_2| + (1 + \rho_2^2)^{1/2}};$$

$$c_d = \frac{1}{(1 + t_2^2)^{1/2}};$$

$$s_d = sign(\rho_2) \cdot t_2 \cdot c_d$$

end

{ Compute $c_l = \cos\theta_l$, $s_l = \sin\theta_l$, $c_r = \cos\theta_r$ and $s_r = \sin\theta_r$ }

$$c_l = c_s c_d + s_s s_d \quad c_r = c_s c_d - s_s s_d$$
$$s_l = s_s c_d - c_s s_d \quad s_r = s_s c_d + c_s s_d$$

We now present an implementation of this scheme using on-line algorithms for the primitive operations. Briefly, the on-line approach has the following characteristics [ERCE84]:

i) The operands are received in a digit-serial fashion, from most significant to least significant.

ii) The result is obtained in the same digit-serial way, with the most-significant digit being produced after δ digits of the operands have been entered. δ is called the on-line delay of the operation, and it ranges from one for addition to four for square root.

iii) Because of this on-line operation, a sequence of dependent operations can be performed in an overlapped fashion, reducing significantly the overall delay.

iv) The digit-serial application of the operands reduces the communication bandwidth between operations.

v) For a precision of n bits, only about $n/2$ bit slices are needed to implement the on-line algorithms.

Properties iii) and iv) make this approach attractive for a complex sequence of dependent operations, such as that appearing in the two-step SVD algorithm. Previously we have applied this technique to the triangularization of a matrix using Givens' rotation [ERCE87a], where details of the floating-point on-line algorithms are given.

Figure 1 shows a block diagram of the implementation. Since the primitive operations required are similar to those in the triangularization case, we do not repeat here the detailed on-line algorithms and implementations. Moreover, the exponent calculations and the alignments are performed in a manner similar to [ERCE87a] so we do not show their details here.

From the triangularization implementation, we obtain the on-line delays, as shown in Figure 1. Since the computations for (c_s, s_s) and (c_d, s_d) are exactly the same, except for the initial sums/differences, two alternative implementations exists:

Alternative A: Use two copies of the hardware in Section C (Figure 1) and compute both (c_s, s_s) and (c_d, s_d) in parallel.

Alternative B: Use only one copy of Section C and compute (c_d, s_d) after (c_s, s_s). Note that the second use can begin as soon as the data for the first computation has completely entered the unit.

The first alternative leads to a faster implementation but uses more hardware. The total delay and the number of units used for both cases are shown in the following table (for n-bit mantissas). We also include the values for a conventional implementation, as discussed in [CAVA87].

	Alternat. A	Alternat. B	Conventional
Time	$t_{fadd} + n + 25$ $\approx 2n^*$	$t_{fadd} + 2n + 25$ $\approx 3n^*$	$10n$
Dividers	6	3	1
Multipliers	6	5	1
Square	4	2	1
Square-root	4	2	1
Adders	-	-	4

* We assume a single-precision format with 24 bits in the mantissa.

The total delay of the on-line implementation compares favorably with that of the conventional approach. The corresponding speed-up ratios are

$$\frac{T_{conv}}{T_{on-line}} \approx 5 \quad (Alternative\ A)$$

$$\frac{T_{conv}}{T_{on-line}} \approx 3.3 \quad (Alternative\ B)$$

This assumes that the clock cycle is the same for both implementations. This is approximately right, if both implementations use redundant addition, which is a necessity in the on-line case and a possibility in the conventional case.

The number of modules required in the on-line case is larger than that for the conventional implementation. However, this is compensated by the fact that the number of bit-slices per module in the on-line case is roughly one half of that for the conventional implementation and by the digit-serial communication among modules.

In addition to the calculation of the rotation matrices, performed by the diagonal processors, it is necessary to do the two-sided rotations using the off-diagonal processors. In this cos/sin-based approach these rotations are performed by on-line multiplications and additions in an overlapped manner with the operations in the diagonal processors.

3. On-line CORDIC implementation

The CORDIC implementation of the direct two-angle method is described in [CAVA87]. Here we adapt this scheme to an on-line approach, with significant improvement in speed.

The algorithm described in Figure 2 [CAVA87], first computes, by means of two concurrent CORDIC circular operations, the angles

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

Then, the two angles θ_l and θ_r are obtained as

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

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

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.

Here we present a modification of this implementation. The most significant differences introduced are the following:

i) The addition/subtraction (and division by 2) of the angles in step 3 are done using the decomposed representation produced by the CORDIC 1 and CORDIC 2 steps. This has the following three important advantages:

- First, it reduces the area required by all CORDIC modules, because none of them will require the angle recurrence. That is, we implement only the x, y recurrences. Consequently, the implementation is simpler and no ROM modules are needed.

- Second, it produces the angles θ_l and θ_r in an on-line fashion, which permits the overlapping of CORDIC 1 and 2 with CORDIC 3 and results in a significant reduction in the overall delay.

- Third, the communication between the angle processor and the rotation processors is digit serial, with the corresponding reduction in bandwidth.

However, as we will discuss in the next section, this decision of using the decomposed angles eliminates the possibility of using an on-line version for CORDIC 1 and 2.

ii) The CORDIC 3 and CORDIC 4 operations are implemented on-line. This allows for the overlapping of these operations, reduces the step time to the on-line delay of addition (instead of a carry-propagate adder delay), and eliminates the need for area-consuming shifters.

iii) The scale-factor correction is done by on-line division. This permits the overlap of this step with the CORDIC steps. The on-line division module also incorporates the on-the-fly conversion from redundant into conventional representation [ERCE87c].

We now present the implementation of each of the modules and then discuss the overall scheme.

Computation of Rotation Angles

As indicated in Figure 2, the rotation angles are computed by first determining the angles θ_s and θ_d. To do this, first perform in parallel four addition/subtractions and then two concurrent circular CORDIC operations.

