Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

FLOATING-POINT IMPLEMENTATION OF REDUNDANT CORDIC FOR QR DECOMPOSITION

Jeong-A Lee
Tomas Lang

July 1989
CSD-890044

Floating-point implementation of Redundant CORDIC for QR Decomposition

Jeong-A Lee, Tomás Lang [1]
Computer Science Department
School of Engineering and applied Sciences
University of California, Los Angeles
3680 Boelter Hall
Los Angeles, Calif. 90024

[1]This research has been supported by NSF, Contract MIP-88-13340

Abstract

The floating-point implementation of the redundant CORDIC scheme for matrix triangularization is presented. Due to the characteristic of CORDIC recurrence equations, the floating-point operations can be converted into fixed-point operations with pre and post-processing. The advantages of converting into fixed-point operations with pre and post-processing are the eliminations of exponent calculation and alignment operation in every CORDIC iteration. To utilize the scheme, the range of mantissa need to be calculated so that overflows in fixed-point operations with pre and post-processing can be avoided. Using parallel carry save scheme, implementation issues are discussed in detail including the range requirement.

1 Introduction

Matrix triangularization is a useful tool to solve a linear system, $Ax = b$ [GL83]. Redundant CORDIC implementation for matrix triangularization using Given's rotations has been proposed in [EL87]. The purpose of this report is to present the floating-point implementation of the redundant CORDIC scheme and discuss implementation issues in detail.

For fixed-point CORDIC implementation, several have been presented [HT80,CL87,SPH87] and have shown that registers, adders, and variable shifters are needed. For floating-point (non-redundant) CORDIC implementation, previous work [Wal71,Ahm82,JK86,CL88] transforms the shifting requirement of fixed-point implementation into exponent subtraction and mantissa alignment. Thus in every iteration, conventional floating-point operation is needed, i.e., result exponent calculation, alignment, and add/subtract operation.

However, as discussed in [EL87] and partially shown in [dLvdHDJ88], due to the characteristic of CORDIC recurrence equations, floating-point operations in every recurrence step are not needed. Instead, floating-point operations in every iteration can be converted into fixed-point operations with pre and post-processing. The advantages of converting into fixed-point operations with pre and post-processing are the eliminations of exponent calculation and alignment operation in every CORDIC iteration. To utilize this scheme, the range of mantissa needs to be calculated so that overflow can be avoided. Fortunately, with circular mode of CORDIC which is used for matrix triangularization, the range of mantissa is bounded and can be determined.

Concerning the block-level implementation, the recurrence equations can be implemented either using on-line CORDIC or parallel redundant arithmetic [EL87]. Here, we choose to have parallel carry save(CS) scheme for both angle and rotation units. In the following sections, the range requirements and other issues related to floating-point implementation for redundant CORDIC are discussed in detail.

2 Floating-point implementation for redundant CORDIC angle unit

We want to find the angle θ_Q and rotated vector $a\prime_{i,i}$ satisfying the following relation:

$$\begin{bmatrix} a\prime_{i,i} \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\theta_Q & -\sin\theta_Q \\ \sin\theta_Q & \cos\theta_Q \end{bmatrix} \begin{bmatrix} a_{i,i} \\ a_{i+1,i} \end{bmatrix}$$

Thus,

$$\theta_Q = \tan^{-1} -(\frac{a_{i+1,i}}{a_{i,i}})$$

This can be calculated by using the CORDIC approach [Vol59,Wal71,Ahm82]. As shown in [EL87], the conventional CORDIC recurrence equations has been modified to eliminate one area-consuming variable shifter and, at the same time, to look at a few significant bits of $W[j]$ to determine σ_j. Let $W[j]$ be $2^jY[j]$, then the recurrence equations for redundant CORDIC become

$$X[j+1] = X[j] + \sigma_j 2^{-2j} W[j]$$

$$W[j+1] = 2(W[j] - \sigma_j X[j])$$

Here, the angle recurrence equation, which is $Z[j+1] = Z[j] + \sigma_j \tan^{-1}(2^{-j})$, is not needed since instead of the angle $\theta_Q = Z[n] = \tan^{-1}(\frac{Y[0]}{X[0]})$, σ_j are used directly in the rotation unit.

The direction of the angle θ_Q, i.e., σ_j is computed from the following equation:

$$\sigma_j = \begin{cases} 1 & \text{if } \hat{W}[j] > 0 \\ 0 & \text{if } \hat{W}[j] = -1/2 \\ -1 & \text{if } \hat{W}[j] \leq -1 \end{cases}$$

where $\hat{W}[j]$ is computed using one fractional bit of $W[j]$.

To use $\hat{W}[j]$ to determine σ_j, we need to find appropriate range of $X[j-1]$ to have enough overlap in P-D graph shown in figure 1. Moreover, to have a low precision constant threshold of $W[j]$, we also need overlap of selection intervals. Low precision constant threshold of $W[j]$ allows to have a simple selection function module for σ_j. Considering all these factors, the range of mantissa of $X[1]$, $M_{X[1]}$ has been determined [EL87] to be: $\frac{1}{2} \leq M_{X[1]} < 1$, i.e., mantissa of $X[1]$ needs to be normalized.

To normalize $M_{X[1]}$ we need to get one vector from the two vector forms of $M_{X[1]}$, i.e., carry vector and sum vector with parallel CS arithmetic implementation. In other words, to normalize $M_{X[1]}$ we need to perform the time-consuming carry-propagate addition. However, this can be avoided by changing the range $M_{X[1]}$ to be $\frac{1}{2} \leq M_{X[1]} < 1.063$. The upper bound 1.063 instead of the conventional upper bound 1 is derived from the following facts: First, the implementation of simple selection function only requires that $M_{X[1]} \geq \frac{1}{2}$ for overlapped regions in P-D graph shown before. Secondly, the bound for mantissa $W[j]$ for all integer j, $|M_{W[j]}| \leq 3.5$ is needed for simple selection function implementation as explained in the section 2.3. As shown in the section 2.2, if $M_{X[1]} < 1.063$, then $|M_{W[j]}| \leq 3.5$.

The procedure to make $M_{X[1]}$ within the range, $\frac{1}{2} \leq M_{X[1]} < 1.063$, is simple: We need to compare the exponent of $X[0]$ and $W[0]$ and need to look at a few most significant bits of $M_{X[0]}$ and $M_{W[0]}$. The detailed procedure is described in the section 2.1. Moreover, with the new range of $M_{X[1]}$ the implementation complexity of the selection function for σ_j remains the same as before.

Now, let's see how we can convert floating-point operations of the angle recurrence equations into fixed-point operations with pre and post-processing.

2.1 Fixed-point operations with pre and post-processing

We want to have the same exponent for $M_{X[1]}$ and $M_{W[1]}$ so that the rest of the iterations can be done with fixed-point operations. Furthermore, we want to satisfy the condition, $|M_{W[1]}| < 2M_{X[1]}$ so that $M_{X[i]}$ and $M_{W[i]}$ for all i are bounded within a range in order to avoid overflows with fixed number of integer bit implementation.

Regardless of σ_0 value, the exponent of $M_{X[1]}$ and $M_{W[1]}$ can be made the same. However, if $\sigma_0 = 0$, the condition, $|M_{W[1]}| < 2M_{X[1]}$ can not be always satisfied. To satisfy the condition, σ_0 has to be ± 1. Fortunately, with floating-point (non-redundant) inputs $X[0]$ and $W[0]$, we can guarantee that $\sigma_0 \neq 0$ because of the following fact: Since we already have the exact value of input

$W[0]$, σ_0 can be determined from the original selection function [Wal71], i.e.,

$$\sigma_0 = \begin{cases} 1 & \text{if } W[0](= Y[0]) \geq 0 \\ -1 & \text{if } W[0] < 0 \end{cases}$$

In other words, by looking at the most significant bit of $W[0]$, σ_0 can be determined.

The function of pre-processing is to find the exponent of $M_{X[1]}$ and $M_{W[1]}$, Exp_{result} where $M_{X[1]}$ satisfies the range: $\frac{1}{2} \leq M_{X[1]} < 1.063$.

From the conventional floating-point inputs,

$$X[0] = M_{X[0]} 2^{Exp_X} \ , \frac{1}{2} \leq M_{X[0]} < 1$$

$$W[0] = M_{W[0]} 2^{Exp_W} \ , \frac{1}{2} \leq |M_{W[0]}| < 1$$

Here we require normalized inputs $M_{X[0]}$ and $M_{W[0]}$ to have maximum significant digits(bits) possible. Furthermore, we force initial $X[0] \geq 0$ by changing the sign of $W[0]$ appropriately so that $X[i+1] \geq X[i]$ for all i. When $X[i+1] \geq X[i]$ for all i, $X[n]$ becomes the upper bound as shown later in section 2.2. Notice that forcing $X[0] \geq 0$ has no effect on the value of θ_Q as θ_Q depends on $\frac{W[0]}{X[0]}$. However, the sign of $X[n]$ need to be negated to get rotated vector $(a_{i,i})$ if given $X[0] < 0$.

