# FAST TRIANGULARIZATION BY GIVENS ROTATION USING ON LINE CORDIC

Milos Ercegovac
Tomas Lang

September 1987
CSD-870045

# FAST TRIANGULARIZATION BY GIVENS ROTATION USING ON-LINE CORDIC

Miloš D. Ercegovac and Tomas Lang

UCLA Computer Science Department
University of California, Los Angeles

## Abstract

A scheme for triangularization of a matrix using redundant and on-line CORDIC modules is proposed. Its implementation is simpler and faster than implementations using conventional CORDIC modules. The proposed scheme has the following features: • the rotation processors use angles in a decomposed form thus eliminating the angle recurrence and allowing overlap between the angle processor and rotation processors; no ROMs are required; • the $(x,y)$-recurrences are transformed so that only one variable shifter is required; • the carry-propagate addition is replaced by a redundant addition (carry-save or signed-digit) thus reducing the clock cycle; • the rotation recurrences are unfolded and implemented in on-line manner thus replacing the variable shifter with simple delays; • the scale factor is computed in on-line manner; • the scheme uses efficiently floating-point representations.

## 1. Introduction

The triangularization of a matrix by Givens' rotations is discussed in [GOL83]. Traditionally, this transformation was proposed for execution in a sequential computer [SAME78]. Recently, interest has evolved in the computation using concurrent structures of processing elements, such as linear and triangular arrays, to improve the speed of execution [GENT81, CIMI81, AHME82a]. 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: no square roots [GENT73], on-line [CIMI81, ERCE87b], CORDIC [AHME82a]. In this paper we discuss methods to increase the speed of the CORDIC scheme, by eliminating the need of carry-propagate adders. Moreover, we use an on-line technique to reduce the need of shifters, to eliminate the use of ROMs and to allow the overlapping of the CORDIC rotation with the divisions required to compensate for the scaling factor. In addition, the proposed scheme uses efficiently floating-point representations.

### *Triangularization by Givens rotation*

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**

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

**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 [AHME82a] a linear and a triangular array are proposed to perform the triangularization. For instance, in the linear array, shown in Figure 1, the angle $\theta_{ri}$ 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 redundant (cary-save or signed-digit) 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

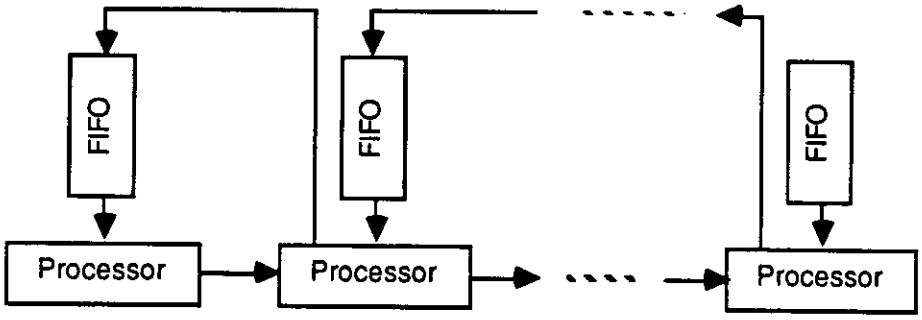


Figure 1: Linear Array of Processors
for Givens Rotations

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 suggested algorithm is done 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, of 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 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 large, 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. Making a carry-save addition the primitive operation that takes one clock cycle, we estimate that the CORDIC step takes around six clock cycles, resulting in an overall delay of $13.5n$ clock cycles.

To reduce the delay we present an implementation using redundant addition (carry-save or signed-digit [AVIZ61]) and on-line techniques [ERCE84, IRWI87].

## 2. Implementation Using Redundant and On-line CORDIC

The time of execution of the CORDIC operations can be reduced significantly by performing the CORDIC recurrence using a redundant addition (carry-save or signed-digit [AVIZ61]). Moreover, when possible, the implementation with on-line adders replaces the area-consuming shifters by delays, resulting in a more cost-effective implementation. We now describe and evaluate this alternative. Finally, the implementation using on-line techniques permits the overlapping of several operations [ERCE84, IRWI87].