The circular CORDIC operations have the form

$$x[i+1] = x[i] + \sigma_i 2^{-i} y[i]$$

$$y[i+1] = y[i] - \sigma_i 2^{-i} x[i]$$

with the selection of σ_i given by

$$\sigma_i = \begin{cases} 1 & \textbf{if } y[i] \geq 0 \\ -1 & \textbf{if } y[i] < 0 \end{cases}$$

To compute both angles, two concurrent CORDIC operations are performed, with subscripts s and d. The initial conditions are

$$x_s[0] = c + b \quad y_s[0] = d - a$$

$$x_d[0] = c - b \quad y_d[0] = d + a$$

The resulting angles are

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

In a complete CORDIC module the summation of the angle is computed by another recurrence of the form

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

where the angles $\tan^{-1}(2^{-i})$ are obtained from a ROM. In contrast to this, we do not compute the angles but perform the operations of Step 3 on each of the σ_i components. That is, we obtain the σ_i components of θ_l and θ_r from the following relations:

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

This results in the following table:

Table 1

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

These resulting $\sigma_i's$ are transmitted on-line to the rotation modules.

As described before, the recurrences x and y require the use of one shifter each. Since a shifter consumes a significant area, we transform the recurrences to reduce the number of shifters to one. We replace $y[i]$ by $w[i]$ such that

$$w[i] = 2^i y[i]$$

This results in

$$x[i+1] = x[i] + \sigma_i w[i] 2^{-2i}$$

$$w[i+1] = 2(w[i] - \sigma_i x[i])$$

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

This version of the recurrences requires only one shifter (for x). In addition, to reduce the hardware required, we pipeline the recurrence step and use one module for the computation of both angles θ_s and θ_d. This pipelined structure is also used to perform the initial addition/subtractions. The corresponding implementation and timing is shown in Figure 3. The time of one iteration of the recurrence depends essentially on the time for carry-propagate addition plus the time for shifting. Since this is relatively slow (compared to the times for operations in the rotation, as we will see in the next section), we assume that one iteration takes d clock cycles and is partitioned into two stages of $d/2$ clock cycles each.

Note that to obtain a fast recurrence step it would be convenient to implement the CORDIC operation using a redundant addition - to eliminate the carry-propagate adder. Moreover, an on-line approach eliminates the slow and area-consuming shifters. An implementation of this type is described in [ERCE87b] for the use of matrix triangularization. Unfortunately, this approach cannot be used for the CORDIC 1 and CORDIC 2 operations considered here because the redundant implementation produces an angle decomposition with digit set {-1,0,1} [ERCE87b], which would not allow us to perform Step 3 directly using the decomposed angles. The reason for this can be seen from the case in which the corresponding components of θ_s and θ_d are 1 and 0, respectively; in such a case the component of θ_r is 1/2 which cannot be used in the rotation step. Consequently, since we see many advantages to performing the calculation using the decomposed angles, we decided to use a non-redundant addition for the recurrence.

On-line left-angle rotation

The left-angle rotation is done by an on-line circular CORDIC operation. It has the following characteristics:

i) Since the angle θ_l for the rotation is known in decomposed form, it is only necessary to perform the x and y recurrences. This reduces the amount of hardware (i.e., no angle recurrence modules, no ROMs).

ii) The rotation can be overlapped with the calculation of the angle, because the angle is produced in decomposed form in an on-line fashion. This reduces the overall time.

iii) The CORDIC does not involve a sign detection (since the angle is known in decomposed form). Consequently, an implementation using a redundant addition is straightforward. In particular, it is simple to use an on-line implementation with the following advantages [ERCE87b]:

- the area-consuming shifters are replaced by area-efficient delays,

- the results are obtained on-line, which makes it possible to overlap the left-angle rotation with the right-angle rotation and with the scale correction, reducing significantly the overall delay.

Note that in this case we do not obtain the basic advantage of using the redundant-addition approach, namely of reducing the iteration time. This is because the initiation of each iteration has to wait for the corresponding component of the angle. In other words, the iteration time is determined by the CORDIC 1 and 2 steps, which do not use a redundant addition.

We now describe the on-line scheme [ERCE87b]. The rotation corresponds to the recurrence

$$x[i+1] = x[i] + \sigma_i^l 2^{-i} y[i]$$

$$y[i+1] = y[i] - \sigma_i^l 2^{-i} x[i]$$

Note that in this case the digit-set of σ_i is {-1,0,1} (as produced by the angle module), in contrast with conventional CORDIC in which the values are {-1,1}. This does not pose any problem in the implementation.

To replace the area-consuming shifters by more efficient delays, we implement the recurrences with on-line additions. To do this, the n iterations are unfolded and overlapped as shown in Figure 4a. As a result, the multiplication by 2^{-i}, which in conventional CORDIC is implemented by a shifter, corresponds here to a delay.

The results are obtained on-line. The on-line delay corresponds to $n\tau$, where τ is the interval between the initiation of two consecutive iterations. Since the direction of rotation (σ_i^l) is obtained in an overlapped manner with the rotation, in a simplistic implementation τ is the maximum of the delay to produce one digit σ_i^l and the on-line delay of addition. Because of the need to use a carry-propagate adder to produce σ_i^l, the corresponding delay dominates. However, we can reduce τ by observing that the operand requiring the knowledge of σ_i^l is multiplied by 2^{-i}, that is, its i most significant digits are 0. Consequently, it is not necessary to know the value of σ_i^l until the i-th digit. Therefore,

$$t_i + i = i \times d$$

where t_i is the initiation time of the i-th iteration and d is the delay of one step of computation of σ_i^l (all measured in number of clock cycles). This results in

$$t_i = i(d-1)$$

That is, the time between initiations is

$$\tau = d-1 \quad \textit{clock cycles}$$

Therefore, the on-line delay of the left-angle rotation is $(d-1)n$ clock cycles.

After the last iteration is initiated, one digit of the result is obtained each clock cycle. To achieve this, it is necessary to input one digit of the operands per clock cycle; since consecutive initiations are separated by $d-1$ clocks, this requires buffers of length $d-1$ between the on-line adders (Figure 4a). The timing of the angle CORDIC and the left-angle rotation is shown in Figure 4b.

On-line right-angle rotation

The right-angle rotation is also performed by a on-line circular CORDIC. However, in this case the angle is known beforehand (in decomposed form). This makes it possible to use the potential of the redundant addition implementation of having an iteration cycle of one clock. To achieve this, it is necessary to minimize the on-line delay of the additions; therefore, radix-4 additions are performed. The timing diagram of the angle CORDIC and both rotations is shown in Figure 5.

Since, as shown in the diagram, the right-angle rotation begins when all the digits of the inputs have already entered the left-angle rotation unit, it is possible to use the same unit for both rotations, feeding back the results of th first rotation as inputs for the second. Note that, since in the right-angle rotation initiations occur every cycle, it is necessary to bypass the buffers of length $(d-1)$ between the on-line adders.