From the recurrence equation, we know that

$$X[1] = X[0] + \sigma_0 W[0]$$

$$W[1] = 2(W[0] - \sigma_0 X[0])$$

Since $\sigma_0 W[0] > 0$, i.e., the two operands to get $X[1]$ are always positive, and one of them (either $M_{X[0]}$ or $\sigma M_{W[0]}$) is always greater than 0.5 and less than 1, $M_{X[1]}$ in CS form and Exp_{result} can be obtained as follows: If the difference of the two exponents is greater than 3, no more processing is needed since the sum of the two operands is always less than the upper bound, 1.063. Therefore, the two operands can be output as sum and carry vector. Otherwise, the sum of the two operands can be bigger than the upper bound in some cases. For example, if $M_{X[0]}(\sigma M_{W[0]}$ respectively) is 0.1111 and $\sigma M_{W[0]}(M_{X[0]}$ respectively) is 0.0001, the sum of the two is bigger than the upper bound. The cases are shown below.

$$\begin{array}{cccc} |0.1111| & |0.111| & |0.11| & |0.1| \\ |0.0001| & |0.001| & |0.01| & |0.1| \end{array} \quad (1)$$

Moreover, in these cases, the sum of the two operands is always greater than 1. Thus, by shifting both operands one position to the right, which is equivalent to dividing by 2, the sum of the two operands is greater than 0.5 and less than 1.063.

Notice that since $X[0]$ and $W[0]$ are in conventional number representation, i.e., with one vector each, the two vectors of CS form of both $X[1]$ and $W[1]$ can be obtained by doing only alignment operations. In other words, the add/subtract operations in the recurrence equations need not to be implemented to get $X[1]$ and $W[1]$. For the hardware implementation of pre-processing, the alignment operations can be implemented using shifters.

The functional description of the pre-processing to find Exp_{result} for CS form of $X[1]$ and $W[1]$ is shown below. To reduce the number of alignment unit from two to one, we purposely assign aligned operands to one of the output vectors (carry or sum vector). Here we choose sum vector. The block-level implementation of pre-processing is shown in figure 2.

Pre-processing

/*

- Inputs:
  Conventional (not CS form) floating-point inputs $X[0]$ and $W[0]$
- Outputs:
  SignChanged to indicate whether given $X[0] < 0$,
  CS form $M_{X[1]}$ satisfying $\frac{1}{2} \leq M_{X[1]} < 1.063$,
  CS form $M_{W[1]}$,
  and Exp_{result} for both $M_{X[1]}$ and $M_{W[1]}$

*/

If $X[0] < 0$

then $X[0] = -X[0]$; $W[0] = -W[0]$;
SignChanged = true ;

If ($|Exp_{X[0]} - Exp_{W[0]}| \leq 3$) and
the two operands belong to the cases shown in equation 1

then $Exp_{result} = \max(Exp_{X[0]}, Exp_{W[0]}) + 1$
/* Align one of the operand according to the Exp_{result}. */

If ($Exp_{X[0]} \geq Exp_{W[0]}$)
then Carry($M_{X[1]}$) = $\frac{1}{2}M_{X[0]}$;
Sum($M_{X[1]}$) = Aligned ($\sigma_0 M_{W[0]}$)
/* apply shift right ($Exp_{X[0]} - Exp_{W[0]}$) + 1 to $\sigma_0 M_{W[0]}$ */

else Carry($M_{X[1]}$) = $\frac{1}{2}\sigma_0 M_{W[0]}$;
Sum($M_{X[1]}$) = Aligned ($M_{X[0]}$)

else $Exp_{result} = \max(Exp_{X[0]}, Exp_{W[0]})$
/* Align one of the operand according to the Exp_{result} */

If ($Exp_{X[0]} > Exp_{W[0]}$)
then Carry($M_{X[1]}$) = $M_{X[0]}$;
Sum($M_{X[1]}$) = Aligned ($\sigma_0 M_{W[0]}$)
else Carry($M_{X[1]}$) = $\sigma_0 M_{W[0]}$;
Sum($M_{X[1]}$) = Aligned ($M_{X[0]}$)

/*
Similarly, $M_{W[1]}$ is computed according to Exp_{result} as follows:

$$W[1] = 2(W[0] - \sigma_0 X[0]) = (M_{W[1]})2^{Exp_{result}}$$

*/

If ($|Exp_{X[0]} - Exp_{W[0]}| \leq 3$) and
the two operands belong to the cases shown in equation 1

then /* Corresponding Exp_{result} is as follows: $Exp_{result} = \max(Exp_{X[0]}, Exp_{W[0]}) + 1$. */

If ($Exp_{X[0]} \geq Exp_{W[0]}$)
then Carry($M_{W[1]}$) = $-\sigma_0 M_{X[0]}$;
Sum($M_{W[1]}$) = Aligned ($M_{W[0]}$)
else Carry($M_{W[1]}$) = $M_{W[0]}$;
Sum($M_{W[1]}$) = $-$ Aligned ($\sigma_0 M_{X[0]}$)

else /* corresponding Exp_{result} is as follows: $Exp_{result} = \max(Exp_{X[0]}, Exp_{W[0]})$
The two times operation need not to be implemented using shifter. Instead, wiring is used. */

If ($Exp_{X[0]} > Exp_{W[0]}$)
then Carry($M_{W[1]}$) = $-2\sigma_0 M_{X[0]}$;
Sum($M_{W[1]}$) = 2 Aligned ($M_{W[0]}$)
else Carry($M_{X[1]}$) = $2M_{W[0]}$;
Sum($M_{X[1]}$) = -2 Aligned ($\sigma_0 M_{X[0]}$)

Once we obtain the exponent, Exp_{result}, we do not need to align the operand X and W in the following recurrence equations since the exponent is not changed. By having the same exponent for $X[1]$ and $W[1]$, the floating-point operations for angle unit can be implemented as fixed-point operations. In other words, floating point addition/subtraction operations of most iterations become fixed-point operations, that is, the alignment step in floating point operation can be eliminated in all the n iterations except the first one.

Angel unit produces two outputs, σs for angle θ_Q and $X[n]$ for rotated vector $a_{i,i}$. Notice that post-processing is not needed to produce θ_Q since $\theta_Q - \frac{\sigma_0 \pi}{4}$ only depends on $\frac{W[1]}{X[1]}$. On the other hand, post-processing are needed to get the rotated vector $a_{i,i}$ from $X[n]$.

Due to the inexact rotation of CORDIC, $X[n]$ need to be corrected with scale correction factor K. As explained in section 4 we choose to use SRT division for correction step. In SRT, the dividend,$X[n]$, needs to be smaller than the divisor,K. Since the range of variable scaling factor K is between 1 and 1.65, the function of post-processing in rotation unit is to make the mantissa of $X[n]$ smaller than 1 using Exp_{result}. Implementation of post-processing is simple because with proper wiring to the SRT unit, the mantissa of $X[n]$ can be made smaller than 1. Notice that the exponent of $X[n]$ is not needed until the SRT operation is done. Thus, adjusting the exponent of $X[n]$ due to the post-processing can be done in parallel with the SRT operation. In addition, since

we force $X[0] > 0$ in angle unit, we need to negate the sign of $X[n]$ if SignChanged is true. The functional description of the post-processing is shown below.

Post-processing

/*

- Inputs:
  $X[n]$ where mantissa of $X[n] < 1.75$ as shown in section 2.2
  SignChanged indicating whether given $X[0] < 0$
- Output:
  $X[n]$ where mantissa of $X[n] < 1$

*/

If SignChanged is true

then $X[n] = -X[n]$;
Wire to SRT unit so that mantissa of $|X[n]| < 1$

To have fixed-point operations and avoid overflows, the ranges of X mantissa and W mantissa must be bounded. The discussion to determine the ranges of mantissa X and mantissa W is covered in the following section 2.2.

2.2 Range requirement of mantissa in angle unit

In the following descriptions, a value of operand A is assumed to be the value of its mantissa A, M_A. From X recurrence, we know that if we force the initial value of $X[0] \geq 0$ and $X[1] \geq \frac{1}{2}$, then for all integer i, $X[i+1] \geq X[i]$ since $\sigma_i 2^{-2i} W[i] \geq 0$. As mentioned before, the condition of $X[0] \geq 0$ can be easily accommodated by changing the sign of $W[0]$ appropriately and the angle output θ is still the correct one because θ depends on $\frac{W[0]}{X[0]}$.