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 $\theta_{ri}$ and the new diagonal element $a_{rr}$ by $n$ CORDIC steps [WALT71] of the form

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

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

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

with

$$x[0] = a_{rr}, \quad y[0] = a_{ir}$$

and the results being

$$a_{rr}(new) = x[n]/K, \qquad \theta_{ri} = z[n]$$

where

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

In conventional CORDIC, the value of $\sigma_j$ is obtained as

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

As mentioned before, this CORDIC operation is relatively slow because each of the $n$ steps requires a carry-propagate addition and a variable shift. Moreover, the use of a standard complete CORDIC is area consuming because of the carry-propagate adders, the shifters, and the storage of the constants $\tan^{-1}(2^{-j})$. We now develop a scheme that reduces these limitations by:

i) Elimination of the angle recurrence. This is possible because the rotation processors can use directly the $\sigma_j$'s. This reduces the area by eliminating one adder and the storage for the constants $\tan^{-1}(2^{-j})$. Moreover, it also eliminates these components from the rotation processors. In addition, with this modification the angle is transmitted in a digit-serial fashion ($\sigma_i$) between processors and the angle calculation and the rotations can be overlapped.

ii) Elimination of one of the shifters by modifying the recurrences as follows. Making

$$w[j] = 2^j y[j],$$

the recurrences are transformed to

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

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

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

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

iii) Replacing the carry-propagate addition by a redundant addition (carry-save or signed-digit). This requires that the determination of $\sigma_j$ 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 $\theta_{ri}$ in terms of the $\sigma_j$'s. This is achieved by allowing $\sigma_j$ to take values from the set $\{-1,0,1\}$ instead of from the set $\{-1,1\}$, which is the one used in conventional CORDIC.

Selection functions for $\sigma_j$ using this redundant set are developed in the Appendix for 2's complement carry-save representation as well as for the signed-digit case. For the carry-save case the resulting function is

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

Similarly, for the signed-digit case we get,

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

Any of the two redundant adders can be used. Here we illustrate the carry-save case. The implementation of the corresponding recurrence is shown in Figure 2a. The step time corresponds to the shifter delay plus the 4-2 carry-save adder, since the selection function is overlapped with the shifting. The angle is transmitted in a digit-serial fashion ($\sigma_i$) to the rotation processor.

For the computation of the new diagonal element, it is necessary compensate $x[n]$ for the scaling factor $K$. The value of this scaling factor is

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

Since in this case the set of values of $\sigma_j$ is $\{0,1,-1\}$ (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 the method proposed in [DELO83] depends 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

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

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 2b. Note that only $n/2$ stages are needed.

ii) Compute $K = P^{1/2}$ by an on-line square-root algorithm [ERCE78]. The on-line delay of this module is four clock cycles. After that, an on-line division unit computes $a_{rr} = x[n]/K$. The on-line divider incorporates the on-the-fly conversion from a signed-digit representation to a conventional 2's complement representation [ERCE87a].

Consequently, the angle processor consists of the CORDIC recurrences, the on-line recurrence for $K^2$, the on-line square-root unit, and the on-line divider unit, as shown in Figure 3a.

