On-Line Scheme for Computing Rotation Factors

Miloš D. Ercegovac and Tomas Lang

UCLA Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

An integrated radix-2 on-line algorithm for computing rotation factors for matrix transformations is presented. The inputs and outputs are in parallel form, conventional 2's complement, floating-point representation. The exponents are computed using conventional arithmetic while the significands are processed using on-line algorithms. The conventional result is obtained by using an on-the-fly conversion scheme. The rotation factors are computed in $9+n$ clock cycles for n-bit significands. The clock period is kept small by the use of carry-save adder schemes. The implementation and performance of the algorithm are discussed.

1. Introduction

The rotation factors that we propose to compute are an essential part of matrix triangularization methods [GOL83] and the QR-decomposition [LUK86]. In particular, in systolic arrays in which the rotations are done in parallel, the time of computation of the rotation factors is critical since it determines the step time of the array. We present an implementation using on-line arithmetic [ERC84], since this approach is potentially advantageous in evaluating arithmetic expressions consisting of sequentially-dependent operations [ERC81]. The on-line scheme for the computation of rotation factors was proposed previously in [CIM81], where it is shown that it can produce a significant speed-up with respect to the use of conventional algorithms. However, the description is given at high-level and using previously developed algorithms for the operations of multiplication, division, and square root [TRI77, ERC78]. In this paper we develop an integrated algorithm for the rotation factors, which consists of modifications of the algorithms of the primitive operations to provide for suitable interfaces with their predecessors and successors in the computation. In this way, both the on-line delay and the complexity of the implementation are reduced. The clock period is kept low by the use of carry-save adder techniques and the conversion of the result from redundant to conventional representation is done using an on-the-fly scheme [ERC85]. Moreover, the scheme is for floating-point representations.

We propose to compute the rotation factors of the Givens matrix transformation [GOL83], defined as

$$c = \frac{x}{(x^2+y^2)^{1/2}} \tag{1.1}$$

and

$$s = \frac{y}{(x^2+y^2)^{1/2}}$$

We assume that $x = fx \cdot 2^{ex}$, $y = fy \cdot 2^{ey} \geq 0$ (due to the characteristics of the transformation [GOL83]), $c = fc \cdot 2^{ec}$ and $s = fs \cdot 2^{es}$ are normalized floating-point numbers with n-bit fractions in two's complement bit-parallel form. The exponents are represented and processed in a conventional, bit-parallel manner and on-line representations and algorithms are used internally to compute the fractions.

The scheme, shown in Figure 1, performs the following functions:

1. Alignment of operands x and y
2. Computation of $z = x^2 + y^2$
3. Computation of $d = z^{1/2}$
4. Computation of $c = x/d$ and $s = y/d$

We now describe each component separately and then comment on the whole system.

2. Alignment

The alignment of the input operands is performed in on-line manner during the parallel-to-serial conversion according to following algorithm:

Algorithm *Align*

/* Get exponent difference and set common exponent */
$ed = ex - ey$; $e = max(ex, ey)$;
$x_0 = sign(x)$; $y_0 = 0$; /* since $y > 0$ */

/* Delay operand with smaller exponent */
for $i = 1,2,...,n$

if $ed \geq 0$ **then**

$\{x_i = fx_i;$
if $i \leq ed$ **then** $y_i = 0;$
else $y_i = fy_{i-|ed|};\ \}$

else

$\{y_i = fy_i;$
if $i \leq |ed|$ **then** $x_i = sign(x);$
else $x_i = fx_{i-|ed|};\ \}$

end *Align*

The alignment scheme is shown in Figure 2. The exponent difference is obtained in one clock cycle. Thereafter, the bits of the aligned operands are obtained one per cycle. The on-line delay $p_{align} = 1$.