Since $X[1]$ is adjusted to satisfy $\frac{1}{2} \leq M_{X[1]} < 1.063$, the upper bound of $X[i]$ can be found as follows: For all i, $X[i+1] \geq X[i]$. We also proved that $X[n] < 1.75$. Therefore, for all i, $X[i] < 1.75$.

Proof $X[n] < 1.75$
We know that $X[n] = K\sqrt{X[0]^2 + Y[0]^2} = K'\sqrt{X[1]^2 + Y[1]^2}$ where $K = \prod_{i=0}^{n-1} \sqrt{1 + \sigma_i^2 2^{-2j}}$ and $K' = \prod_{i=1}^{n-1} \sqrt{1 + \sigma_i^2 2^{-2j}}$. Thus, $K = \sqrt{2}K'$ since $\sigma_0^2 = 1$. Furthermore, $X[1] \leq 1.063$. From the recurrence equation, we know that $|Y[1]| < X[1]$. Thus, $|Y[1]| < 1.063$. As a result, $X[n] = K'\sqrt{X[1]^2 + Y[1]^2} < 1.75$ because $1.65 < K < 1.66$.

Furthermore, for all i, the relation of $|W[i]| < 2X[i]$ holds. This is satisfied because of the way the selection function is developed in [EL87]. As $|W[i]| < 2X[i]$, we get $|W[i]| < 3.5$. For these ranges, we need to provide enough number of integer bits to avoid overflows and to assure the correct angle result.

As the range of $X[j]$ is bounded by 1.75 and for all integer j, $X[j]$ is positive number, one integer bit is enough for $X[j]$. On the other hand, four integer bits are needed for $W[j]$ and the explanation is as follows: As $W[j]$ is bounded by 3.5, three integer bits for W are necessary to have two's complement number representation. Furthermore, one more integer bit extension for W is needed to protect the MSB which denotes the sign of its operand in CS(carry save) representation. This extra bit extension is necessary because as shown in recurrence equation, the following operation after computing $W[i]$ is $2^{-2i}W[i]$ in $X[i+1]$ calculation. This 2^{-2i} term multiplication is implemented as ASR(Arithmetic Shift Right) where sign extension is needed. In order to keep MSB pair of carry and sum vector to represent the right sign, the extra bit extension in MSB of W is needed.

More specifically, we do not want to have the two most significant bits of the carry and sum vector respectively with the following patterns.

$$WS \quad |00| \quad |01| \quad |10| \quad |11|$$

$$WC \quad |01| \quad |00| \quad |11| \quad |10|$$

With these patterns, the sign-extension of the present MSB can be wrong if there is any carry propagation. For example, if the two vectors have the bit patterns of table 1, the value of the two vectors, i.e., the sum of the two vectors, represents a negative number due to the carry propagation. If we just apply the normal ASR operation, the result of one-bit ASR, i.e., one-bit sign-extended vectors, represents a positive number now. On the other hand, if there is no carry propagation, the sign-extension of the present MSB extension gives the correct result. In summary, the carry propagation needs to be checked to provide the correct sign-extension. In order to avoid checking of the carry propagation, one more integer bit is extended so that the above bit patterns can not appear. With one more integer bit extension, the bit patterns of (01)(00) and (10)(11) become overflow data since $|W| < 3.5$ and 4 integer bits are used in the mantissa representation.

With 4 integer bits used for W, the following are the valid cases of the two most significant bits of carry and sum vector.

$$WS \quad |00| \quad |01| \quad |10| \quad |11| \quad |0x| \quad |1x|$$

$$WC \quad |00| \quad |01| \quad |10| \quad |11| \quad |1x| \quad |0x|$$

Among the valid cases, the two cases shown below cause error, i.e., changing the sign of the original data after the following ASR operation.

$$WS \quad |01| \quad |10|$$

$$WC \quad |01| \quad |10|$$

For example, the second case (10)(10) can be obtained in CS form of $W[j+1] = 3$ with $W[j] = 3$ and $X[j] = 1.5$. With $W[j] = 3$, we know that $\sigma_j = 1$. Thus, $W[j+1] = 2(W[j] - \sigma_j X[j]) = 3$.

$$WC[j] \quad \textit{bit representation} \quad 0011.0$$

$$WS[j] \quad \textit{bit representation} \quad 0000.0$$

$$W[j] \textit{ decimal value } 3$$

$$-XS[j] \quad \textit{bit representation} \quad 0001.0$$

$$-XC[j] \quad \textit{bit representation} \quad 1101.1$$

$$-X[j] \textit{ decimal value } -1.5$$

Then, $WC[j] + WS[j] - XS[j]$ produces the following intermediate sum and carry vector, IS and IC from 3-to-2 reduction.

$$IS \quad 0010.0$$

$$IC \quad 0010.0$$

Next, $IS + IC - XC[j]$ produces the final sum and carry vector, FC and FS which is in decimal value $3 + (-1.5) = 1.5$ as follows:

$$FS \quad 1101.1$$

$$FC \quad 0100.0$$

$$\textit{output of } WC[j] + WS[j] - XS[j] - XC[j] \textit{ decimal value } 1.5$$

When the two times operation is applied to the output of FC and FS to get $W[j+1] = 2(W[j] - \sigma_j X[j])$ which corresponds decimal value 3, the resulting vector becomes:

$$2 * FS \quad 1011.0$$

$$2 * FC \quad 1000.0$$

$$W[j+1] \textit{ output decimal value } 3$$

By itself, the value is still a positive number since two's complement arithmetic is used. Thus, it does not produce any different outcome. However, when $2^{-2(j+1)}W[j+1]$, i.e., ASR operation is needed in the following iteration step for $X[j+2] = X[j+1] + \sigma_{j+1}2^{-2(j+1)}W[j+1]$, due to sign extension of ASR operation, 1 and 1 in MSB will change the resulted shift right carry save form into a negative number.

Thus we need to apply a remedy scheme. The key of the remedy scheme is changing the MSB pair of carry and sum vector into new pair so that the following ASR operation (sign extension) does not change the sign. For example, the pair (01)(01) for WC and WS represents a negative number. When ASR operation is applied to (01)(01), the resulted vector represents a positive number. To correct the problem, the remedy scheme changes (01)(01) for WC and WS into (11)(11). When ASR operation is applied to (11)(11), the resulted vector represents a negative number. Notice that the value of W after the application of remedy scheme is the same as the original one.

Before the remedy scheme

$$WS \quad |01| \quad |10|$$

$$WC \quad |01| \quad |10|$$

After the remedy scheme

$$WS \quad |11| \quad |00|$$

$$WC \quad |11| \quad |00|$$

The logic implementation of the remedy scheme is as follows:

Remedy scheme

If (WS[MSB] == WC[MSB])

then { WS[MSB]= WS[MSB+1] ; WC[MSB] = WC[MSB+1]; }
/* Assume index 0 for MSB and n for LSB in n-bit data */

The implementation using some logic and multiplexors is shown in figure 3. As recurrence equations related to X do not have ASR operation, the remedy scheme only applies to W.

2.3 Selection function implementation

The direction of angle, σ_i is obtained from estimated W, i.e., $\hat{W}[i]$ in carry save (CS) form. Let t be the number of the fractional bits for $\hat{W}$ and m be the number of the fractional bits for W. Thus, $t \leq m$. Then, the truncated sum and carry vector in CS form of $\hat{W}[i]$ could generate an error of $2(2^{-t} - 2^{-m})$. Thus, to be able to cover a range of $-A \leq W \leq A$, the range of $\hat{W}$ needs to be $-A - 2^{-t} \leq \hat{W} \leq A - 2^{-t}$.

For example, in the angle unit where $|W| < 3.5$ and $t = 1$, the range of $\hat{W}$ needs to be $-3.5 - 2^{-1} < \hat{W} < 3.5 - 2^{-1}$. Thus, 3 integer bits and 1 fractional bit for $\hat{W}[i]$ are enough to implement the selection function for σ_i although 4 integer bits are used for W due to the sign-extension operation as shown before. The reduction of 1 integer bit for $\hat{W}$ implies the reduction of 1 FA(full adder) time for selection function. As the selection function is included in the critical path of the angle unit, the reduction time of selection function implementation implies the reduction step time of angle unit. Finally the block-level implementation for floating-point angle unit is shown in figure 4.

3 CORDIC rotation unit

With given vector (X, Y) and angle θ_q, the rotated vector (Rx, Ry) is defined as:

$$\begin{bmatrix} Rx \\ Ry \end{bmatrix} = \begin{bmatrix} \cos\theta_q & -\sin\theta_q \\ \sin\theta_q & \cos\theta_q \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

If the angle θ_q is as follows:

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

Then, rotated vector can be obtained using the following CORDIC recurrence equations.

$$X[j+1] = X[j] + \sigma_j 2^{-j} Y[j]$$

$$Y[j+1] = Y[j] - \sigma_j 2^{-j} X[j]$$

with initial condition $X[0] = X$ and $Y[0] = Y$.
The rotated vector (Rx, Ry) has the following relation with $X[n]$ and $Y[n]$ for the n-bit precision result.

$$\begin{bmatrix} Rx \\ Ry \end{bmatrix} = \frac{1}{K} \begin{bmatrix} X[n] \\ Y[n] \end{bmatrix}$$

where K, correction factor, is defined as:

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

In this section, we deal with floating-point implementation of CORDIC rotation recurrence equations using parallel CS arithmetic. The correction factor,K, is covered in the following section.