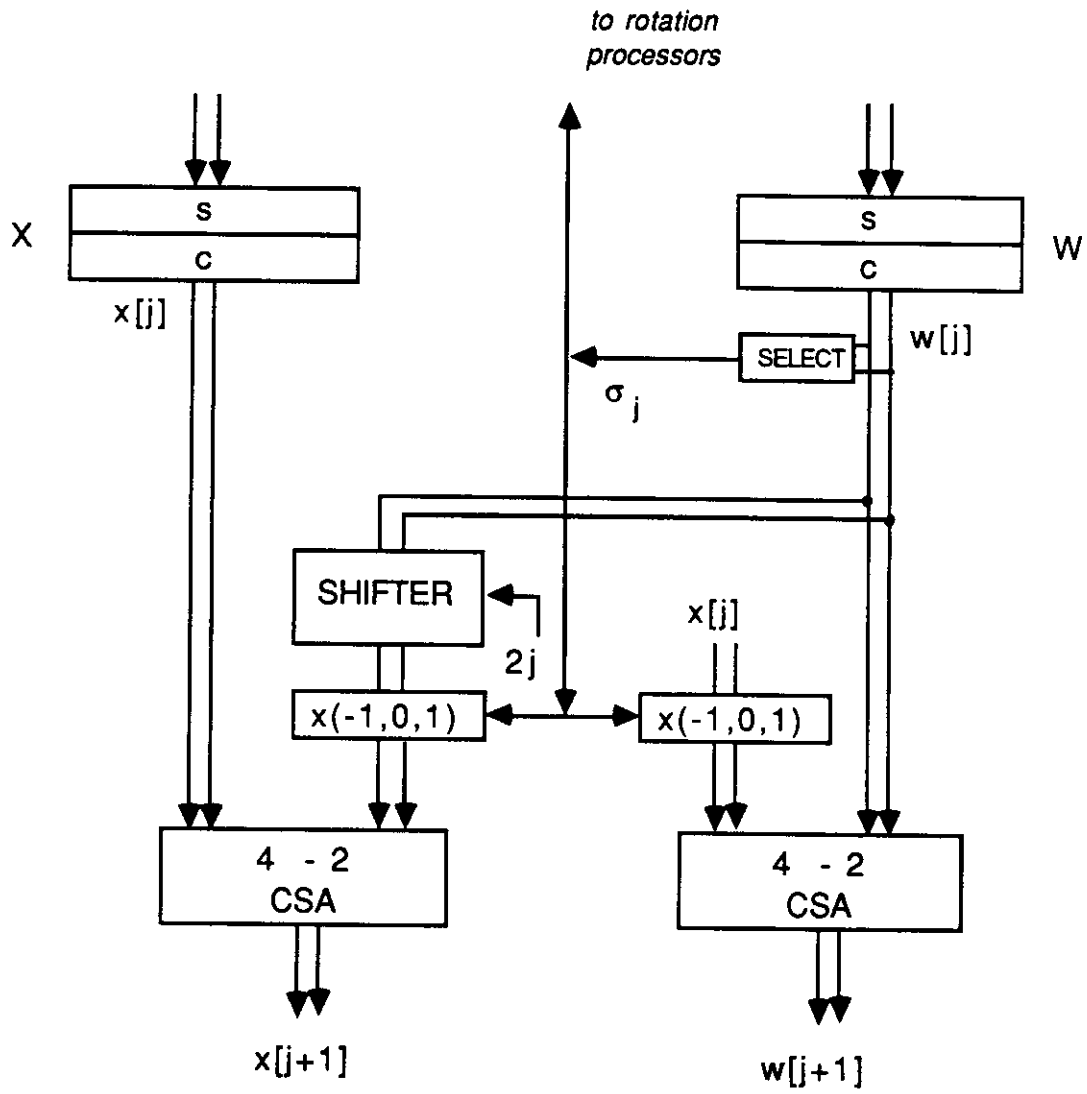


Figure 2a: Implementation of x,w Recurrences

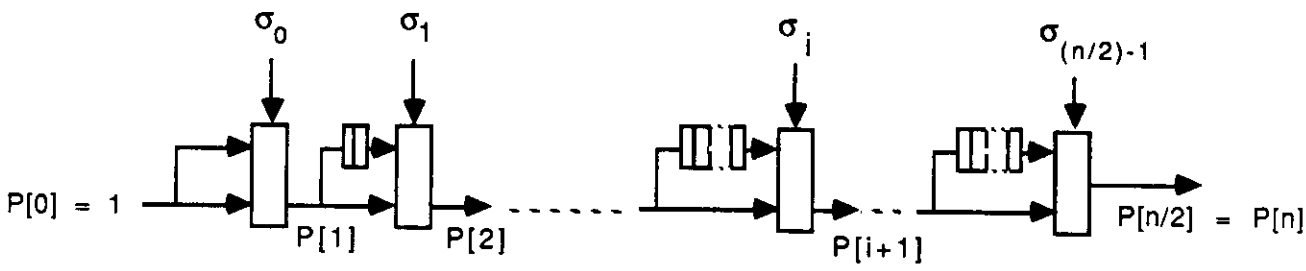


Figure 2b: Implementation of Scaling-Factor Recurrence

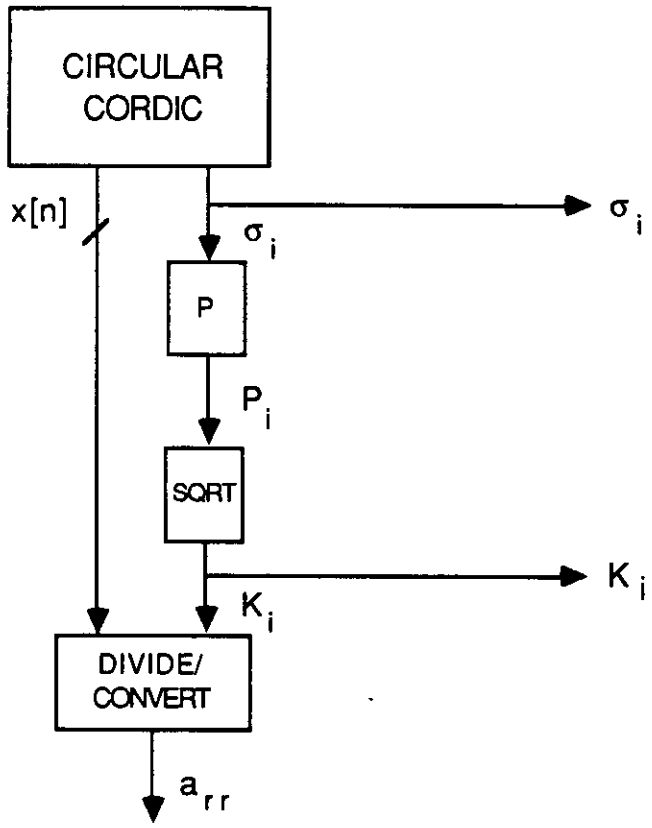


Figure 3a: Angle Processor Organization

The timing is shown in Figure 3b. 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. The total delay is of $2n+9$ clock cycles.