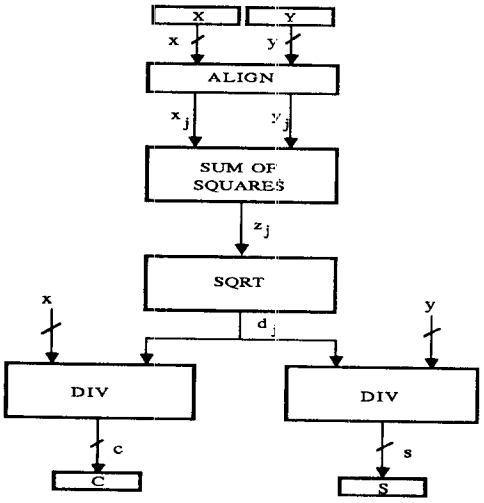

Figure 1. General Scheme

3. On-line Sum of Squares Algorithm

In this section we present the algorithm and implementation for on-line sum of squares to be used in the computation of the rotation factors. The algorithm computes the fraction and the exponent of the result. The exponent is computed in a conventional, bit-parallel manner (see Section 6). Consequently, the following discussion deals with the computation of the fraction.

To incorporate a possible on-line delay we compute

$$z^* = 2^{-p} z = 2^{-p}(x^2 + y^2) \tag{3.1}$$

If the on-line forms of x, y, and z^* are

$$X[j] = X[j-1] + x_j 2^{-j}, \quad X[0] = -x_0 \tag{3.2}$$

(since x is in 2's complement)

$$Y[j] = Y[j-1] + y_j 2^{-j}, \quad Y[0] = 0 \quad (\text{since } y > 0)$$

$$Z[j] = Z[j-1] + z_j 2^{-j}, \quad Z[0] = 0$$

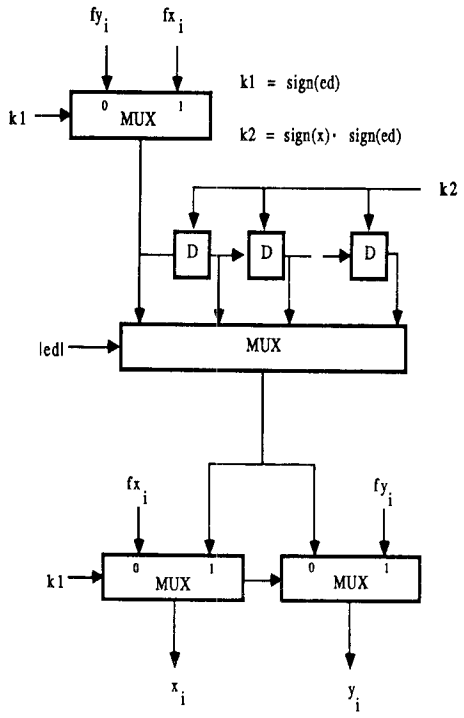

Figure 2. Alignment Scheme

then we can define the residual function

$$w[j] = 2^j (2^{-p} X[j]^2 + 2^{-p} Y[j]^2 - Z[j]) \tag{3.3}$$

From this residual expression we get the recurrence

$$w[j] = 2w[j-1] - z_j + (2X[j-1]x_j + x_j^2 2^{-j})2^{-p} + (2Y[j-1]y_j + y_j^2 2^{-j})2^{-p} \tag{3.5}$$

with the initial condition

$$w[0] = (X[0]^2 + Y[0]^2)2^{-p} = x_0 2^{-p} \tag{3.6}$$

To keep $w[j]$ bounded we make the selection

$$z_j = integer(2w[j-1]) \tag{3.7}$$

which transforms the recurrence into

$$w[j] = fraction(2w[j-1]) + (2X[j-1]x_j + x_j^2 2^{-j})2^{-p} + (2Y[j-1]y_j + y_j^2 2^{-j})2^{-p} \tag{3.8}$$

Since we want to implement this with a carry-save adder, the exact integer and fraction parts cannot be determined (without propagating carries). Consequently, the previous expressions are transformed into

$$z_j = csint(2w[j-1]) \tag{3.9}$$

$$w[j] = csfrac(2w[j-1]) \tag{3.10}$$

$$+ (2X[j-1]x_j + x_j^2 2^{-j})2^{-p}$$

$$+ (2Y[j-1]y_j + y_j^2 2^{-j})2^{-p}$$

Since $x_i = 0$ and $y_i = 0$ for $i > n$, we have

$$w[n+1+p] < 1$$

so that

$$|(x^2+y^2) - Z[n+1+p]| < 2^{-n}$$

The corresponding carry-save operation is shown in Figure 3. From it we see that the bounds on z_j are (for $|x|, |y| < 1$ and $y \geq 0$)

$$-4 \leq z_j \leq 10 \quad \textbf{if} \quad p=0 \tag{3.11}$$

$$-2 \leq z_j \leq 6 \quad \textbf{if} \quad p=1$$

These values of z_j are over-redundant (that is, they are larger than 1 for a radix-2 representation). We choose this alternative because it simplifies the selection, reduces the on-line delay, and is suitable for the interface with the square root. We choose to implement the case with $p=1$ to minimize the overall on-line delay.