Similar to the floating-point implementation of angle unit, the floating-point operations in rotation unit can be converted into fixed-point operations with pre and post-processing. As discussed previously, the implementation of pre and post processing is simple: some logic and wiring is sufficient to implement them.

3.1 Fixed-point operations with pre and post-processing

The rotation unit obtains σs from angle unit for the computation of recurrence equations. The initial inputs are in conventional forms, not in CS form.

$$X[0] = M_{X[0]} 2^{Exp_X}, \frac{1}{2} \leq |M_{X[0]}| < 1$$

$$Y[0] = M_{Y[0]} 2^{Exp_Y}, \frac{1}{2} \leq |M_{Y[0]}| < 1$$

Here we require normalized inputs $X[0]$ and $Y[0]$ to have maximum significant digits(bits) possible. From the recurrence equation,

$$X[1] = X[0] + \sigma_0 Y[0]$$

$$Y[1] = Y[0] - \sigma_0 X[0]$$

As $\sigma_0 \neq 0$, the floating point addition/subtraction of the above equations produce an exponent $Exp_{result} = \max(Exp_X, Exp_Y)$ and $X[1]$ and $Y[1]$ can be represented as follows:

$$X[1] = (M_{A+}) 2^{Exp_{result}}$$

$$Y[1] = (M_{A-}) 2^{Exp_{result}}$$

where M_{A+} denotes the mantissa output of $X[0] + \sigma_0 Y[0]$ and M_{A-} denotes the mantissa output of $Y[0] - \sigma_0 X[0]$.

In conventional floating-point operation, the outputs M_{A+} and M_{A-} are normalized at the end of the floating-point operation. As a result, exponents of $X[1]$ and $Y[1]$ can be different. Different exponents implies alignment operations is needed in the following iteration for $X[2]$ and $Y[2]$. However, the proposed implementation shown here eliminates alignment operations in the following iterations by not normalizing the outputs M_{A+} and M_{A-}, i.e., the exponent of $X[1]$ is the same one as that of $Y[1]$. Eliminating the post-normalization of M_{A+} and M_{A-} has an advantage that implementation becomes simpler since only alignment operations are needed to find the mantissa of $X[1]$ and $Y[1]$ in CS form. Moreover, in conventional floating-point operation, significant digits(bits) can be lost by doing the post-normalization at the end of the floating-point operation. However, the implementation proposed here eliminates post-normalization. Thus, the implementation shown here has another advantage of keeping more significant digits than the conventional floating-point implementation.

The fact that we do not normalize the mantissa outputs, M_{A+} and M_{A-} requires to find the range of X and Y to avoid overflows. The detailed procedure to find the range of X and Y is discussed in the following section 3.2. In brief, with the initial condition of conventional floating-point inputs $X[0]$ and $Y[0]$, the range of $X[n]$ and $Y[n]$ is found to be ± 2.33. Notice that in rotation mode, $X[n]$ can be negative value and $M_{X[1]}$ is not normalized while in angle calculation mode, $X[n]$ is always positive and $M_{X[1]}$ need to be in special range. As a result, the range of $X[n]$ and the number of integer bits to cover the range are not the same for both modes although X recurrence equation is the same for both angle calculation mode and rotation mode.

The function of pre-processing is summarized below:

Pre-processing

/*

- Inputs:
  Conventional (not CS form) floating-point inputs $X[0]$ and $Y[0]$
- Outputs:
  CS form $M_{X[1]}$,
  CS form $M_{Y[1]}$,
  and $Exp_{result} = \max\{Exp_{X[0]}, Exp_{Y[0]}\}$

*/

$Exp_{result} = \max(Exp_{X[0]}, Exp_{Y[0]})$
/* Align one of the operand according to the Exp_{result}. */

If $(Exp_{X[0]} > Exp_{Y[0]})$
then $\text{Carry}(M_{X[1]}) = M_{X[0]}$;
$\text{Sum}(M_{X[1]}) = \text{Aligned } (\sigma_0 M_{Y[0]})$;
$\text{Carry}(M_{Y[1]}) = -\sigma_0 M_{X[0]}$;
$\text{Sum}(M_{Y[1]}) = \text{Aligned } (M_{Y[0]})$
else $\text{Carry}(M_{X[1]}) = \sigma_0 M_{Y[0]}$;
$\text{Sum}(M_{X[1]}) = \text{Aligned } (M_{X[0]})$;
$\text{Carry}(M_{Y[1]}) = M_{Y[0]}$;
$\text{Sum}(M_{Y[1]}) = \text{Aligned } (-\sigma_0 M_{X[0]})$

After finding the exponent, Exp_{result} from the pre-processing, only fixed-point operations are applied to the rest of the iterations. Sufficient number of integer bits needs to be provided to avoid overflow of operands $X[n]$ and $Y[n]$ in the rest of iteration steps. The upper bound of operands $X[n]$ and $Y[n]$ are evaluated and the number of integer bits are decided in the following section 3.2.

Similar to angle unit, the output of rotation unit, $X[n]$ and $Y[n]$ need post-processing for the following SRT division(correction unit). Unlike angle unit, we do not force $X[0] > 0$ in rotation unit. Thus, changing of the sign of $X[n](Y[n])$ is not needed. The following is the functional description of post-processing of rotation unit.

Post-processing

/*

- Inputs:
  $X[n]$ where mantissa of $|X[n]| < 2.33$ as shown in section 3.2
  $Y[n]$ where mantissa of $|Y[n]| < 2.33$ as shown in section 3.2

- Outputs:
  $X[n]$ where mantissa of $|X[n]| < 1$
  $Y[n]$ where mantissa of $|Y[n]| < 1$

*/

Wire to SRT unit so that mantissa of $|X[n]| < 1$ and $|Y[n]| < 1$

In summary, the floating-point implementation of rotation unit consists of one time operation of pre-processing, fixed-point operations for the most of the iterations, and the post-processing. The implementation shown here has advantages of eliminating alignment and post normalization step in the conventional floating-point operation.

3.2 Range requirement of mantissa in rotation unit

Now, we need to find the proper bound for operands X and Y to have fixed-point operation for iterations except the first one. In circular mode of CORDIC, the value of $(X[n], Y[n])$ is bounded within the circle where the radius is defined as $K\sqrt{X[0]^2 + Y[0]^2}$ with given initial vector $(X[0], Y[0])$. Using the sequence of Walther where $\sigma_j \in \{\pm 1\}$, the maximum value of K is about 1.65. Therefore, in redundant parallel CS implementation where $\sigma_j \in \{0, \pm 1\}$, the variable scale factor $K \leq 1.65$. With floating-point inputs $X[0]$ and $Y[0]$, each mantissa has a value less than one. Therefore, the mantissa of rotated output $X[n]$ (or $Y[n]$) is bounded by 2.33 because $K\sqrt{X[0]^2 + Y[0]^2} \approx 2.33$.

Thus, three integer bits are necessary to have two's complement number representation. Furthermore, as discussed in angle unit, one more integer bit extension and use of remedy scheme are needed because recurrence equations for $Y[j+1]$ and $X[j+1]$ have $2^{-j}X[j]$ and $2^{-j}Y[j]$ operations respectively. The block-level implementation for floating-point CORDIC rotation is shown in figure 5.

4 Scale factor calculation and correction

In redundant CORDIC, scale factor $K = \prod_{j=0}^{n-1} \sqrt{1 + (\sigma_j 2^{-j})^2}$ is not a constant as $\sigma_j \in$ {-1, 0, 1}. Therefore K must be computed for each decomposed angle, σ_js.

The scheme to compute K is developed in [EL87] as follows: It has two steps. First compute K^2 with recurrence equation of $P[j+1] = P[j] + |\sigma_j|2^{-2j}P[j]$ with initial condition $P[0] = 1$. Due to the term $2^{-2j}P[j]$, the final output P can be obtained from $\frac{n}{2}$ steps instead of n steps, i.e., $P = P[\frac{n}{2}] = P[n]$. Secondly we compute $K = P^{\frac{1}{2}}$ using on-line square root algorithm [Erc78,OE82].