An alternative implementation of the CORDIC recurrences would be to use on-line addition instead of parallel carry-save 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 4. However, we do not favor this implementation because the selection function of $\sigma_j$ requires that three bits of $w[j]$ be computed before $\sigma_j$ can be determined. This increases to five the number of cycles between the initiation of consecutive iterations, which makes this implementation significantly slower than the parallel one. A faster implementation of this alternative is being developed.

*On-line rotation processor*

The rotation processor performs the rotation by means of a circular CORDIC operation. It uses the angle produced by the angle processor. Since this angle is passed to the rotation processor in its circular decomposed form (that is, the $\sigma_j$'s are passed), it can be used directly in this form so that the rotation processor does not require an angle recurrence. 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 angle recurrence step, the rotations have 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 5a. 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 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 cycles. Since, as mentioned before, the CORDIC recurrence of the angle processor has a step time of two clock cycles, this time between initiations matches the time at which the corresponding $\sigma_j$ is obtained.

The division by the scaling factor $K$ is done by two on-line division units (Figure 5a). The timing is shown in Figure 5b. The total delay of rotation is $3n+3$ clock cycles.

*Overall timing*

The overall timing is shown in Figure 6. 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 CORDIC of the angle processor is of two clock cycles, while for the other recurrences it is of one clock cycle.

