

RADIX-4 SQUARE ROOT WITHOUT INITIAL PLA

Miloš D. Ercegovac and Tomas Lang

Computer Science Department
School of Engineering and Applied Science
University of California, Los Angeles

Abstract

A systematic derivation of a radix-4 square root algorithm using redundancy in the partial residuals and the result is presented. Unlike other similar schemes the algorithm does not use a table-lookup or PLA for the initial step. The scheme can be integrated with division. It also performs on-the-fly conversion and rounding of the result, thus eliminating a carry-propagate step to obtain the final result. The selection function uses 4 bits of the result and 8 bits of the estimate of the partial residual.

1. Introduction

Several implementations of radix-4 square root have been presented in the literature [VINE65, GOSL87, FAND87, ZURA87]. In all these cases a table-lookup or a special PLA is included for the determination of the first few bits of the result, while another PLA implements the result-digit selection for the remaining radix-4 digits. In this paper we show that this initial PLA is not necessary, resulting in a simpler implementation. As in the other designs, the implementation can be combined with division: the result digit selection function and the recurrence are identical in all steps.

The operand and result are in floating-point representation. To permit the computation of the exponent of the result, the exponent of the operand has to be even. To accomplish this, the mantissa of the operand is multiplied by 1/2 when its exponent is odd. Consequently, the operand mantissa is in the range [1/4, 1). The mantissa of the result is then in the range [1/2, 1).

To obtain a fast implementation, as done in [GOSL87, FAND87, ZURA87], carry-save addition is used and the result-digit selection depends on low-precision estimates of the residual and of the partial result. This requires that the result digit be from a redundant digit-set. As the other referenced implementations we use the set $\{-2, -1, 0, 1, 2\}$ to simplify the multiple formation required by the recurrence.

The signed-digit result is converted on-the-fly to conventional representation. Moreover, during this conversion on-the-fly rounding is performed [ERCE88].

2. Recurrence and Square Root Step

The algorithm is based on a continued-sum recurrence. We now develop the digit-recurrence for the algorithm and show the implementation of the corresponding iteration step.

2.1 Recurrence

Each iteration of the recurrence produces one digit of the result, most-significant digit first. Let $S[j]$ be the value of the result after j iterations, that is

$$S[j] = \sum_{i=1}^j s_i 4^{-i} \quad (1)$$

The final result is then $s = S[n] = \sum_{i=1}^n s_i 4^{-i}$. Define an error function ϵ so that its value after j steps is

$$\epsilon[j] = x^{1/2} - S[j] \quad (2)$$

where $1/4 \leq x < 1$ is the operand.

To have a correct result this error has to be bounded. To use a redundant representation of the result, we allow a positive or negative error such that

$$-4^{-j} < \epsilon[j] < 4^{-j} \quad (3)$$

If a positive final remainder is required, a restoration step is included. We now transform (2) to eliminate the square root operation:

$$4^{-2j} - 2 \times 4^{-j} S[j] + S[j]^2 < x < 4^{-2j} + 2 \times 4^{-j} S[j] + S[j]^2$$

Subtracting $S[j]^2$ produces

$$4^{-2j} - 2 \times 4^{-j} S[j] < x - S[j]^2 < 4^{-2j} + 2 \times 4^{-j} S[j] \quad (4)$$

Define a residual (or partial remainder) w so that

$$w[j] = 4^j (x - S[j]^2) \quad (5)$$

From (4) the bound on the residual is

$$-2S[j] + 4^{-j} < w[j] < 2S[j] + 4^{-j} \quad (6)$$

and its initial condition

$$w[0] = x \quad (7)$$

In terms of this residual, the basic recurrence used in the square root algorithm is

$$w[j+1] = 4w[j] - 2S[j]s_{j+1} - s_{j+1}^2 \times 4^{-(j+1)} \quad (8)$$

2.2 Implementation of Square Root Step

The square root algorithm consists in performing m iterations of the recurrence (8). Moreover, each iteration consists of four subcomputations (Figure 1a):