The algorithm is summarized next:

Algorithm *Sum of Squares*

/* Initialization */
$w[0] \leftarrow x_0 \times 2^{-1}$; $z_0 \leftarrow 0$;
$X[0] \leftarrow -x_0$; $Y[0] \leftarrow 0$;

/* Recurrence */
for $j=1,2,\ldots,n+1$
{ $w[j] \leftarrow csfract(2w[j-1])$
$+ (X[j-1]+x_j 2^{-j-1})x_j$
$+(Y[j-1]+y_j 2^{-j-1})y_j$;
$z_j \leftarrow csint(2w[j-1])$;
$X[j] \leftarrow concat(X[j-1], x_j)$;
$Y[j] \leftarrow concat(Y[j-1], y_j)$; }

end *Sum of Squares*

The exponent of the result is $2e$ and the fraction is in the range $0.25 \leq z^2 \leq 2$.

Implementation

The scheme is shown in Figure 4. Note that its implementation complexity is similar to that of the on-line multiplier. The on-line delay is $p_{ss}=1$. The critical path consists of two multiplexers, one 4-to-2 carry-save adder, and a 4-bit CPA. The internal arithmetic is in 2's complement.

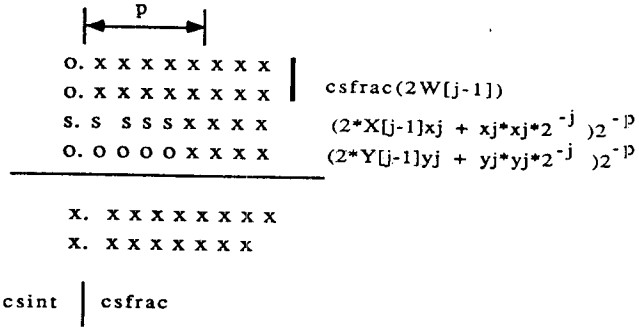

Figure 3. CSA Operation in Sum of Squares

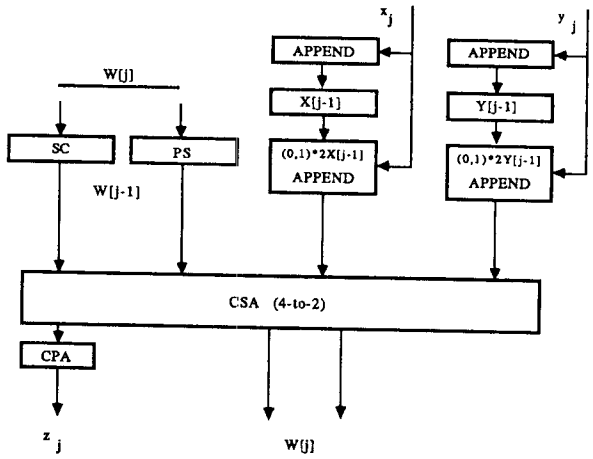

Figure 4. Sum of Squares Scheme

4. On-Line Square Root

We now describe an on-line square-root algorithm to interface with the sum of squares and the division, for the computation of rotation factors. This algorithm allows an over-redundant input digit set, differing in this respect from previous on-line square root algorithms [ERC78, OKL82].

Let the on-line forms of the argument z and of the square root d be

$$Z[i] = Z[i-1] + z_i 2^{-i}, \quad -b \leq z_i \leq a \tag{4.1}$$

(where, according to the sum of squares implementation, $b=2$ and $a=6$)

$$D[i] = D[i-1] + d_i 2^{-i}, \quad d_i \in \{-1,0,1\}$$

To bound the error of the computation we make

$$(D[i]-2^{-i})^2 + b\,2^{-p-i} \leq Z[i+p] \leq (D[i]+2^{-i})^2 - a\,2^{-p-i}$$

where p is the on-line delay to be determined later and, as indicated, the argument digits satisfy $z_i \in \{-b,\ldots,0,\ldots,a\}$.