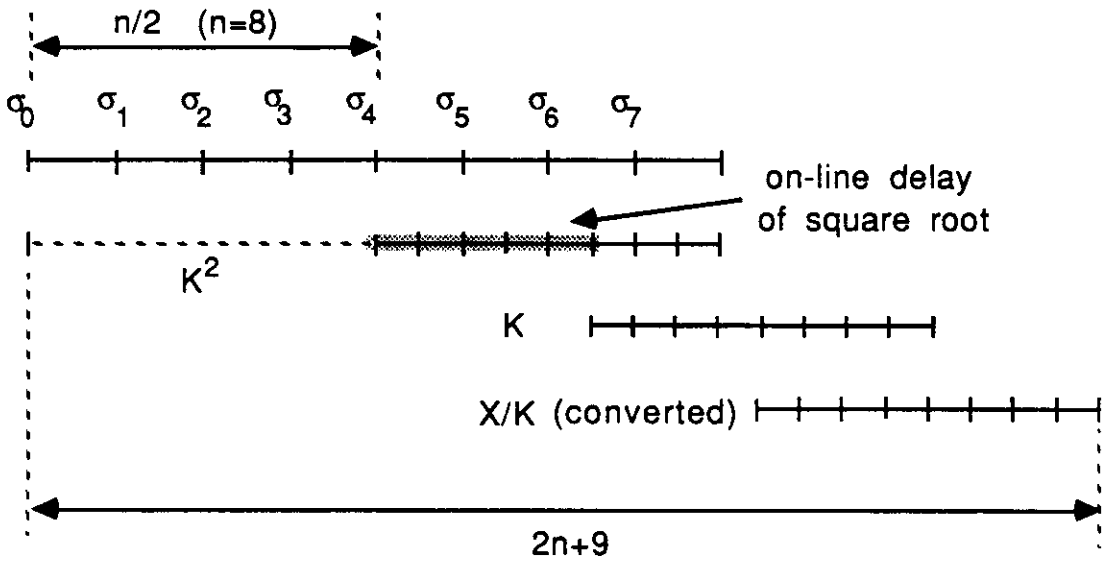


Figure 3b: Timing of the Angle Processor

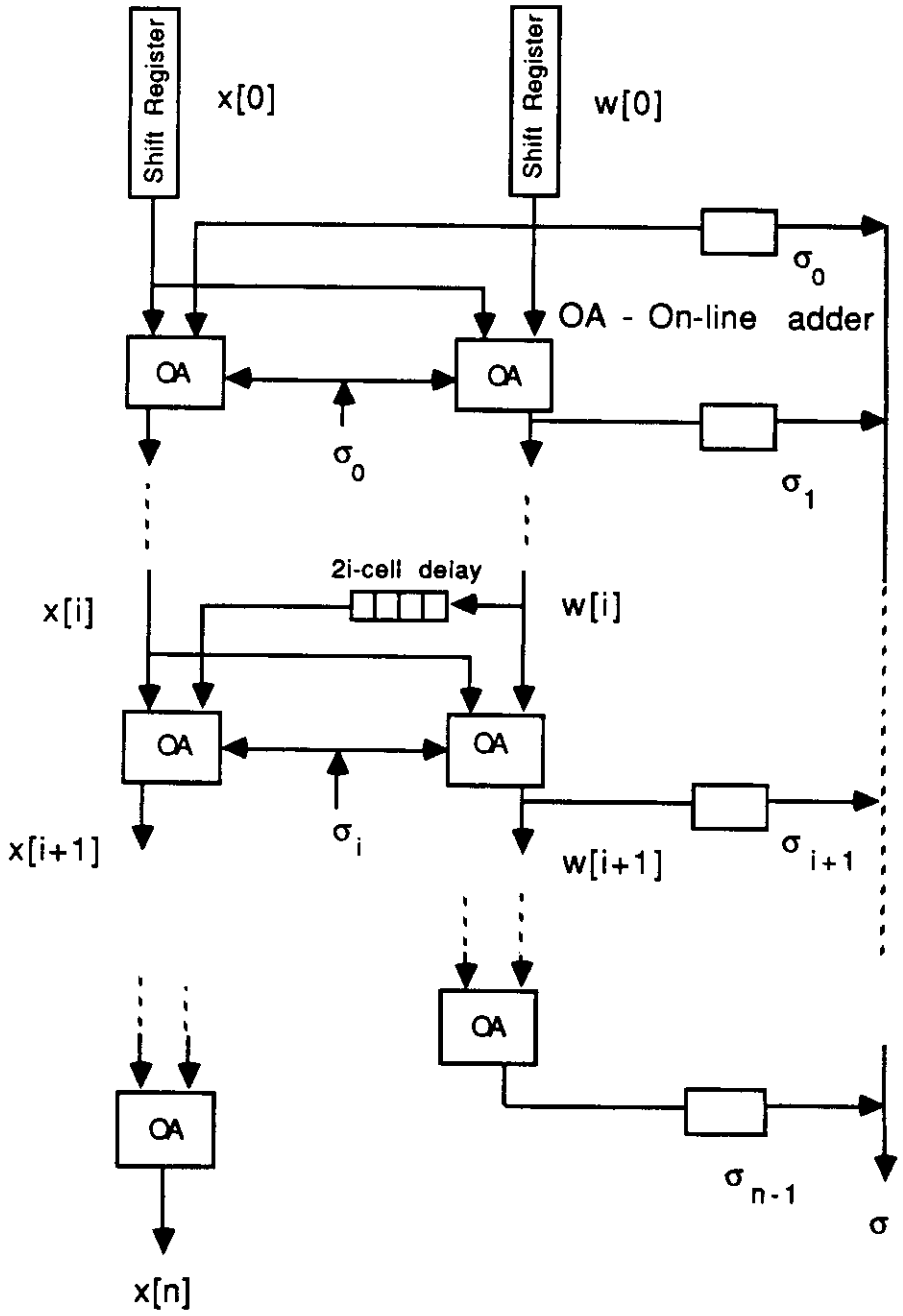


Figure 4. On-Line Alternative
for Angle Computation

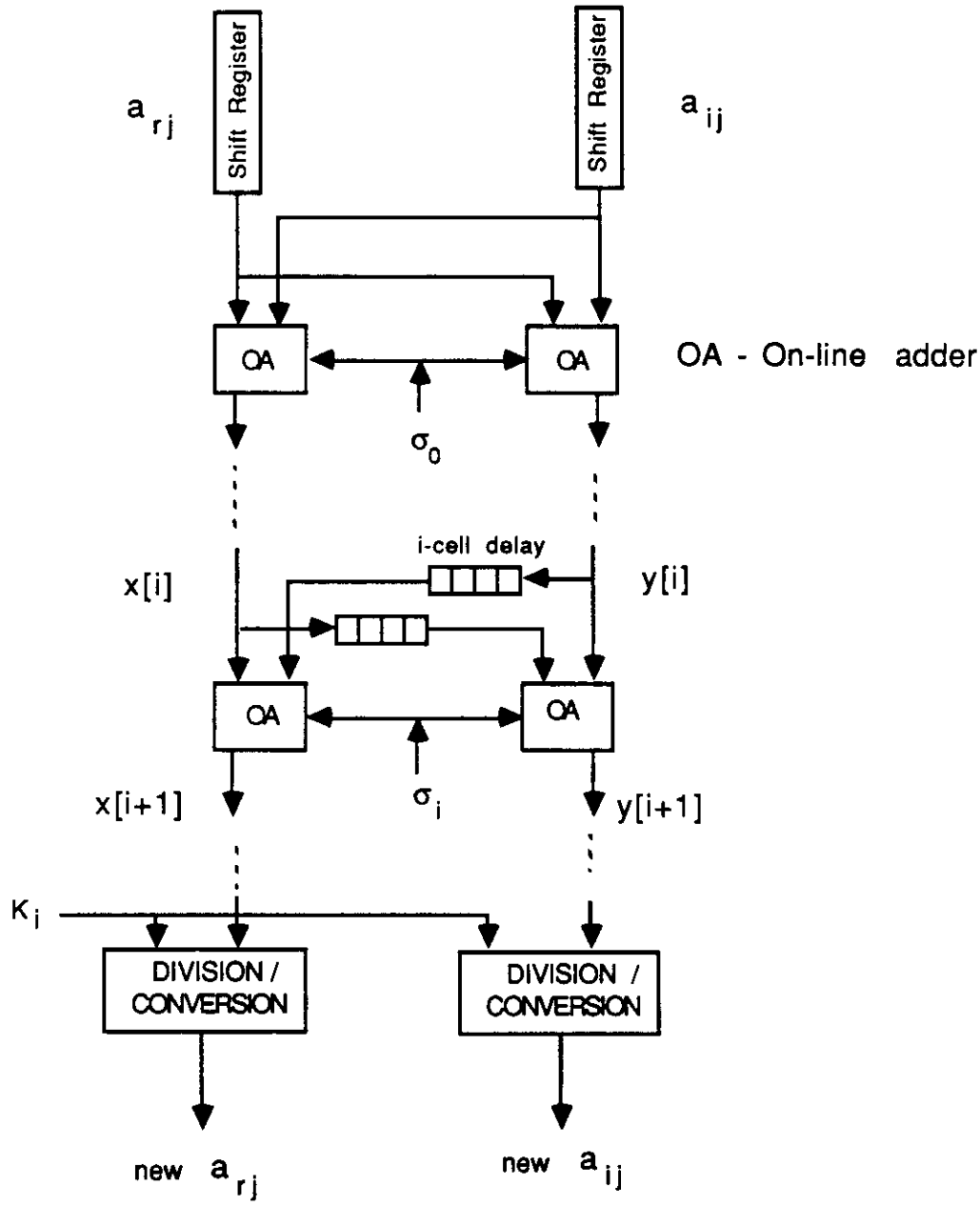


Figure 5a: On-Line Rotation

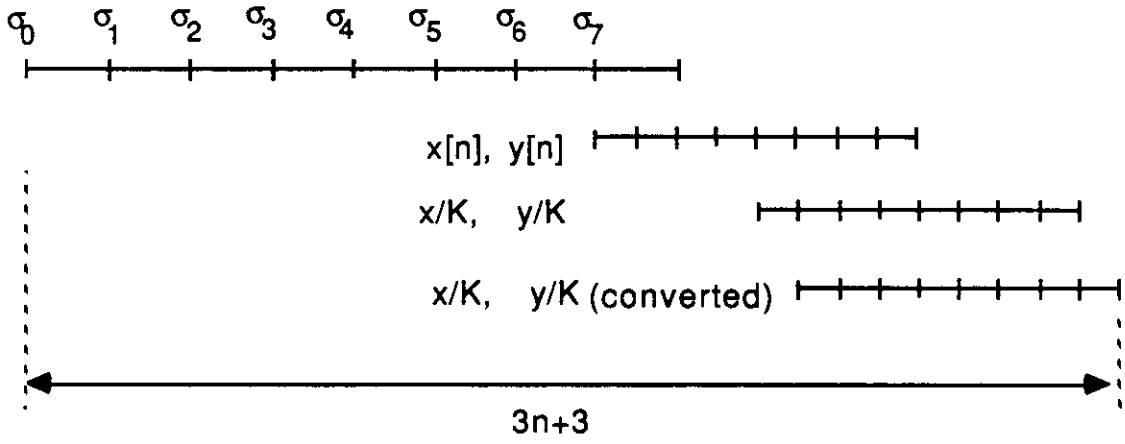


Figure 5b: Timing of the Rotation Processor

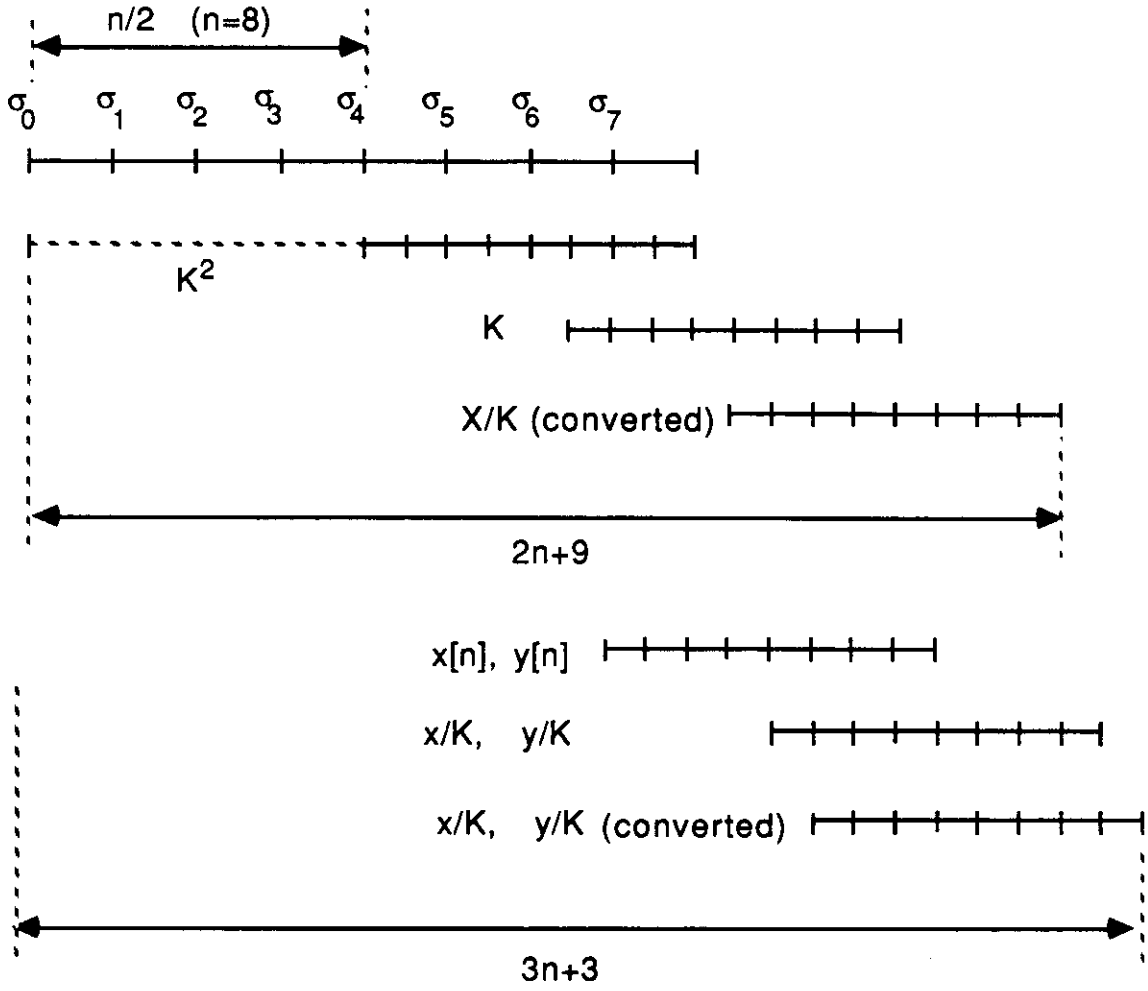


Figure 6: Overall Timing

The time of the scheme presented here compares favorably with that of the scheme using conventional CORDIC, as proposed in [AHME82a]. In that scheme, 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 schemes), 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.

## 3. 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 of the alignment requirements. We now develop a simpler alternative adapted to the triangularization case.

*Angle and new diagonal element*

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

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

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

To perform the CORDIC transformations we first align these initial value. However, since $y[0]$ is positive, the value of $\sigma_0$ is 1 and we get

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