- 1) One digit arithmetic left-shift of $w[j]$ to produce $4w[j]$.
- 2) Determination of the result digit s_{j+1} using a *result-digit selection function*. The value of the digit s_{j+1} is selected so that the application of the recurrence produces a $w[j+1]$ that satisfies the bound (6). The function has as arguments $\hat{w}[j]$ (an estimate of $4w[j]$) and $\hat{S}[j]$ (an estimate of $S[j]$) and produces s_{j+1} . That is,

$$s_{j+1} = f(\hat{w}[j], \hat{S}[j]) \quad (9)$$

- 3) Formation of $F = -2S[j]s_{j+1} - s_{j+1}^2 \times 4^{-(j+1)}$ and $S[j+1] = S[j] + s_{j+1} \times 4^{-(j+1)}$.
- 4) Subtraction of F from $4w[j]$ to produce $w[j+1]$.

The four subcomputations are executed in sequence (Figure 1b). No time is allocated for the arithmetic shift since it is performed just by suitable wiring. Moreover, the relative magnitudes of the delay of each of the components depend on the specific implementation.

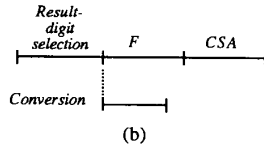
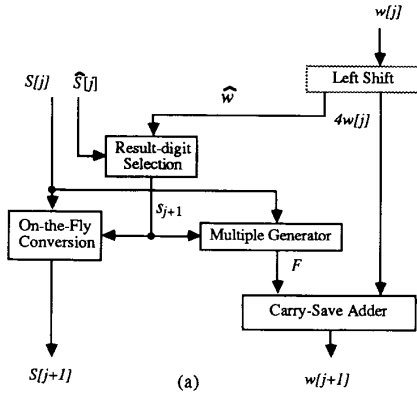


Figure 1. (a) Square root step, (b) Timing

To have a fast recurrence step we use a carry-save adder and a result-digit selection function that depends on low-precision estimates of the residual and of the partial result. To achieve this, it is necessary to have a redundant representation of the result-digit. In particular, we use the symmetric signed-digit set

$$s_j \in \{-2, -1, 0, 1, 2\} \quad (10)$$

because it allows a simpler implementation of the adder input F . Moreover, the signed result-digit makes it necessary to use an on-the-fly conversion to produce $S[j]$ in a conventional form for the formation of F .

2.3 Implementation of Square Root Algorithm

As indicated, the square root algorithm consists of m iterations of the recurrence. The implementation of this algorithm can be *totally sequential*, where the hardware of the step is reused for all the iterations and the residual is updated in a register; *totally combinational*, where the hardware for the step is replicated; or a combination of both, where the step hardware is replicated k times and this superstep is reused m/k times. Specially in the combinational implementations, pipelining can be used so that several operations can use the hardware at the same time, with the corresponding increase in throughput. The selection of one of these alternatives is influenced by cost, speed, and throughput considerations.

3. Result-digit Selection Function

We now present the design of the result-digit selection function. This function determines the value of the result digit s_{j+1} as a function of the residual $w[j]$ and the partial result $S[j]$.

There are two fundamental conditions that must be satisfied by a selection function: *containment* and *continuity*. These conditions determine a *selection interval* for each value of s_{j+1} , from which alternative result-digit selections can be defined.

3.1 Containment Condition and Selection Intervals

One basic requirement for the result-digit selection is that the selection produces a next residual that is bounded. This leads to the containment condition, which we now develop. Let the bounds of the residual $w[j]$ be called \underline{B} and \bar{B} , that is,

$$\underline{B}[j] \leq w[j] \leq \bar{B}[j] \quad (11)$$

Define the *selection interval* of $4w[j]$ for $s_j = k$ to be $[L_k, U_k]$. That is, L_k (U_k) is the smallest (largest) value of $4w[j]$ for which it is possible to choose $s_{j+1} = k$ and keep $w[j+1]$ bounded. Therefore,

$$L_k[j] \leq 4w[j] \leq U_k[j] \quad (12)$$

implies that

$$\underline{B}[j+1] \leq 4w[j] - 2S[j]k - k^2 \times 4^{-(j+1)} \leq \bar{B}[j+1]$$