Scale-factor correction

The CORDIC rotations produce a modification of the magnitudes [WALT71] by the factor

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

and

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

It is necessary to correct for these factors. Instead of performing an individual correction, 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 $\sigma_i's$) because the possible values of σ_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 each value of the angle. Moreover, 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. The diagonal processors compute the values of K_l and K_r in an on-line fashion and send them to the rotation processors, where an on-line multiplication unit produces the correction factor K.

The computation of the left correction factor (the right one is done in an identical manner) is done by the following on-line scheme. The algorithm has two steps:

i) Compute

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

ii) Compute $K_l = P_l^{1/2}$. The computation of P_l is done by the recurrence

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

with $P[0] = 1$ and $P_l = P[n] = P[n/2]$ to the implementation precision.

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

The computation of $K_l = P_l^{1/2}$ is done by an on-line square-root algorithm [ERCE78].

The scaling (division by the correction factor) is done by two concurrent on-line division units. Since the divisor is a constant (off-line), this division is very similar to a SRT radix-2 implementation. The on-line division unit also performs the on-the-fly conversion from signed-digit representation to 2's complement representation [ERCE87c].

Overall SVD system

We now summarize the complete system. As shown in Figure 6a, the diagonal processors contain the following components:

- a partial CORDIC module to evaluate the angles θ_l and θ_r (by means of the angles θ_s and θ_d). This CORDIC module does not contain an angle recurrence since the angle is used by the rest of the system in decomposed form. The main components are a carry-propagate adder and a shifter. The module is pipelined with two stages, to compute both angles. The step time is relatively slow since it is determined by the carry-propagate adder and by the shifter. To interface with other faster modules, we make this step time equal to d clock cycles (typically from 4 to 6 cycles).

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

- two partial on-line CORDIC module to perform the rotations of the 2x2 matrix. Again, this module does not require an angle recurrence, since the angle is produced in suitable decomposed form by the first CORDIC module. The main components of each module are n on-line adders and shift registers for buffering and delaying.

- an on-line multiplication unit and two on-line division units to correct for the scaling factor. These dividers also convert to conventional representation.

The off-diagonal processor, shown in Figure 6b, contains two CORDIC on-line rotation module, one multiplier, and four division units.

Figure 6c shows the timing of the complete system. We estimate that the operation takes

$$T = n(d+1) + 5 \quad clock\ cycles$$

In comparison, the implementation proposed in [CAVA87] takes $3.25nd$ cycles. For $d \approx 4$, the implementation proposed here is about 2.6 times faster. With respect to area, we cannot make a significant comparison without actual realization. However, we can point to the following differences:

i) we eliminate the need for the angle recurrence in the CORDIC modules.

ii) we use digit-serial communication between modules.

iii) we pipeline the first CORDIC module to be used for the computation of both angles

iv) we add a partial on-line CORDIC module to the diagonal processor (to perform the rotation). We also replace the standard CORDIC module by a partial on-line CORDIC for the off-diagonal processor.

v) we add the units to compute the scaling factors and division units for the scaling.

Floating-point representation

We now consider the modifications required for the described implementation when floating-point representations are used. In [AHME82b] floating-point CORDIC is described. However, it uses floating-point adders to implement the recurrences. This is not attractive here because of the alignment requirements. We now develop a simpler alternative adapted to the SVD case.

i) Computation of Angles

Assume that the initial values in the 2x2 matrix are represented in normalized floating point. Then, as shown in Figure 1, the floating-point additions/subtractions produce

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

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

The first step of the circular CORDIC results in

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

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

These can be written as

$$x[1] = A^+ \cdot 2^a \qquad w[1] = A^- \cdot 2^a$$

with one of them normalized.

This indicates that the CORDIC iterations can be done by performing the first step using floating-point addition/subtraction to obtain A^+, A^-, and a. After that, 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

$$|\theta_{\min}| = \tan^{-1}(2^{-(n-1)}) \approx 2^{-(n-1)},$$

$$|\theta_{\max}| = \sum_{i=0}^{n-1} \tan^{-1}(2^{-i})$$

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. The angles θ_l and θ_r are computed as before.

ii) Rotation

The modifications to the rotation step are similar to those for the calculation of the angle. Let the initial values be

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

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

Again, a floating-point addition and a subtraction produces

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

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

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

4. Comparisons and Conclusions

We have presented two schemes for the use of on-line techniques in the implementation of the direct two-angle method for SVD; the first is a modification of the cos/sin-based approach, while the second uses the CORDIC approach. Table 2 gives execution-time comparisons between the conventional and our on-line implementations. As discussed before, the time-measure used is number of clock cycles, since this is a suitable standard for accurate comparisons.

Table 2: Time comparisons
(clock cycles)

	Conv. cos/sin	On-line cos/sin	Conv. CORDIC	On-line CORDIC
Angles	$10n$	$2n$ $3n$	nd	nd
Two-sided rotation	$2n$	overlapped	$2.25nd$	$n+5$ (overlapped)
TOTAL	$12n$	$2n$ $3n$	$3.25nd$	$(d+1)n + 5$
$d=4$	-	-	$13n$	$5n$
$d=6$	-	-	$19n$	$7n$

From the table we draw the following conclusions:

- The use of on-line techniques produces a speed improvement in both approaches.

- The fastest approach is the on-line cos/sin-based, but this is also the one requiring the largest number of modules. However, this is compensated by the fewer bit slices per module and by the digit-serial communication.

- The conventional CORDIC-based approach is the slowest, because of the need of carry-propagate adders. However, it requires the smallest number of modules.

• The on-line CORDIC-based approach seems a good scheme to implement because it probably has the best time-area characteristics.

As mentioned in Section 3, the on-line CORDIC scheme can still be made faster by the use of redundant addition in the recurrences. This approach has the potential of achieving an execution time approximately of $2n$ clock cycles, with an amount of hardware not significantly larger than that of the non-redundant case. We are developing the details of such an implementation.

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

[AHME82] H.M. Ahmed, "Signal Processing Algorithms and Architectures," Ph.D. Dissertation, Dept. of Electrical Engineering, Stanford University, June 1982.

[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.

[DELO83] J.M. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces", IEEE Int. Conf. Acoustics, Speech and Signal Processing, 2, pp. 927-930, April 1983.