We can define a residual $R[i]$ such that

$$R[i] = 2^i (Z[i+p] - D[i]^2) \qquad R[0] = D[p] \tag{4.2}$$

satisfying the bounds $C_{-1}[i] \le R[i] \le C_1[i]$ where

$$C_{-1}[i] = (-2D[i] + 2^{-i} + b\,2^{-p}) \tag{4.3}$$

$$C_1[i] = (2D[i] + 2^{-i} - a\,2^{-p})$$

The resulting recurrence is

$$R[i] = 2R[i-1] + z_{i+p}2^{-p} - 2d_i D[i-1] - d_i^2 2^{-i}$$

A selection function for d_i is derived in Appendix A. Assuming $t \ge 3$ fractional bits in the estimate of the remainder and the on-line delay $p_{sqr} = 4$, we obtain for the selection constants

$$k_{01} = 1/8 \quad \text{and} \quad k_{\bar{1}0} = -3/8$$

The corresponding algorithm for on-line square root is summarized next.

Algorithm *Sqrt*

/* Initialization */

$R[-4] \leftarrow 0;\ D[0] \leftarrow 0;\ d_0 \leftarrow 0;$

/* Accumulate 4 input digits */

for i=-3,-2,-1,0

$$\{R[i] \leftarrow 2R[i-1] + z_{4+i}2^{-4};\}$$

/*Recurrence*/

for i=1,2,...,n+4

$$\{\hat{R}^*[i] = \hat{R}[i-1] + z_{4+i}2^{-5};$$

$$d_i = \begin{cases} 1 & \text{if } \hat{R}^*[i] > 1/8 \\ 0 & \text{if } 1/8 \ge \hat{R}^*[i] \ge -3/8 \\ -1 & \text{if } \hat{R}^*[i] < -3/8 \end{cases}$$

$$R[i] \leftarrow 2R[i-1] + z_{4+i}2^{-4} - 2d_i D[i-1] - d_i^2 2^{-i};$$
$$D[i] \leftarrow (D[i-1], d_i);\}$$

end *Sqrt*

The result significand is in the range $[1/2, 2^{1/2}]$ and the exponent is e.

Implementation

The square root scheme is shown in Figure 5. The internal arithmetic is performed in 2's complement system. The signed-digit operands are converted to 2's complement using the on-the-fly conversion method [ERC85]. To avoid costly sign-extension to subtract $d_i^2 2^{-i}$, we assume a borrow-save subtractor. The critical path, shown in Figure 6, consists of a 5-bit CPA, one multiplexer , and a 3-to-2 borrow-save subtractor.

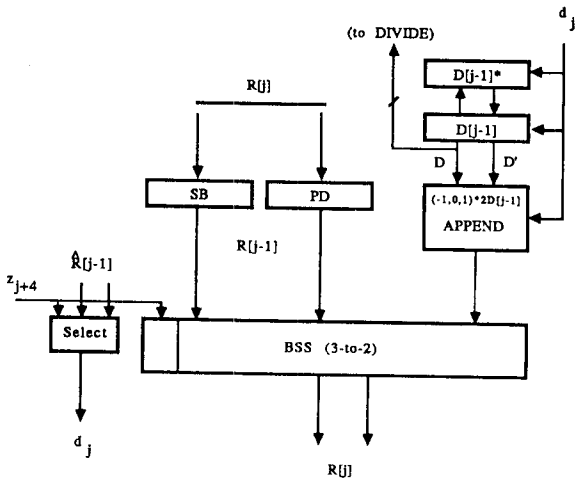

Figure 5. Square Root Scheme

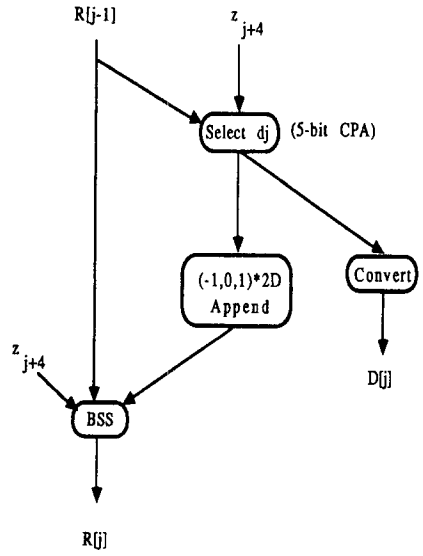

Figure 6. Critical Paths (Square Root)

5. On-line Division Algorithm and Implementation

We now present the algorithm for the division operation. We consider the computation of s (that of c is similar).