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

These can be written as

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

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

with $x[1]$ normalized.

This indicates that the CORDIC iterations can be done by performing first a floating-point addition (normalized) and a floating-point subtraction (not normalized) 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

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

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.

To obtain the new diagonal element it is necessary to multiply by $2^{a}$.

*Rotation*

The rotation can be done in a similar way to the computation of the angle. Let the initial values be

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

and

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

Again, since $\sigma_0$=1, a floating-point addition and a subtraction produces

$$x[1] = B^{+} \cdot 2^{b} \quad \textit{normalized}$$

$$y[1] = B^{-} \cdot 2^{b} \quad \textit{non normalized}$$

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

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

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

## 4. Summary

We have presented a scheme to perform the triangularization of a matrix using redundant and on-line CORDIC modules. We estimate that this results in a speed-up of approximately 4.5 with respect to the implementation using conventional CORDIC. Moreover, reductions in area result because of the elimination of the angle recurrence in the CORDIC modules, the replacement of shifters by delays in the rotation processor, and the digit-serial transmission of the angle between processors.

*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.

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

[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.

[ERCE78] M.D. Ercegovac, "An On-Line Square-Root Algorithm," Proc. 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-the-fly Conversion of Redundant into Conventional Representations," IEEE Transactions on Computers, vol. C-36, pp. 895-897, July 1987

[ERCE87b] M.D. Ercegovac and T. Lang, "On-Line Scheme for Computing Rotation Factors," Proc. 8th Symposium on Computer Arithmetic, pp. 196-203, 1987.

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

[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 Elementary Functions," AFIPS Spring Joint Computer Conf., pp. 379-385, 1971.