Consequently,

$$U_k[j] = \bar{B}[j+1] + 2S[j]k + k^2 \times 4^{-(j+1)} \quad (13)$$

$$L_k[j] = \underline{B}[j+1] + 2S[j]k + k^2 \times 4^{-(j+1)}$$

We now can determine $\bar{B}[j]$ and $B[j]$ because they are the upper bound of the interval for $k=2$ and the lower bound for $k=-2$, respectively. That is,

$$U_2[j] = 4\bar{B}[j] \quad L_{-2}[j] = 4B[j]$$

Introducing these values in (13) we get

$$\bar{B}[j+1] + 2S[j] \times 2 + 2^2 4^{-(j+1)} = 4\bar{B}[j] \quad (14)$$

$$B[j+1] - 2S[j] \times 2 + 2^2 4^{-(j+1)} = 4B[j]$$

This results in

$$\bar{B}[j] = \frac{4}{3}S[j] + \frac{4}{9}4^{-j} \quad B[j] = -\frac{4}{3}S[j] + \frac{4}{9}4^{-j} \quad (15)$$

To show that (15) is correct, it is sufficient to replace in (14). Note that, in contrast to division, the bounds vary with j . These bounds satisfy the bound on the residual of (6).

The *containment condition* is obtained from (13) and (15). It states that the selection interval for $s_{j+1} = k$ is given by the expressions

$$U_k[j] = \frac{4}{3}S[j+1] + \frac{4}{9}4^{-(j+1)} + 2S[j]k + k^2 4^{-(j+1)}$$

Since $S[j+1] = S[j] + k \times 4^{-(j+1)}$ we get

$$U_k[j] = 2S[j](k + \frac{2}{3}) + (k + \frac{2}{3})^2 4^{-(j+1)} \quad (16a)$$

Similarly,

$$L_k[j] = 2S[j](k - \frac{2}{3}) + (k - \frac{2}{3})^2 4^{-(j+1)} \quad (16b)$$

A diagram that contains information useful in the design of the result-digit selection function is the residual vs. partial result plot, called the *R-PR plot* (Figure 2). It is similar to the *P-D plot*, used in division: it has as axes the partial result $S[j]$ and the shifted residual $4w[j]$. The bounds of the selection intervals U_k and L_k are plotted as lines.

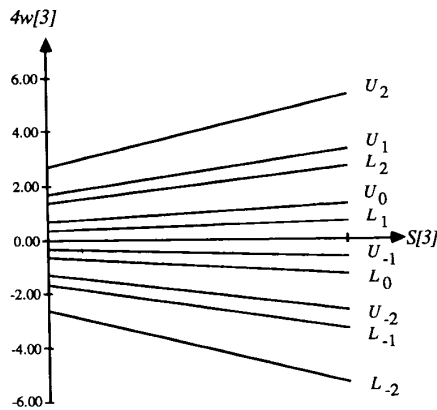


Figure 2. P-PR Diagram ($j=3$)

3.2 Continuity Condition and Overlap

A second requirement for the selection intervals is the *continuity condition*: for any value of $4w[j]$ between $4\bar{B}[j]$ and $4B[j]$ it must be possible to select some value for the result digit. This can be expressed as

$$U_{k-1} \geq L_k - 4^{-m} \quad (17)$$

Moreover, to use estimates of $4w[j]$ and of $S[j]$ for the result-digit selection, it is necessary to have an overlap between the adjacent selection intervals. For the square-root operation with digit-set $\{-2, -1, 0, 1, 2\}$ the overlap is

$$U_{k-1} - L_k = \frac{1}{3}(2S[j] + (2k-1)4^{-(j+1)}) \quad (18)$$

Note that the overlap depends on $S[j]$, on k , and on j . We will analyze the different cases later and show that there is sufficient overlap to use estimates for the result-digit selection.

3.3 Result-digit selection for residual in carry-save form

We now determine the result-digit selection function using an estimate of the shifted residual obtained by truncating the carry-save form.