The computation of P is implemented using the parallel CSA(Carry Save Adder). From the recurrence equation P and initial condition $P[0] = 1$, we can observe that the output of P yields

a positive number. Moreover, the upper bound of P is square of upper bound of K, i.e., 1.65^2. Therefore, two integer bits are used to cover the range of P. The block-level implementation is shown in figure 6. Concerning the timing of K computation, since K^2 can be obtained from $\frac{n}{2}$ steps instead of n steps, the computation of K^2 can be entirely overlapped with operation of angle unit. Moreover, K^2 is obtained using CSA, the following square root operation need not to be on-line. Thus, K is obtained from square root operation one digit at a time. The output of square root operation is is in signed-digit representation.

As mentioned before, due to the inexact rotation of CORDIC, output of the rotation unit, $X[n]$ and $Y[n]$, need to be corrected with scale correction factor K to determine the rotated vector (Rx, Ry). To correct the rotated output, actual division need to be done since K is a variable factor in redundant CORDIC. Since the rotation unit is implemented using parallel CS arithmetic, redundant division scheme using SRT divider is used for the correction. Thus the scale correction factor unit starts operation when the rotated output comes in parallel at the end of the rotation iteration. For SRT division, the divisor, K need to be in conventional representation. Using recurrence equation of $P(= K^2)$ and square root operation, K is obtained in signed-digit representation. Thus, on-the-fly conversion is used to convert signed-digit output K into conventional representation for the following SRT division.

As the equation, $K = \prod_{j=0}^{n-1} \sqrt{1 + (\sigma_j 2^{-j})^2}$, where $\sigma_j \in$ {-1, 0, 1}, indicates, K is larger than 1 and smaller than 1.65. Thus, the divisor K in SRT operation is larger than the normalized number, which simplifies the SRT operation as the selection of quotient digit becomes independent of the divisor. In SRT, the dividend also needs to be smaller than the divisor. Here, the dividend is $X[n](Y[n])$. This condition is satisfied by the post-processing of rotation unit. Output of SRT is obtained one digit at a time. The output of SRT operation is in signed-digit representation.

The corrected output is used in the next QR step as an input to the angle unit. As parallel CS arithmetic is used in the angle unit, the conversion from signed-digit representation to conventional number representation is needed. For the conversion, on-the-fly conversion in [EL86] is used.

5 Overall-timing

Overall timing is shown in figure 7. Both CORDIC unit and scale factor evaluation unit have critical path of variable shifter, multiplexor, and two 3-to-2 reduction (4-to-2 reduction) carry save adder. Other units such as correction unit(SRT), on-the-fly unit, and square root unit have shorter critical path, for example in SRT, quotient selection, remainder computation using adder, and multiplexor. Thus, we assume that the step time of CORDIC units takes two basic cycles while the step time of SRT operation takes one basic cycle.

As shown in the overall-timing diagram in figure 7, K computation is completely overlapped with angle unit computation. Moreover, on-the-fly conversion operation does not cost the time since it can be done in parallel while SRT operation computes the remainder. Signed-digit quotient is used to compute the remainder and at the same time is used for on-the-fly-conversion unit. Thus, without costing any time, quotient digit in conventional number representation can be stored into a register. Thus, the total timing depends on the angle unit to generate σs and SRT division operation. The total timing for the overall system takes $3n$ basic cycles.

6 Evaluation of redundant parallel CS scheme

In conventional(non-redundant) CORDIC implementation for QR, the overall timing requires $2.25T_C$ where the cordic step time, $T_C \approx n(T_{CPA} + T_{variable-shifter})$ with n iteration. In terms of the same clock cycle used in redundant CORDIC, the conventional cordic step time becomes $5n$ or $6n$ basic cycles. Thus, overall timing is $12.25n$ or $13.50n$ basic cycles [EL87]. Parallel CS approach presented here takes $3n$ basic cycles. As a result, it produces a speed up of around factor 4 with respect to non-redundant CORDIC implementation.

References

[Ahm82] H. M. Ahmed. *Signal Processing Algorithms and Architectures.* Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, 1982.

[CL87] Joseph R. Cavallaro and Franklin T. Luk. Cordic arithmetic for an svd processor. *Proceeding of 8th Symposium on Computer Arithmetic*, 113–120, 1987.

[CL88] Joseph R. Cavallaro and Franklin T. Luk. Floating-point cordic for matrix computations. In *ICCD*, pages 40–42, October 1988.

[dLvdHDJ88] 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 *ISCAS*, pages 2043–2047, 1988.

[EL87] Milos Ercegovac and Tomas Lang. *Redundant and On-Line CORDIC: Application to Matrix triangularization and SVD.* Technical Report, UCLA, University of California, Los Angeles, September 1987.

[Erc78] Miloš D. Ercegovac. An on-line square rooting algorithm. In *Proceedings of the 4th Symposium on Computer Arithmetic*, pages 183–189, 1978.

[GL83] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* The Johns Hopkins University Press, 1983.

[HT80] Gene L. Haviland and AL A. Tuszynski. A cordic arithmetic processor chip. *IEEE Transactions on Computers*, C-29(2):68–79, February 1980.

[JK86] S. Lennart Johnsson and Venkatesh Krishnaswamy. *Floating-point CORDIC.* Research Report YALEU/DCS/RR-473, Department of Computer Science, Yale University, April 1986.

[OE82] V.G. Oklobdzija and Miloš D. Ercegovac. An on-line square root algorithm. *IEEE Transactions on Computers*, C-31(1):70–75, Jan 1982.

[SPH87] Tze-Yun Sung, Tai-Ming Parng, and Yu-Hen Hu. Doubly pipelined cordic array for digital signal processing algorithms. *J. Chin. Inst. Eng.*, 10(4):375–383, July 1987.

[Vol59] J.E. Volder. The cordic trigonometric computing technique. *IRE Trans. Electronic Computers*, EC-8:330–334, September 1959.

[Wal71] J.S. Walther. A unified algorithm for elementary functions. *AFIPS Spring Joint Computer Conference*, 379–385, 1971.

Bit patterns with carry propagation

000.1|10

011.1|11

Corresponding decimal value is $-3.375 (> -3.5)$

After ASR 1 (i.e., one-bit sign-extension)

000.0110

001.1111

Now corresponding decimal value becomes 2.0625 instead of $-\frac{3.375}{2}$

Bit patterns without carry propagation

000.0|00

011.0|10

Corresponding decimal value is $3.25 (< 3.5)$

After ASR 1 (i.e., one-bit sign-extension)