[ERCE78] M.D. Ercegovac, "An On-Line Square Root Algorithm", Proc. of the 4th IEEE Symposium on Computer Arithmetic, pp. 183-189, 1978.

[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, "Redundant On-Line CORDIC Algorithms", in preparation.

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

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

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

[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.

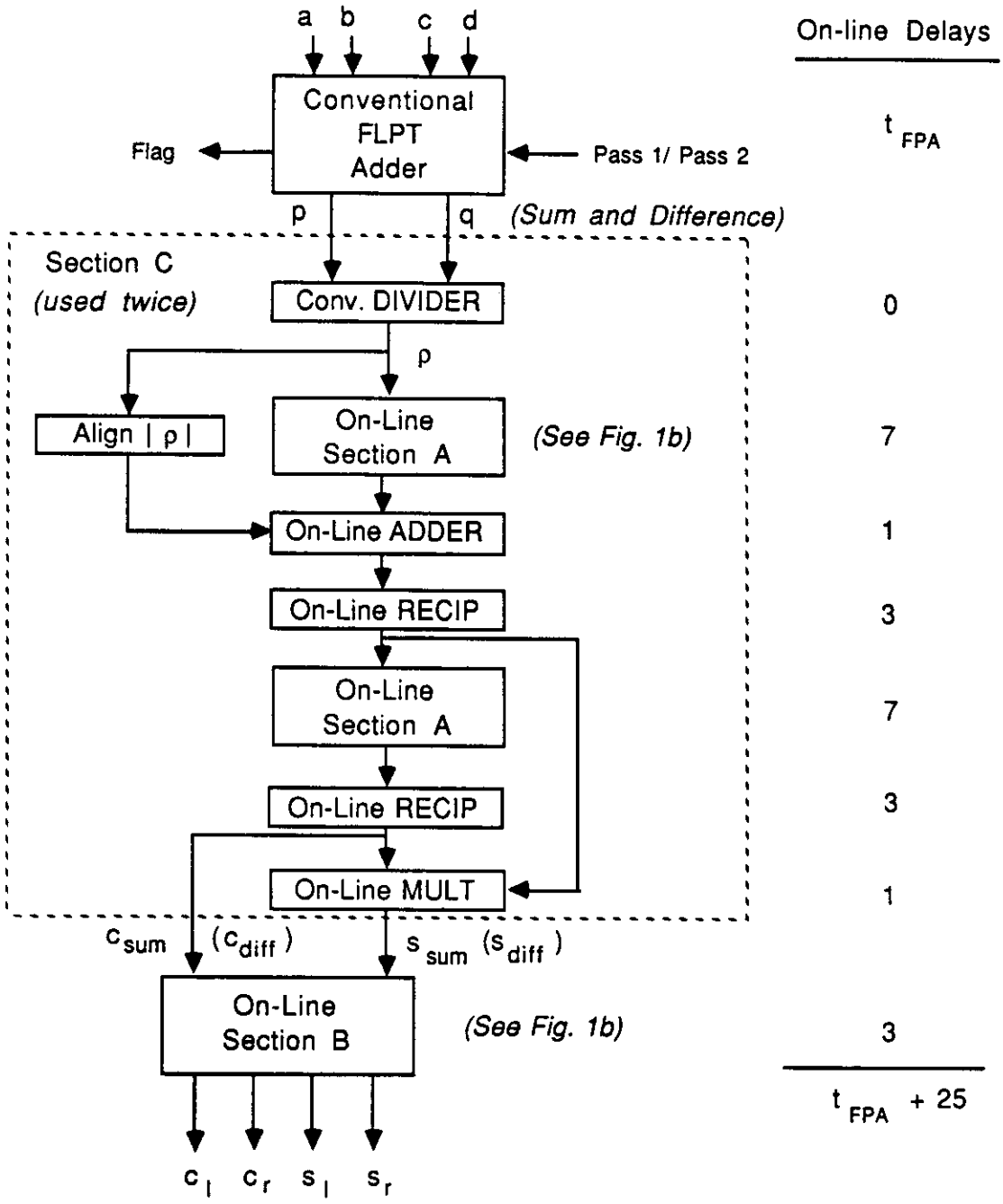
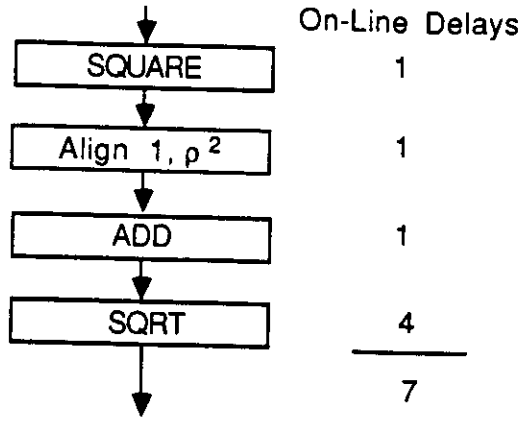
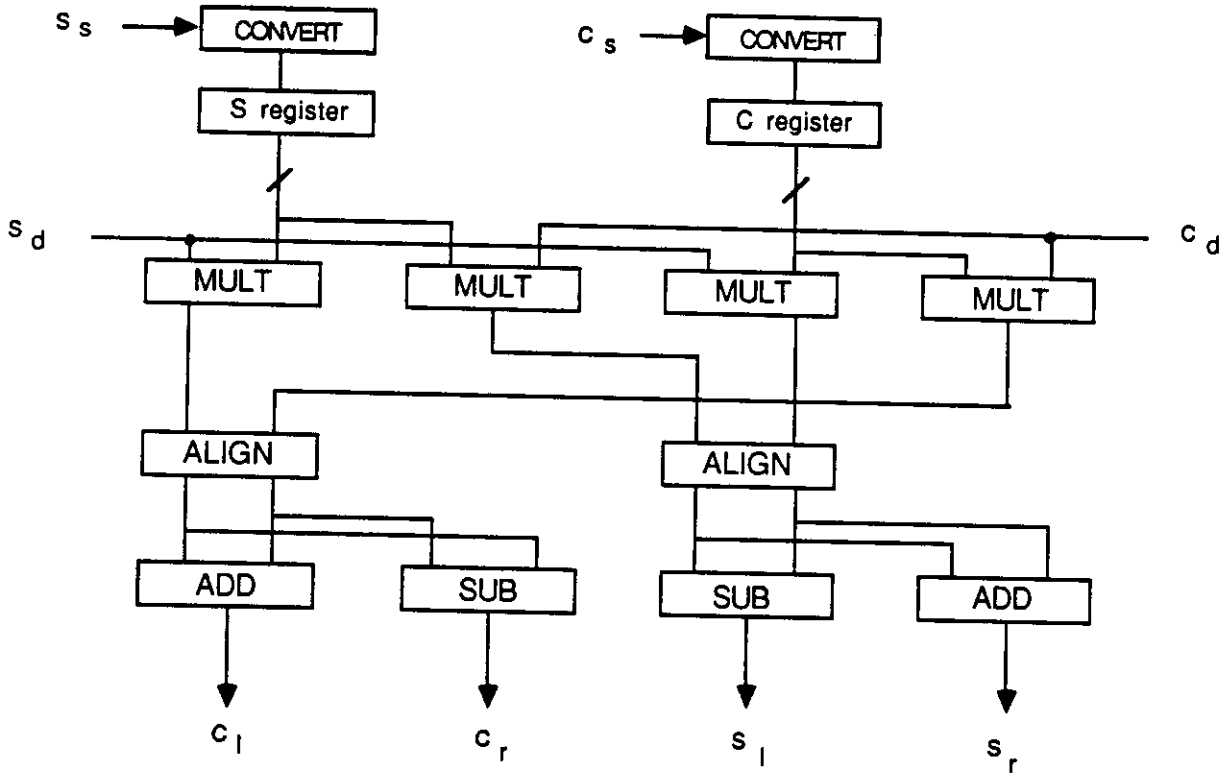