To incorporate the necessary on-line delay p we redefine the division operation as

$$s = 2^{-p}\frac{y}{d} \qquad y \in [1/2,1) \qquad d \in [1/2, 2^{1/2}] \tag{5.1}$$

where s is obtained without delay. The real quotient is obtained by shifting s by p positions, that is eliminating the leading 0's. Consequently, the real quotient is obtained with an on-line delay of p.

Since we are developing an algorithm that is specifically suited for the computation of the rotation factors, we assume that y (the dividend) is known in parallel form (not on-line), while d is on-line and s is obtained also on-line (but converted on-the-fly to conventional representation). This formulation leads to the following recurrence. Let

$$P[j] = 2^j (2^{-p} y - S[j] \times D[j]) \tag{5.2}$$

be a partial remainder, satisfying the bound by

$$|P[j]| < |D[j]|$$

The corresponding recurrence is

$$P[j] = 2P[j-1] - S[j-1]d_j - D[j]s_j \tag{5.4}$$

with

$$P[0] = 2^{-p} y \tag{5.5}$$

To determine the actual bounds on $P[j]$ we calculate the interval $[L_k, U_k]$ of $2P[j-1]$ such that $s_j = k$ is a valid choice. From the recurrence we get

$$L_k - 2^{-p} - kD[j] \le P[j](k) \le U_k + 2^{-p} - kD[j]$$

since $|S[j]| \le 2^{-p}$.

To keep the remainder bounded we make

$$c_{-1} = \frac{L_{-1}}{2} \le P[j] \le \frac{U_1}{2} = c_1 \tag{5.6}$$

Consequently,

$$U_1 = 2D[j] - 2^{-p+1} \tag{5.7}$$

$$L_{-1} = -2D[j] + 2^{-p+1} \tag{5.9}$$

and

$$c_1 = -c_{-1} = c = D[j] - 2^{-p} \tag{5.10}$$

The quotient selection function, as shown in Appendix B, is suitably defined using an on-line delay $p_{div} = 3$, the precision of t=2 fractional bits in the estimate of the partial remainder, and the selection constants 1/8 and -3/8.

The corresponding algorithm is given next.

Algorithm *Division*

/* Initialization */
$P[0] \leftarrow y \times 2^{-3}$; $D[0] \leftarrow 0$; $S[0] \leftarrow 0$; $s_0 \leftarrow 0$;

/* Recurrence */
for j=1,2,...,n+3:

{$W[j] = 2P[j-1] - S[j-1]d_j$;

/ * Note that $S[j-1] < 2^{-p} = 2^{-3}$ */

$\hat{W}[j] = (2P[j-1])_4 - (1 - sign(d_j)) * 2^{-3}$;

$$s_j = \begin{cases} -1 & \hat{W}[j] \le -3/8 \\ 0 & -3/8 < \hat{W}[j] < 1/8 \\ +1 & \hat{W}[j] \ge 1/8 \end{cases}$$

$D[j] \leftarrow D[j-1] + d_j 2^{-j}$;
$P[j] \leftarrow W[j] - D[j]s_j$;
$S[j] \leftarrow S[j-1] + s_j 2^{-j}$;}

end *Divide*
$(A)_4$ means the four most significant bits of A.

Implementation

The implementation is shown in Figure 7. Note that the additions/subtractions are done using carry-save adders in 2's complement system; for this reason, the divisor and the quotient are converted on-the-fly [ERC85] into conventional form to be used as operands for the additions.

The critical path is shown in Figure 8. We estimate that its delay corresponds to two multiplexers, one 4-bit CPA and one CSA.