000.0000

001.1010

Now corresponding decimal value becomes 1.625 $(= \frac{3.25}{2})$

Table 1: Examples with and without carry propagation

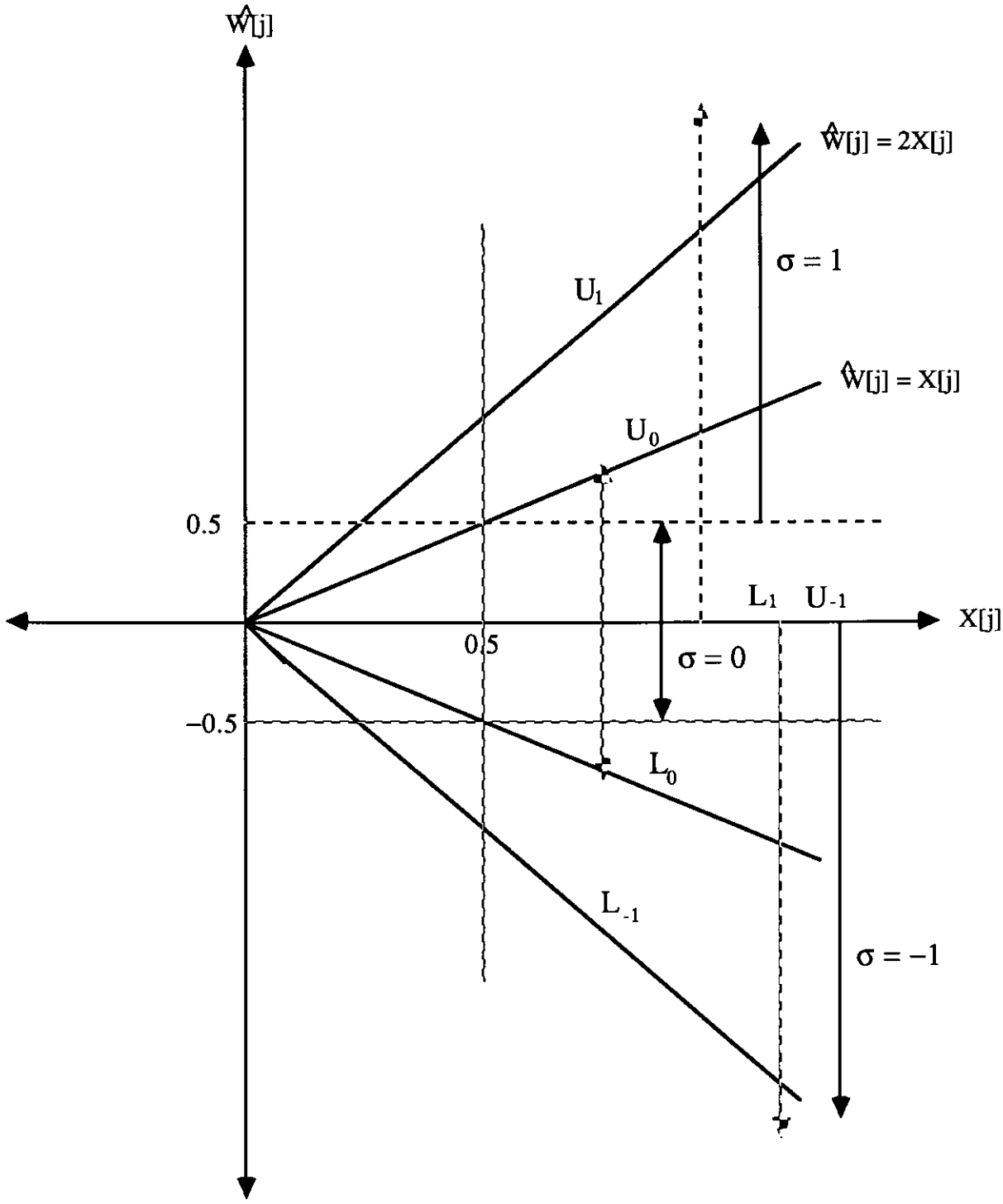


Figure 1. PD-graph

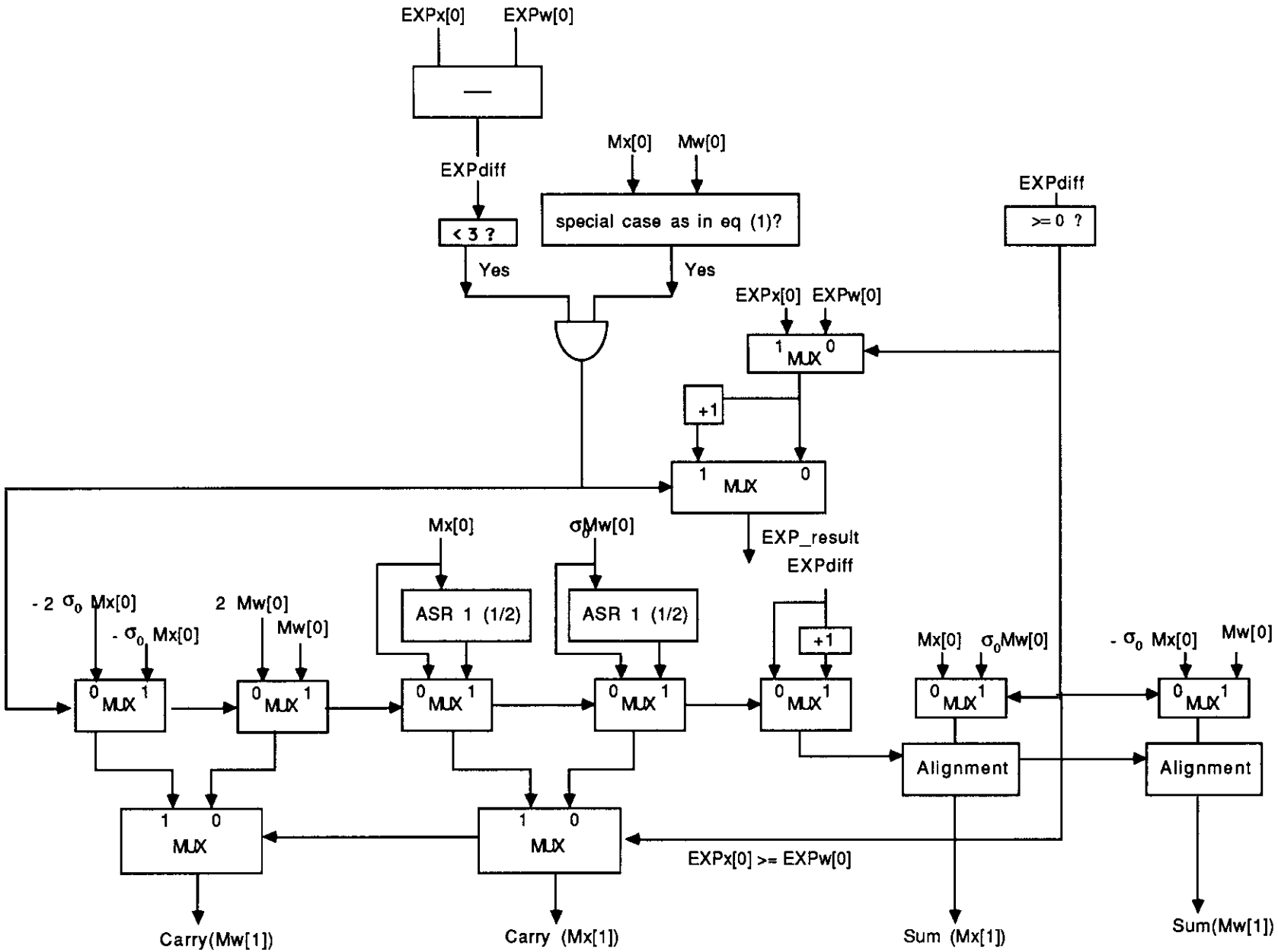
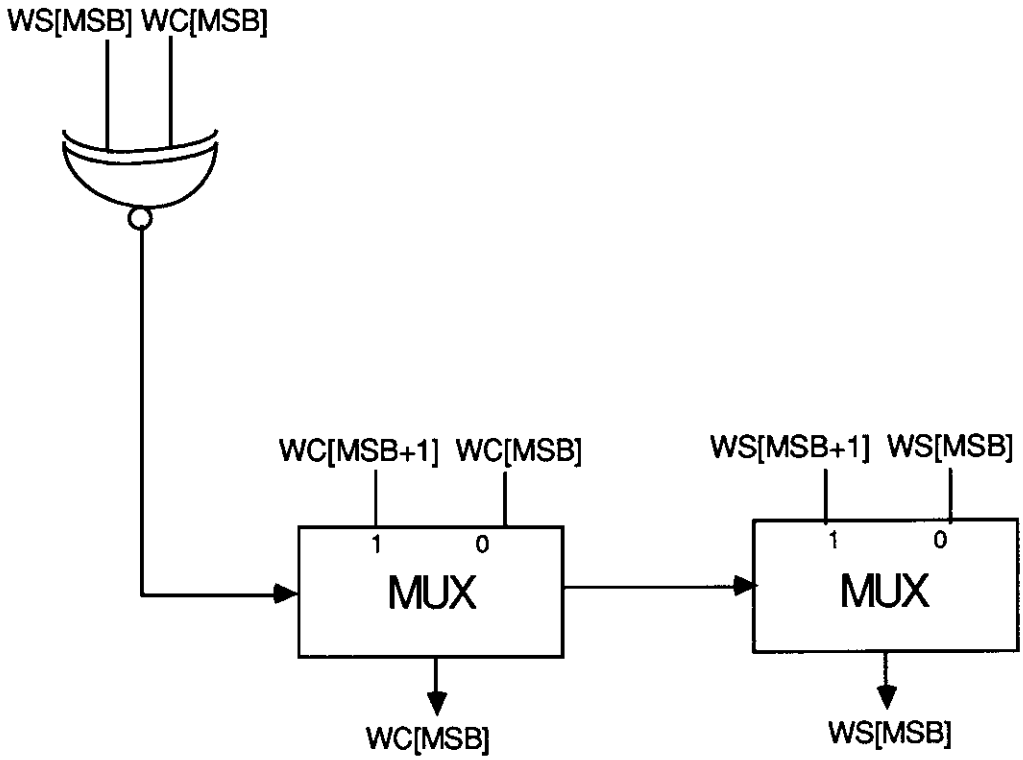