Figure 1a: On-Line Scheme for Computing SVD Rotation Factors



On-Line Section A



On-Line Section B

Figure 1b: On-Line Sections A and B

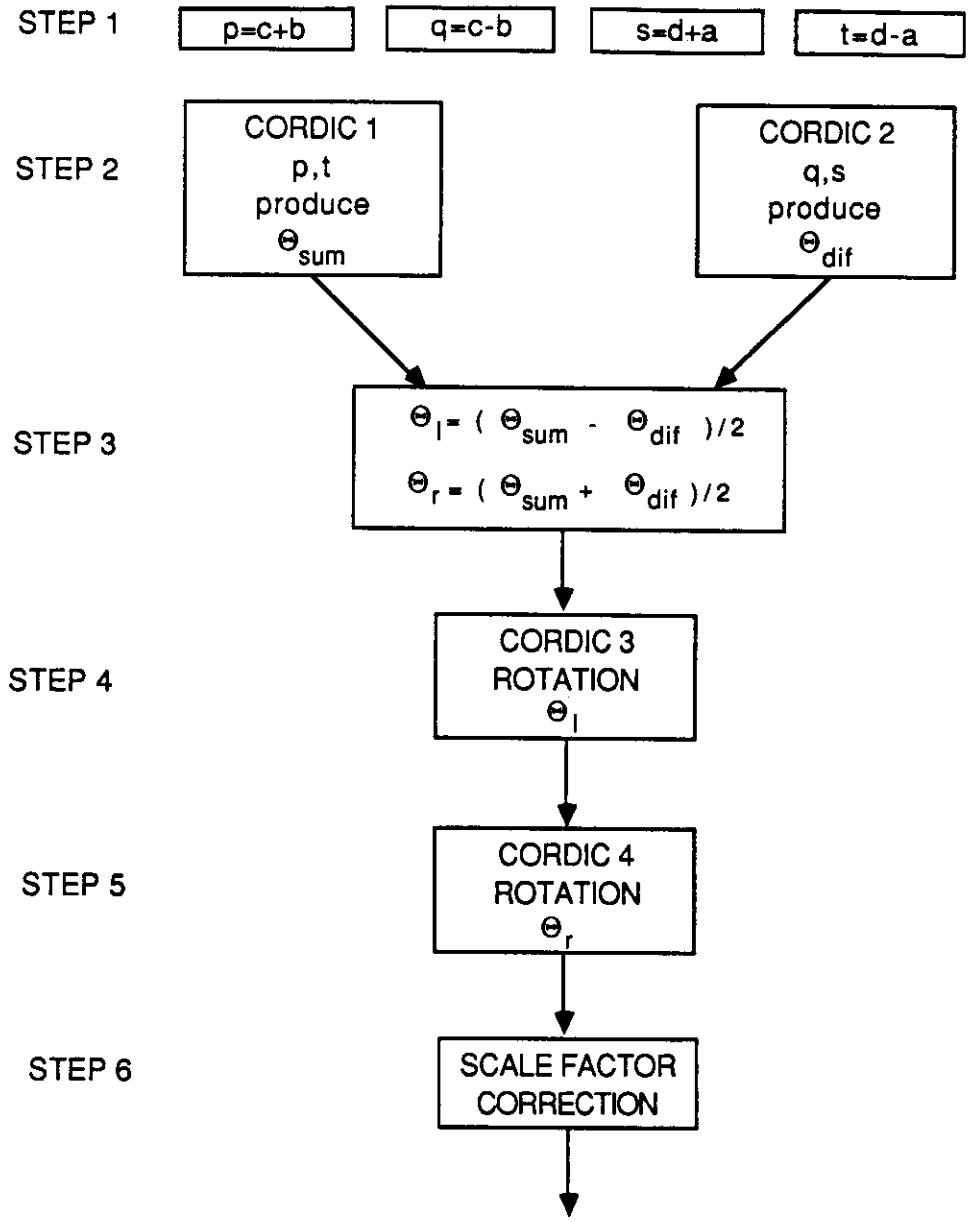
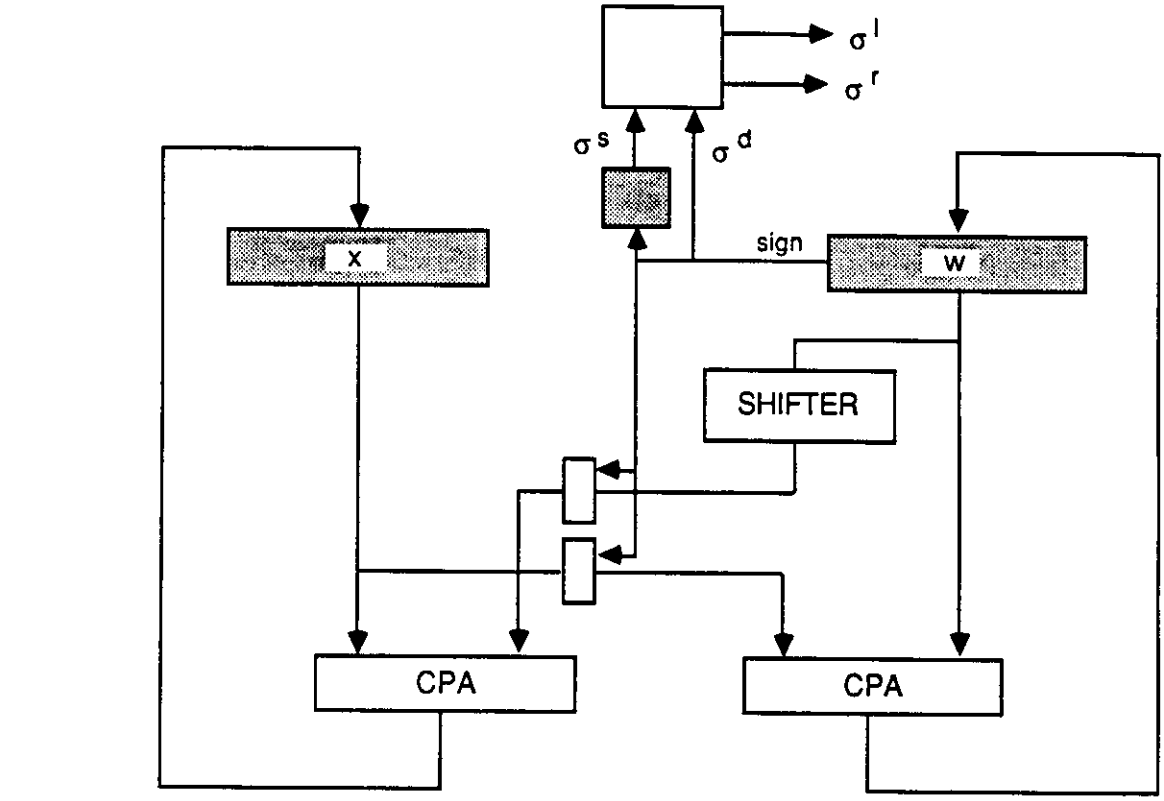