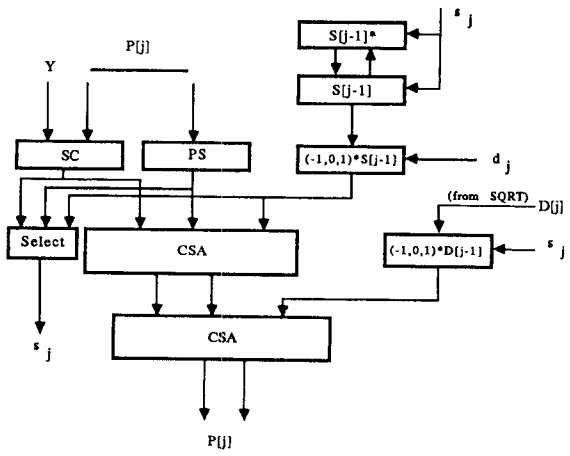

Figure 7. Division Scheme

6. Processing of Exponents

The exponents of the result can be obtained directly, instead of going through the intermediate steps. Since the sequence is sum of squares, square root, and division, we get

$$ec = ex - e \qquad es = ey - e$$

where $e = max(ex, ey)$. Therefore, if $ed = ex - ey$ we get

$$ec = \begin{cases} 0 & \text{if } ed \geq 0 \\ ed & \text{otherwise} \end{cases}$$

$$es = \begin{cases} -ed & \text{if } ed \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

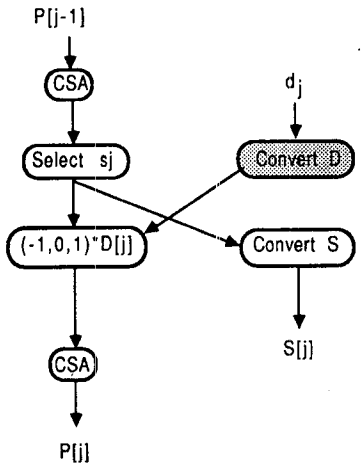

Figure 8. Critical Paths (Division)

7. Summary

An integrated scheme for computing rotation factors using on-line arithmetic algorithms has been presented. The scheme has the following characteristics:

- The inputs and outputs are in a conventional floating-point, bit-parallel format, 2's complement;

- The on-line delay of the scheme, including the on-the-fly conversion delay p_{conv}, is

$$p_{rot} = p_{align} + p_{ss} + p_{sqr} + p_{div} + p_{conv}$$

$$= 1+1+4+3+1 = 10$$

- The latency is $T = p_{rot} + n - 1 = 9 + n$ for n-bit significands;

- The cycle time is determined roughly by one carry-save adder, two multiplexers, and a 5-bit carry-propagate adder.

- The on-line algorithms operate internally on 2's complement representations so that efficient carry-save and borrow-save structures can be used.

Acknowledgements This research has been supported in part by the ONR Contract N00014-85-K-0159 "On-Line Arithmetic Algorithms and Structures for VLSI". We also thank Dr. J. G. Nash of Hughes Research Laboratories, Malibu, for his interest and support.

References

[CIM81] L. Ciminiera, A. Serra, and A. Valenzano, "Fast and Accurate Matrix Triangularization Using an Iterative Structure", *Proc. 5th IEEE Symposium on Computer Arithmetic*, Ann Arbor, 1981, pp. 215-221.

[ERC78] M.D. Ercegovac, "An On-Line Square Root Algorithm", *Proc. 4th IEEE Symposium on Computer Arithmetic*, Santa Monica, 1978, pp.183-189.

[ERC81] M.D. Ercegovac and A. Grnarov, "On the Performance of On-Line Arithmetic", *Proc. 1980 Intl. Conference on Parallel Processing*, 1980, pp.55-62.

[ERC84] M.D. Ercegovac, "On-Line Arithmetic: An Overview", *Proc. SPIE 1984*, Vol.495 - Real Time Signal Processing VII, 1984, pp.86-93.

[ERC85] M.D. Ercegovac and T. Lang, "On-the-Fly Conversion of Redundant into Conventional Representations", UCLA Computer Science Department Technical Report No. CSD 850026, August 1985. (accepted for publication IEEE Transactions on Computers).

[GOL83] G.H. Golub and C. F. Van Loan, *Matrix Computations*, Baltimore: The Johns Hopkins University Press, 1983.

[LUK86] F. T. Luk, "Architectures for Computing Eigenvalues and SVDs", *SPIE Proc. Highly Parallel Signal Processing Architectures*, Vol 614, 1986, pp. 24-33.

[OKL82] V.G. Oklobdzija and M.D. Ercegovac, "An On-Line Square Root Algorithm", *IEEE Transactions on Computers*, Vol. C-31, No.1, January 1982, pp.70-75.