Figure 2. Pre-processing for angle unit



Remaing vectors WC[MSB+1 .. n-1] and WS[MSB+1 .. n-1] are unchanged

Figure 3. Block-level implementation for the remedy scheme

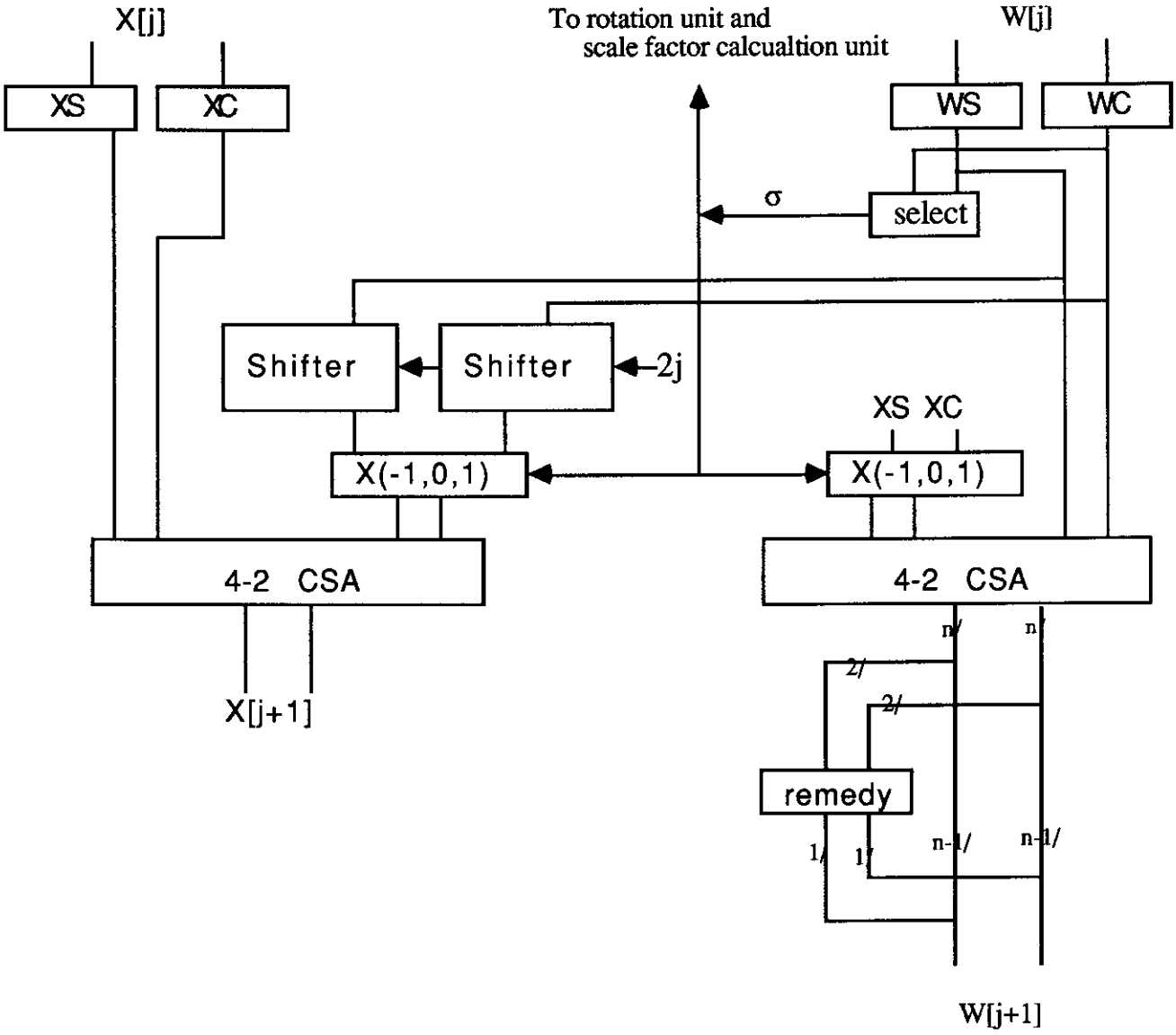
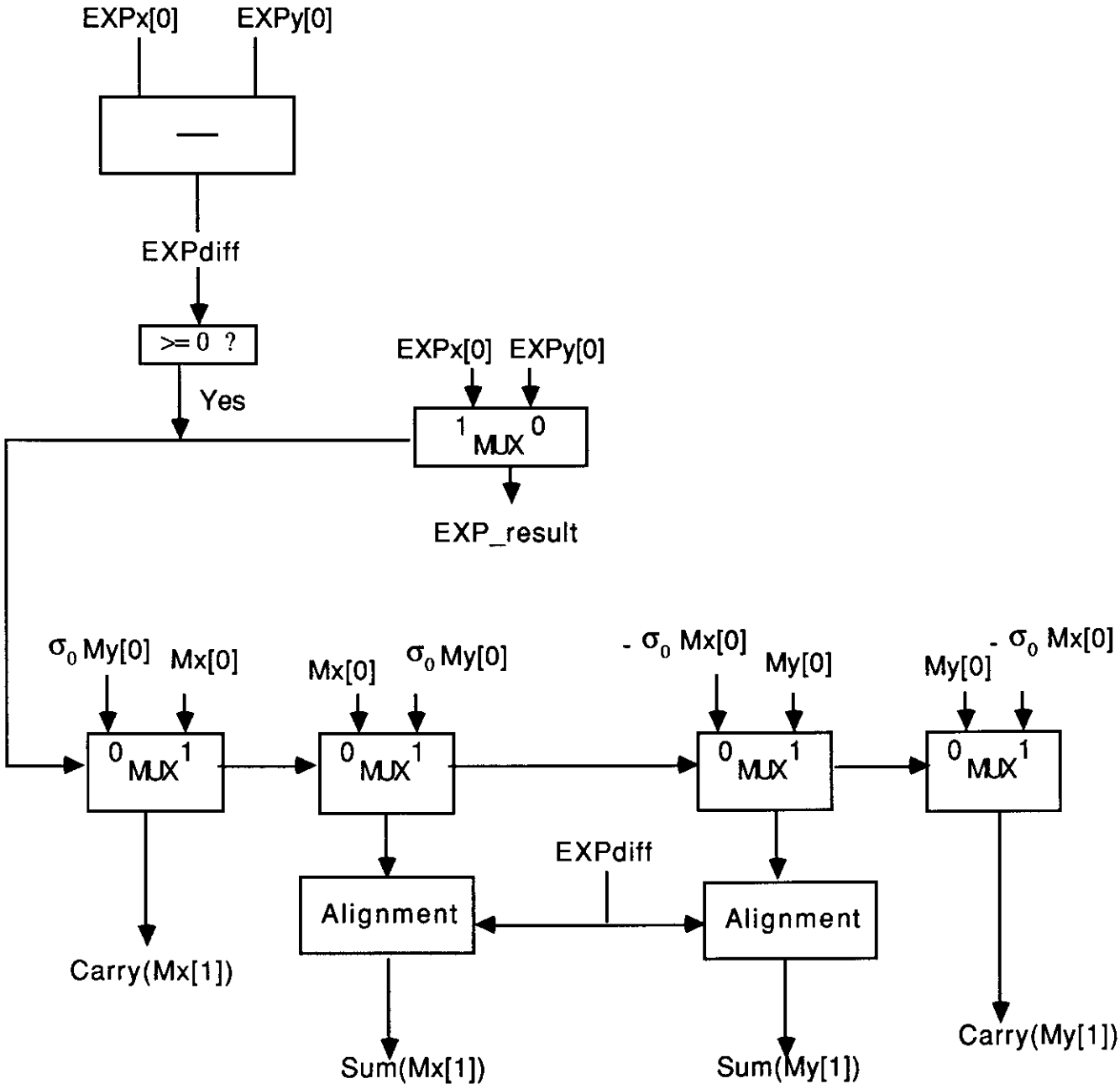