Figure 2: CORDIC Scheme



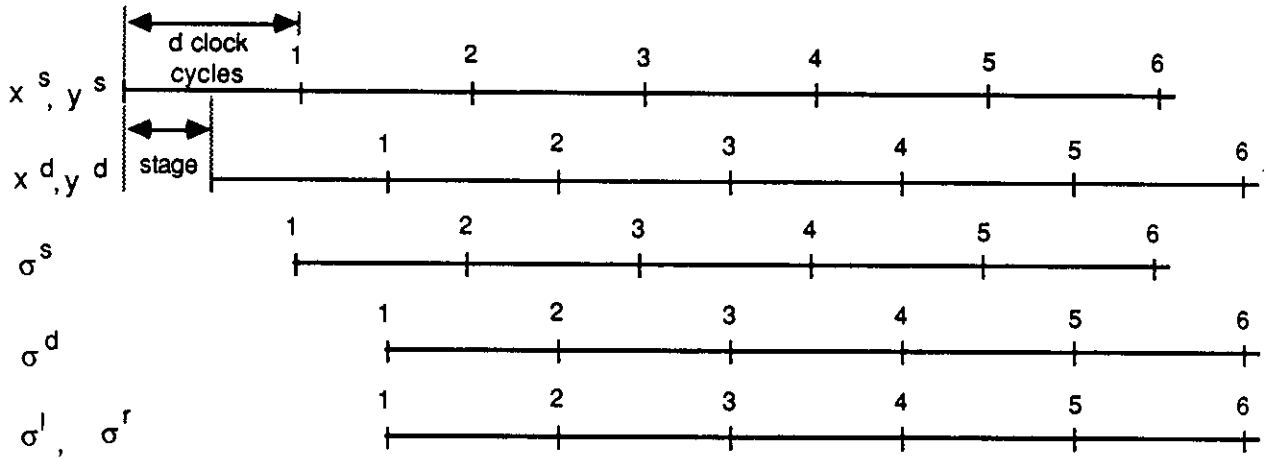


Figure 3: Implementation and Timing

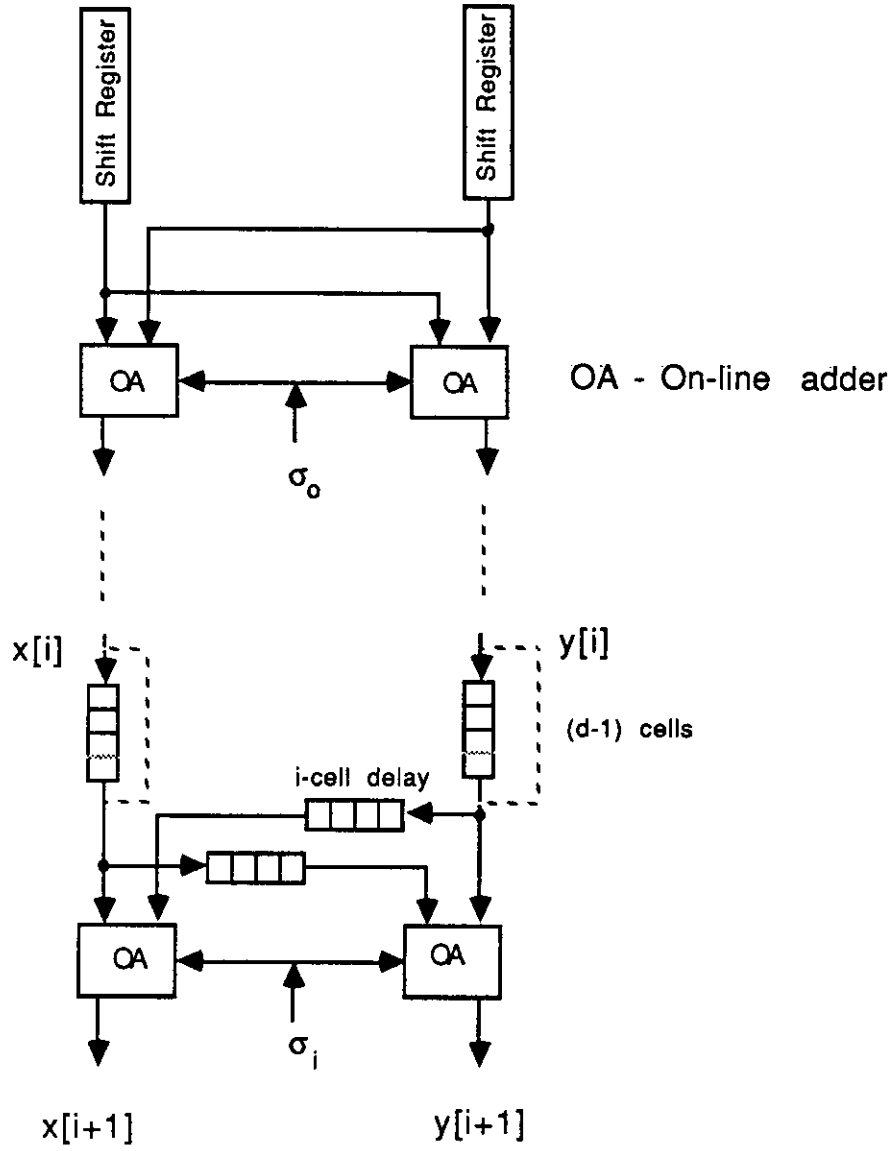


Figure 4a : On-Line Scheme for x,y Recurrence
(for Rotation)