[TRI77] K.S. Trivedi and M.D. Ercegovac, "On-Line Algorithms for Division and Multiplication", *IEEE Transactions on Computers*, Vol. C-26, No.7, July 1977, pp.667-680.

Appendix A

Selection Function for Square Root

We determine here a selection function for d_i so that the bounds on the residual $R[i]$ are satisfied. For this, we compute the intervals $[L_k, U_k]$ of $R[i-1]$ so that the value $d_i=k$ ($k=-1,0,1$) can be selected and $R[i]$ is inside the allowed interval of bounds defined by (4.3). The expression for $R[i-1]$ is

$$R[i-1] = 2^{-1}R[i] - z_{i+p}2^{-p-1} + d_i D[i-1] + d_i^2 2^{-i-1}$$

The intervals are

For $d_i = 1$,

$$\begin{aligned} L_1 &= -D[i] + 2^{-i-1} + b2^{-p-1} - z_{i+p}2^{-p-1} \\ &\quad + D[i-1] + 2^{-i-1} \\ &= b2^{-p-1} - z_{i+p}2^{-p-1} \\ U_1 &= D[i] + 2^{-i-1} - a2^{-p-1} - z_{i+p}2^{-p-1} \\ &\quad + D[i-1] + 2^{-i-1} \\ &= 2D[i-1] + 2^{-i+1} - a2^{-p-1} - z_{i+p}2^{-p-1} \end{aligned}$$

For $d_i = 0$,

$$\begin{aligned} L_0 &= -D[i] + 2^{-i-1} + b2^{-p-1} - z_{i+p}2^{-p-1} \\ &= -D[i-1] + 2^{-i-1} + b2^{-p-1} - z_{i+p}2^{-p-1} \\ U_0 &= D[i-1] + 2^{-i-1} - a2^{-p-1} - z_{i+p}2^{-p-1} \end{aligned}$$

and for $d_i = -1$,

$$\begin{aligned} L_{-1} &= -D[i] + 2^{-i-1} + b2^{-p-1} - z_{i+p}2^{-p-1} \\ &\quad - D[i-1] + 2^{-i-1} \\ &= -2D[i-1] + 2^{-i+1} + b2^{-p-1} - z_{i+p}2^{-p-1} \\ U_{-1} &= D[i] + 2^{-i-1} - a2^{-p-1} - z_{i+p}2^{-p-1} \\ &\quad - D[i-1] + 2^{-i-1} \\ &= -a2^{-p-1} - z_{i+p}2^{-p-1} \end{aligned}$$

Since $(-z_{i+p}2^{-p-1})$ appears in all the expressions, we will base the selection on

$$R^*[i-1] = R[i-1] + z_{i+p}2^{-p-1} \tag{A.1}$$

Consequently,

$$\begin{aligned} L_1^* &= b2^{-p-1} \\ U_1^* &= 2D[i-1] + 2_{-i+1} - a2^{-p-1} \\ L_0^* &= -D[i-1] + 2^{-i-1} + b2^{-p-1} \\ U_0^* &= D[i-1] + 2^{-i-1} - a2^{-p-1} \\ L_{-1}^* &= -2D[i-1] + 2^{-i+1} + b2^{-p-1} \\ U_{-1}^* &= -a2^{-p-1} \end{aligned} \tag{A.2}$$

The containment conditions, $U_1 \le C_1[i-1]$ and $L_{-1} \le C_{-1}[i-1]$, require that $-b \le z_i \le a$ which is satisfied by selecting a and b according to (3.11).

The continuity of the selection intervals and the use of carry-save adders require that the interval overlaps satisfy the following conditions:

$$\begin{aligned} \Delta_{01} &= U_0^* - L_1^* \\ &= D[i-1] + 2^{-i-1} - (a+b)2^{-p-1} \ge 2^{-t} \\ \Delta_{\bar{1}0} &= U_{-1}^* - L_0^* \\ &= D[i-1] - 2^{-i-1} - (a+b)2^{-p-1} \ge 2^{-t} \end{aligned} \tag{A.3}$$

where t is the precision of the assimilated remainder.