## Appendix

We now develop the selection function for $\sigma_j$ 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

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

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 on 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

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

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

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

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

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

These relations are illustrated in Figure A1a. 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:

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

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

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

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

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

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

The following table gives the binary description of this selection function:

| $\hat{w}$ | $\hat{w}_1\hat{w}_0\hat{w}_{-1}$ | $\sigma$ |
|---|---|---|
| 3/2 | 011 | 1 |
| 1 | 010 | 1 |
| 1/2 | 001 | 1 |
| 0 | 000 | 1 |
| -1/2 | 111 | 0 |
| -1 | 110 | -1 |
| -3/2 | 101 | -1 |

The corresponding switching expressions are:

$$(\sigma = 1) = \hat{w}_1'$$

$$(\sigma = 0) = \hat{w}_1\hat{w}_0\hat{w}_{-1}$$

$$(\sigma = -1) = \hat{w}_1(\hat{w}_0' + \hat{w}_{-1}')$$

ii) Signed-digit case

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

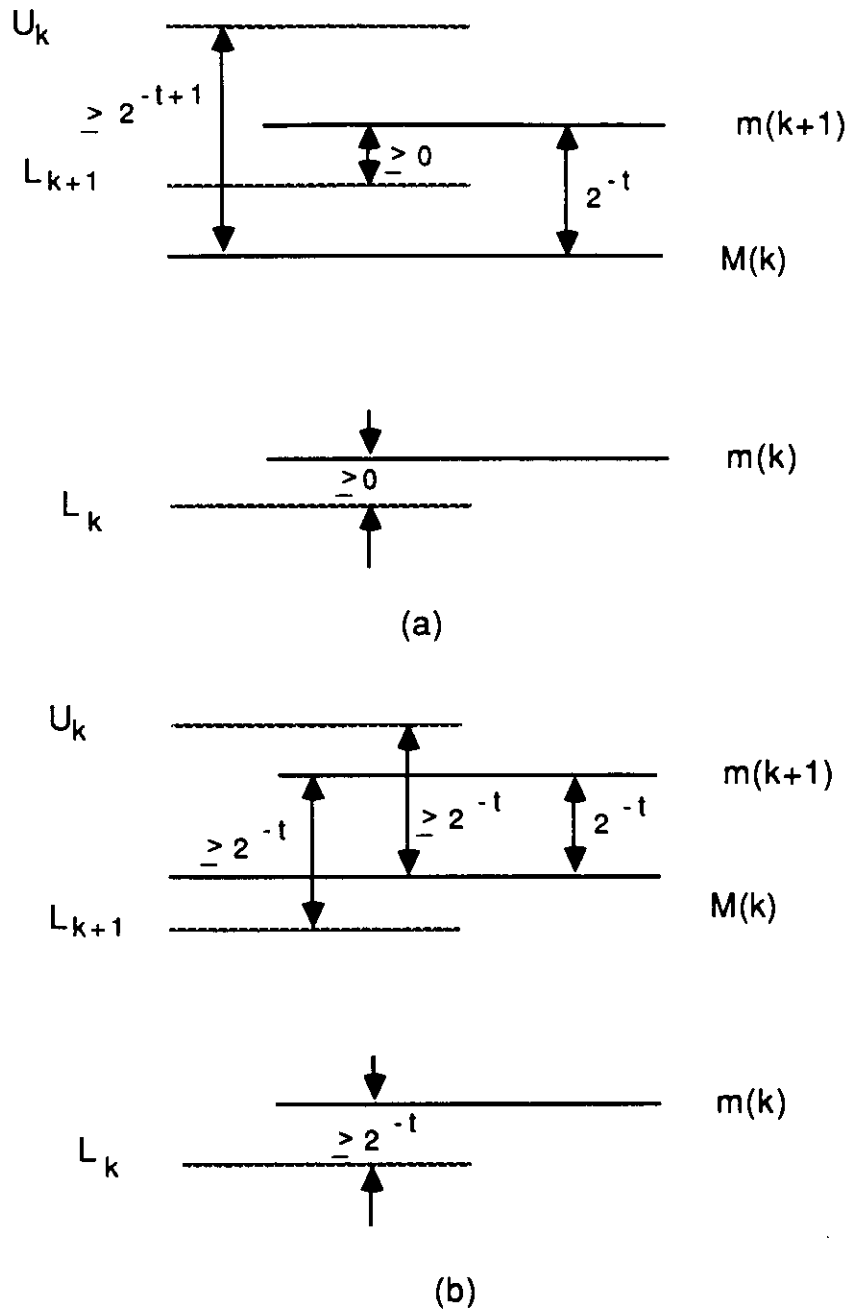


Figure A1: Bounds for Selection of q=k :
a - Carry-Save Case;
b - Signed-Digit Case