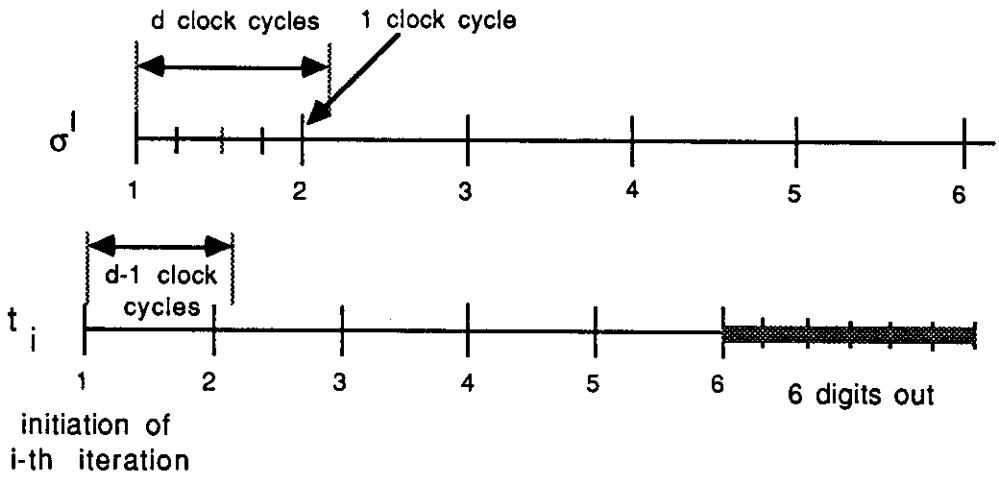


Figure 4b: Timing for Left-Angle Rotation

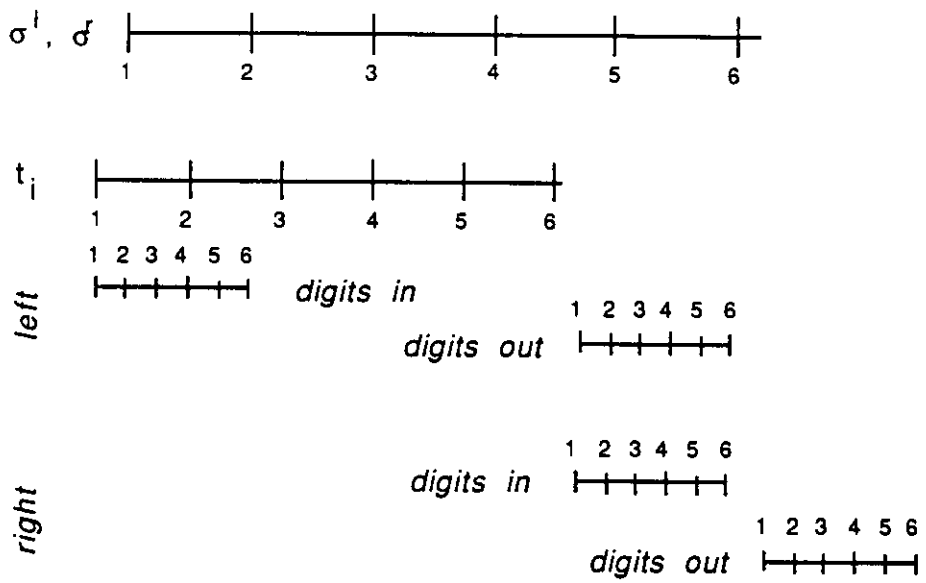


Figure 5: Timing of Two-Sided Rotation

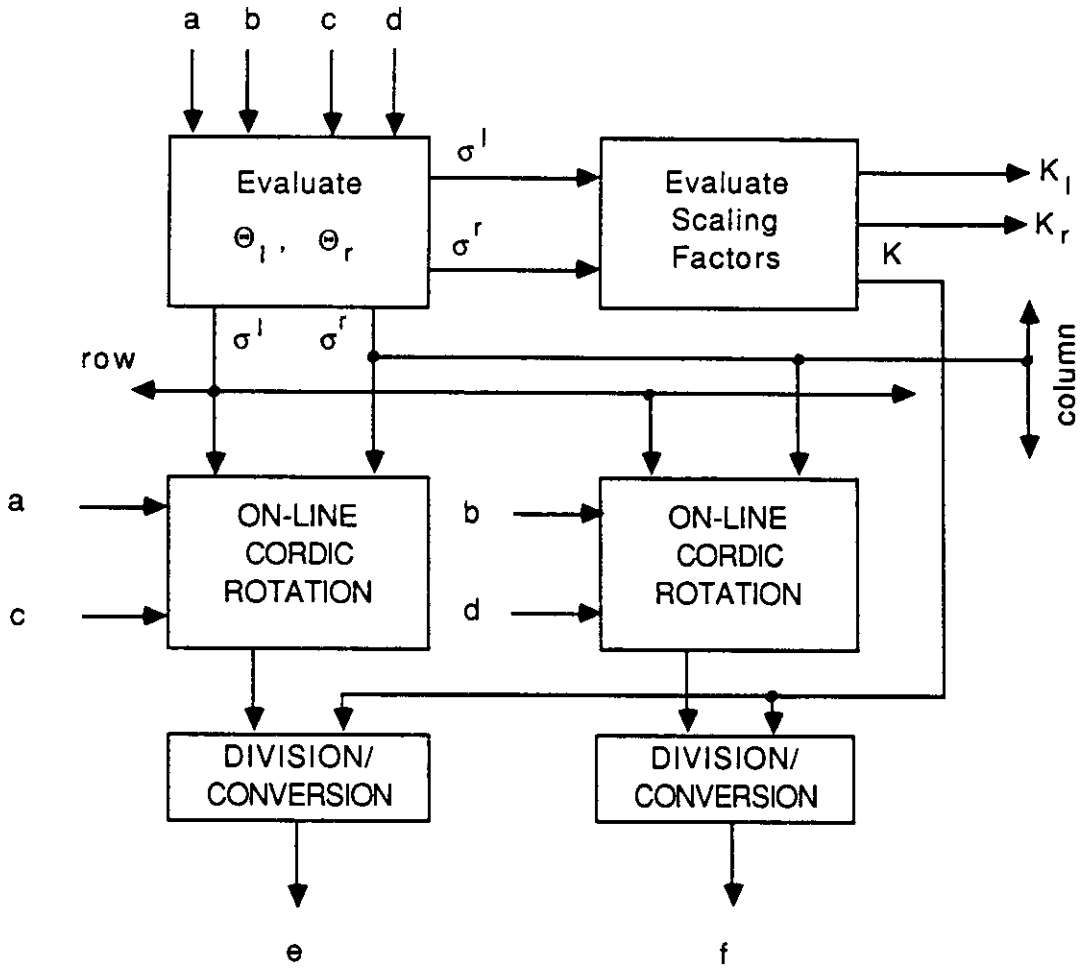