Since $z \ge 2^{-2}$ (from the sum of squares), the smallest possible value of $d=2^{-1}$. Therefore, for $a=6$ and $b=2$ (as produced by the sum of squares), we get

$$\begin{aligned} \Delta_{01}\min &= 2^{-1} - 4\cdot 2^{-p} \ge 2^{-t} \\ \Delta_{\bar{1}0}\min &= 2^{-1} - 2^{-i-1} - 4\cdot 2^{-p} \ge 2^{-t} \end{aligned} \tag{A.4}$$

Since $d \ge 1/2$, the first negative digit cannot happen for $i<3$. Therefore, the positive overlaps imply $p \ge 4$ and $t \ge 2$.

We now determine the selection constants k_{01} and $k_{\bar{1}0}$, assuming $p=4$, $a=6$ and $b=2$. The overlap region for the selection of 0 or 1 is bounded by

$$\begin{aligned} L_1^*(max) &= b2^{-p-1} = 2^{-4} \\ U_0^*(min) &= D[i-1] + 2^{-i-1} - a2^{-p-1} \\ &> 2^{-1} - 6\cdot 2^{-5} = 5/16 \end{aligned} \tag{A.5}$$

Since the error in the remainder estimate is always positive (because of the use of carry-save adders and the 2's complement representation), we can choose $k_{01} = 1/16$. For this choice we require $t \ge 2$.

The overlap region for choosing between 0 and -1 is

$$L_0^*(max) = -D[i-1] + 2^{-i-1} + b2^{-p-1} \tag{A.6}$$

$$\leq -2^{-1} + 2^{-3-1} + 2^{-4} = -3/8$$

$$U^{*}_{-1}\ (min) = -a\,2^{-p-1} = -3/16$$

which leads to choice of $k_{\overline{1}0} = -3/8$ with $t \geq 3$. Taking into account both cases, a suitable choice is

$$k_{01} = 1/8 \quad \text{and} \quad k_{\overline{1}0} = -3/8 \quad \text{with} \quad t \geq 3 \tag{A.7}$$

Appendix B

Quotient Selection Function

Since we want to perform the additions/subtractions in carry-save form and the quotient selections using a limited-precision remainder (with precision of 2^{-t}), it is necessary that the selection regions for k and $k-1$ overlap by at least 2^{-t}. We base the selection on

$$W[j] = 2P[j-1] - S[j-1]d_j = P[j] + D[j]s_j \tag{B.1}$$

The selection interval for $s_j = k$ is

$$A_k = -c + kD_j \leq W_j \leq c + kD_j = B_k \tag{B.2}$$

and the overlap is

$$\Delta(k,k-1) = B_{k-1} - A_k \tag{B.3}$$

$$= c + (k-1)D[j] + c - kD[j]$$

$$= 2c - D[j]$$

Replacing the value of c, we get

$$D[j] - 2^{-p+1} \geq 2^{-t} \tag{B.4}$$

so that

$$2^{-p} \leq \frac{D[j] - 2^{-t}}{2} \tag{B.5}$$

For $D[j] \geq 1/2$ we get

$$2^{-p} \leq \frac{1-2^{-t+1}}{4} \tag{B.6}$$

A possible solution to this inequality is

$$p \geq 3 \quad \text{and} \quad t \geq 2 \tag{B.7}$$

We now determine suitable selection constants. For $p = 3$, we have the following intervals:

for $s_j = -1$: $[-2D[j] + 2^{-3}\ ,\ -2^{-3})$

for $s_j = 0$: $[-D[j] + 2^{-3}\ ,\ D[j] - 2^{-3})$

for $s_j = 1$: $[2^{-3}\ ,\ 2D[j] - 2^{-3})$

To have a selection that is independent of the value of $D[j]$ we look at the values of $D[j]$ that produce the smallest intervals. This occurs in all cases for $D[j] = 1/2$. The resulting intervals are:

for $s_j = -1$: $[-7/8\ ,\ -1/8)$

for $s_j = 0$: $[-3/8\ ,\ +3/8)$

for $s_j = 1$: $[+1/8\ ,\ +7/8)$

Since in 2's complement we have

$$\hat{W} = W - \varepsilon \tag{B.8}$$

with $0 \leq \varepsilon \leq 2^{-t}$, we get as selection constants

for $t=2$: $+1/8$ and $-3/8$

for $t=3$: $+1/4$ and $-1/4$.