The truncation of the shifted residual in carry-save form to t fractional bits produces an estimate \hat{w} with error satisfying

$$0 \leq 4w[j] - \hat{w} < 2 \times 2^{-t} \quad (19)$$

Consequently, to have a correct result the basic requirement is that if we choose $s_{j+1} = k$ for an estimate \hat{w} , then this selection has to be acceptable for the interval

$$4w[j] \in [\hat{w}, \hat{w} + 2^{-(t-1)}] \quad (20)$$

The result-digit selection function we develop is of the "staircase" type as illustrated in Figure 3a. Such a function is defined by selection constants $m_i(k)$ which are used for partial result interval $S[j] \in [S_i, S_{i+1})$, where $S_i = (2^{-1} + i \times 2^{-6})$. That is, for that interval we choose

$$s_{j+1} = k \quad \text{if} \quad m_i(k) \leq \hat{w} < m_{i+1}(k)$$

The set $\{m_i(k) \mid 0 \leq i \leq (2^6 - 1) \text{ and } -2 \leq k \leq 2\}$ defines the result-digit selection function.

If the selection constants have a precision of t fractional bits, that is, $m_i(k) = A_i(k)2^{-t}$, where $A_i(k)$ is an integer, we get from the containment and continuity conditions, and (20)

$$A_i(k)2^{-t} \geq \max(L_k(S_i), L_k(S_{i+1})) \quad (21a)$$

$$(A_i(k) - 1)2^{-t} + 2^{-(t-1)} \leq \min(U_{k-1}(S_i), U_{k-1}(S_{i+1})) \quad (21b)$$

The second expression has to hold because for $(A_i(k) - 1)$ we want to choose $s_{j+1} = k - 1$. These expressions are illustrated in Figure 3b.

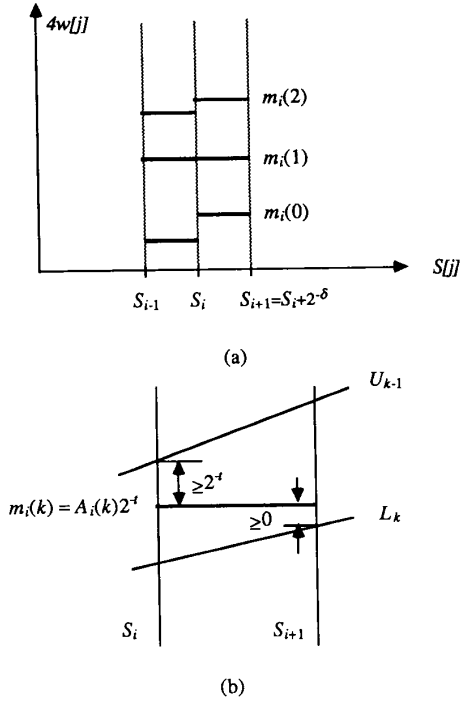


Figure 3. (a) Staircase selection - a fragment, (b) Conditions on selection constants

Consequently, the main relation used for obtaining the corresponding (staircase) result-digit selection is

$$A_i(k)2^{-t} \geq \max(L_k(S_i), L_k(S_{i+1})) \quad (22)$$

$$A_i(k)2^{-t} \leq \min(\hat{U}_{k-1}(S_i), \hat{U}_{k-1}(S_{i+1}))$$

where $\hat{U} = U - 2^{-t}$.

The values of δ and t are obtained by trial and error. To reduce the number of trials, lower bounds are obtained from the need for a sufficient overlap, as described in [ERCE89].

For the radix-4 case with digit-set $\{-2, \dots, 2\}$ we have from (16) the following interval bounds:

k	$U_k[j]$	$L_k[j]$
2	$2S[j] \times (8/3) + (8/3)^2 4^{-j+1}$	$2S[j] \times (4/3) + (4/3)^2 4^{-j+1}$
1	$2S[j] \times (5/3) + (5/3)^2 4^{-j+1}$	$2S[j] \times (1/3) + (1/3)^2 4^{-j+1}$
0	$2S[j] \times (2/3) + (2/3)^2 4^{-j+1}$	$-2S[j] \times (2/3) + (2/3)^2 4^{-j+1}$
-1	$-2S[j] \times (1/3) + (1/3)^2 4^{-j+1}$	$-2S[j] \times (5/3) + (5/3)^2 4^{-j+1}$
-2	$-2S[j] \times (4/3) + (4/3)^2 4^{-j+1}$	$-2S[j] \times (8/3) + (8/3)^2 4^{-j+1}$

Since these limits depend on j , the result-digit selection might vary with the value of j . We now consider the different cases.