Figure 6a: Diagonal Processor Organization

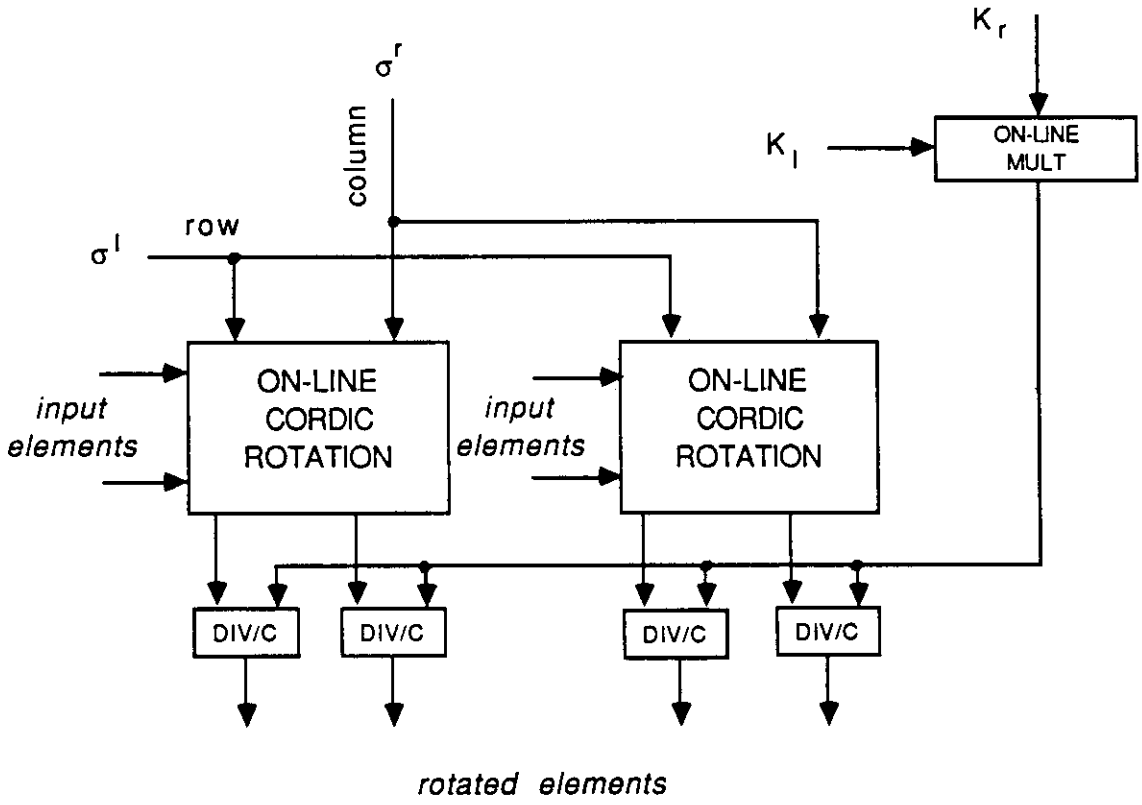


Figure 6b: Off - Diagonal Processor Organization

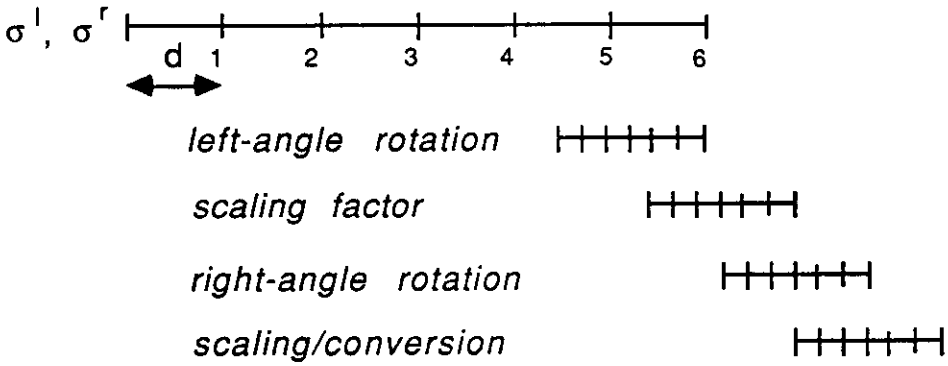


Figure 6c: Overall Timing