Figure 4 Block-level implementation of angle unit



Pre-processing for rotation unit

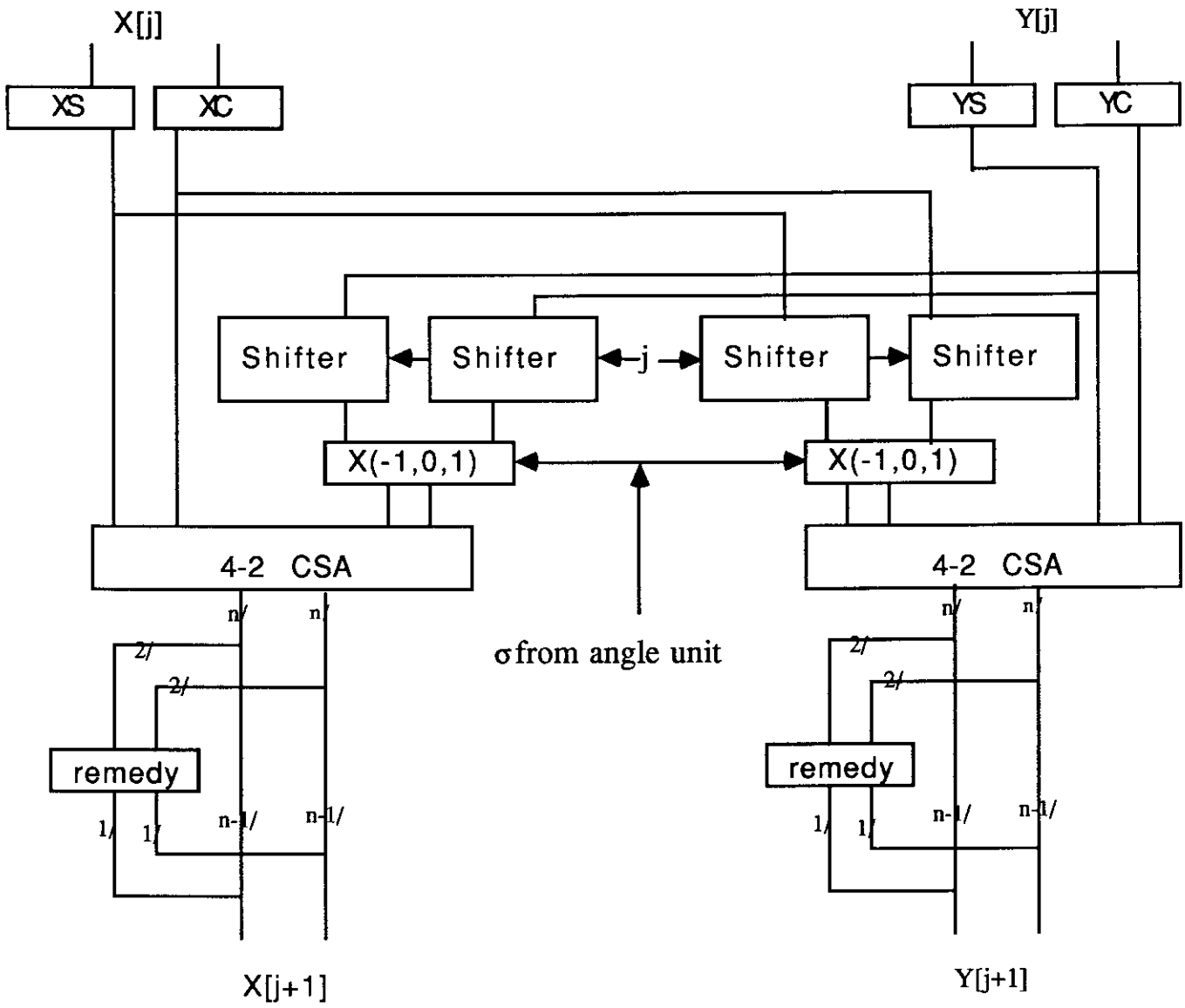


Figure 5 Block-level implementation of rotation unit

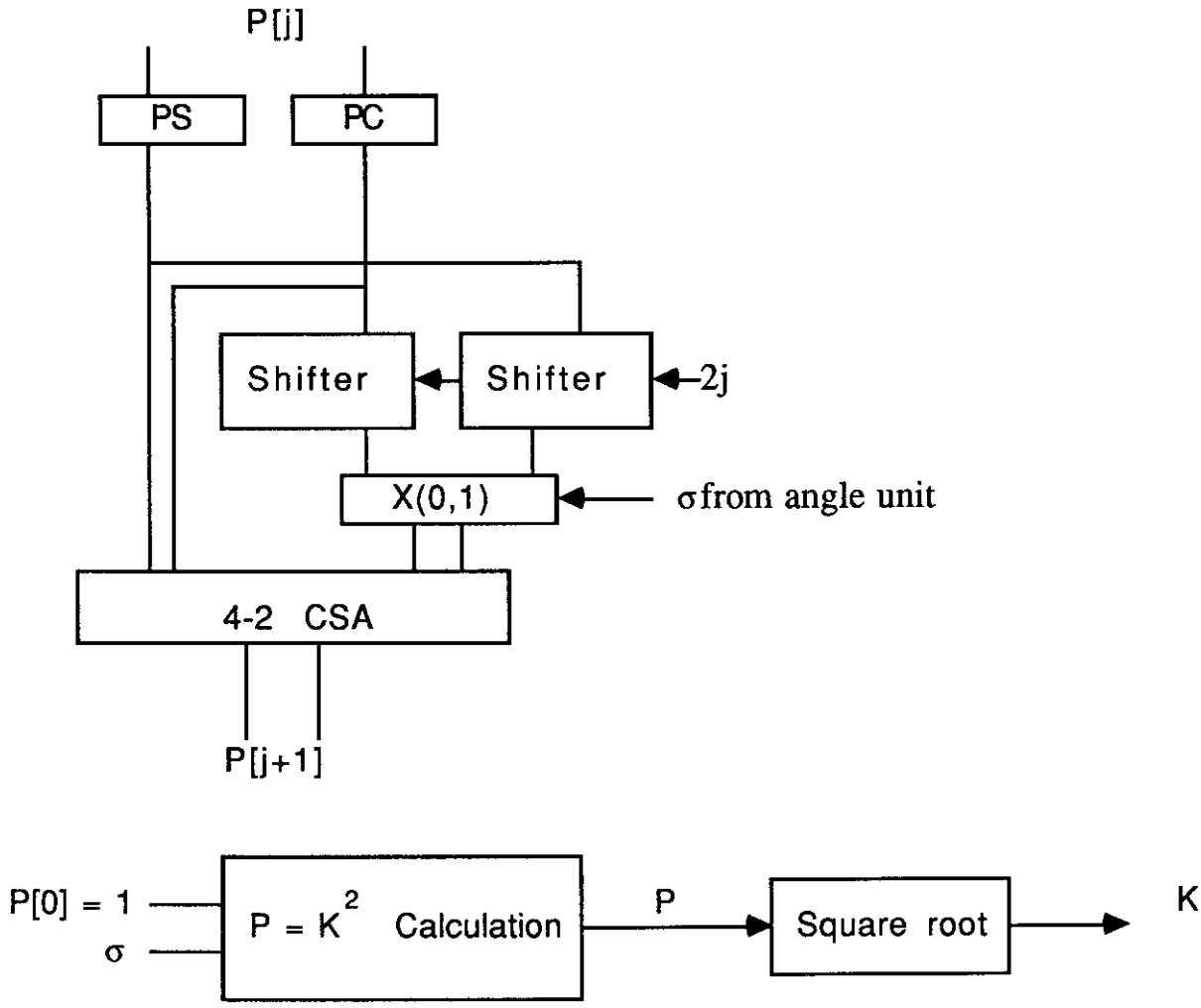


Figure 6. Block-level implementation of scale factor computation

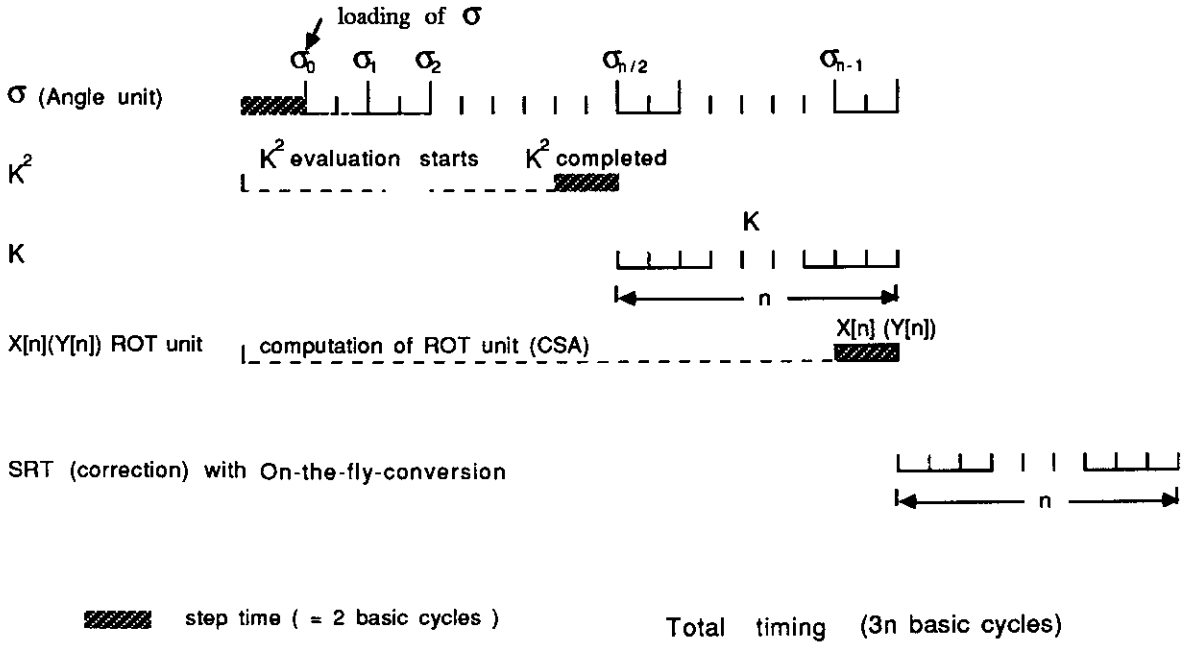


Figure 7. Overall system timing