Since the maximum digit value is 2, it is necessary to have $s_0 = 1$ to be able to represent values of $s \geq 2/3$. Consequently, $S[0] = 1$, which leads to $s_1 = \{0, -1, -2\}$ (to have $s < 1$).

Therefore, for $j = 0$ we obtain,

$$U_{-1}[0] = -2 \times 1 \times (1/3) + (1/9)(1/4) = -(23/36)$$

$$L_0[0] = -2 \times 1 \times (2/3) + (4/9)(1/4) = -(44/36)$$

This results in a possible $m(0) = -1$. Similarly,

$$U_{-2}[0] = -2 \times 1 \times (4/3) + (16/9)(1/4) = -(80/36)$$

$$L_{-1}[0] = -2 \times 1 \times (5/3) + (25/9)(1/4) = -(95/36)$$

which results in a possible $m(-1) = -5/2$. Since $-5/2 = -90/36$ we get a minimum value of $2^{-t} = (90-80)/36 > 1/4$.

For $j = 1$ we can have $s_2 = \{-2, -1, 0, 1, 2\}$. Moreover, since $s_1 = \{-2, -1, 0\}$ and $S[0] = 1$ the possible values of $S[1]$ are $1/2, 3/4, 1$. A lower bound of $t=3$ is obtained [ERCE89a]. For this value, Table 1 shows the corresponding values of L_k and \hat{U}_k and a result-digit selection that satisfies (22).

Table 1. Result-digit selection for $j=1$

	$S[1] = 1/2$	$S[1] = 3/4$	$S[1] = 1$
L_2, \hat{U}_1	208/144, 247/144	304/144, 367/144	400/144, 487/144
$m(2)$	3/2	9/4	3
L_1, \hat{U}_0	49/144, 82/144	73/144, 130/144	97/144, 178/144
$m(1)$	1/2	3/4	1
L_0, \hat{U}_{-1}	-	-140/144, -89/144	-188/144, -113/144
$m(0)$	-	-3/4	-1
L_{-1}, \hat{U}_{-2}	-	-335/144, -290/144	-455/144, -386/144
$m(-1)$	-	-9/4	-3

Note that it is not possible to select $s_2 < 0$ when $S[1] = 1/2$, because this would make $S[2] < 1/2$.

For $j = 2$ we obtain lower bounds of $\delta = 4$ and $t = 3$ (see [ERCE89]). We now develop the result-digit selection using these values. Since we use $\delta = 4$ and the granularity of $S[2]$ is $1/16$, we use exact values instead of intervals. Table 2 shows the corresponding values of L_k and \hat{U}_k and a possible result-digit selection. The term \hat{U} is $\hat{U}_k = U_k - 1/8$.

For $j \geq 3$ we use $\delta = 4$ and $t = 4$ [ERCE89]. Moreover, to make the function independent of j (only dependent on $S[j]$) we use the following bounds:

$$L_k[j] < 2S[j](k-2/3) + (k-2/3)^2 4^{-j} = L_k^* + \frac{(k-2/3)^2}{256}$$

$$U_k[j] > 2S[j](k+2/3)$$

The term $(k-2/3)^2/256$ has the values:

k	2	1	0	-1	-2
$\frac{(k-2/3)^2}{256}$	1/144	1/2304	1/576	1/90	1/36

Table 2. Result-digit selection for $j = 2$

S_i	8/16	9/16	10/16
$L_2, \hat{U}_1 (<576)$ $m(2)$	784, 913 3/2	880, 1033 7/4	976, 1143 7/4
$L_1, \hat{U}_0 (<576)$ $m(1)$	193, 316 1/2	217, 364 1/2	241, 412 1/2
$L_0, \hat{U}_{-1} (<576)$ $m(0)$	-380, -263 -	-428, -287 -1/2	-476, -311 -3/4
$L_{-1}, \hat{U}_{-2} (<576)$ $m(-1)$	-935, -824 -	-1045, -920 -7/4	-1155, -1016 -2
S_i	11/16	12/16	13/16
$L_2, \hat{U}_1 (<576)$ $m(2)$	1072, 1253 2	1168, 1363 9/4	1264, 1473 5/2
$L_1, \hat{U}_0 (<576)$ $m(1)$	265, 460 1/2	289, 508 3/4	333, 556 3/4
$L_0, \hat{U}_{-1} (<576)$ $m(0)$	-524, -335 -3/4	-572, -359 -3/4	-620, -383 -1
$L_{-1}, \hat{U}_{-2} (<576)$ $m(-1)$	-1265, -1112 -2	-1375, -1208 -9/4	-1485, -1304 -5/2
S_i	14/16	15/16	16/16
$L_2, \hat{U}_1 (<576)$ $m(2)$	1360, 1583 5/2	1456, 1693 11/4	1552, 1803 3
$L_1, \hat{U}_0 (<576)$ $m(1)$	357, 604 1	381, 652 1	405, 700 1
$L_0, \hat{U}_{-1} (<576)$ $m(0)$	-668, -407 -1	-716, -431 -1	-754, -455 -1
$L_{-1}, \hat{U}_{-2} (<576)$ $m(-1)$	-1595, -1400 -5/2	-1705, -1496 -11/4	-1815, -1592 -3

Since this term is relatively small with respect to $2^{-i} = 2^{-4}$, we use L^*_k instead of L_k , with the limitation that $m_i(k)$ cannot be equal to L^*_k . Table 3 gives the corresponding intervals and a possible selection function.

Since we want a single result-digit selection (independent of j), we now need to match the result-digit selections for $j=0$, $j=1$, $j=2$, and $j \geq 3$. We take as a basis the selection for $j \geq 3$ and compare the corresponding entries with those for $j=2$, $j=1$, and $j=0$. When the entries are different we adjust them to satisfy all cases. We get Table 4. The only case we cannot match is the value $m(-1) = -5/2$ for $j=0$ (for $S[0] = 1$), since for the other values of j it is -3 . A simple solution is to apply $S[0] = 13/16$ instead of 1 for the result-digit selection. The implementation is shown in Figure 4. The converted $S[j]$ is called $A[j]$, as described in the next Section. Since $\delta = 2^{-4}$, four fractional bits of $A[j]$ are needed to define \hat{S} . Moreover, $A[j]$ can have value 1, so that an additional integer bit is required. However, the result-selection function is the same for $\hat{S} = 1.0000$ and $\hat{S} = 0.1111$, so that when $A[j] = 1.0000$ we produce $\hat{S} = 0.1111$; in such a case, the bit \hat{S}_1 is always 1 and is not needed. Finally, we need a constant value of $13/16 = .1101$ for the first step. Therefore,

$$(\hat{S}_2 \hat{S}_3 \hat{S}_4) = (101)(j=0) + (111)A_0(j \neq 0) + (A_2 A_3 A_4)A_0$$

Table 3. Result-digit Selection for $j \geq 3$

$[S_i, S_{i+1})$	8/16 9/16	9/16 10/16	10/16 11/16	11/16 12/16
$L^*_2(S_{i+1}), \hat{U}_1(S_i)$ $m_i(2)$	3/2, 77/48 25/16	5/3, 29/16 7/4	11/6, 97/48 15/8	2, 107/48 17/8
$L^*_1(S_{i+1}), \hat{U}_0(S_i)$ $m_i(1)$	3/8, 29/48 1/2	5/12, 11/16 1/2	11/24, 37/48 1/2	1/2, 41/48 3/4
$L^*_0(S_i), \hat{U}_{-1}(S_{i+1})$ $m_i(0)$	-2/3, -7/16 -1/2	-3/4, -23/48 -5/8	-5/6, -25/48 -3/4	-11/12, -9/16 -3/4
$L^*_{-1}(S_i), \hat{U}_{-2}(S_{i+1})$ $m_i(-1)$	-5/3, -25/16 -13/8	-15/8, -83/48 -29/16	-25/12, -91/48 -2	-55/24, -33/16 -9/4
$[S_i, S_{i+1})$	12/16 13/16	13/16 14/16	14/16 15/16	15/16 16/16+
$L^*_2(S_{i+1}), \hat{U}_1(S_i)$ $m_i(2)$	13/6, 39/16 9/4	7/3, 127/48 5/2	15/6, 137/48 11/4	8/3, 49/16 3
$L^*_1(S_{i+1}), \hat{U}_0(S_i)$ $m_i(1)$	13/24, 15/16 3/4	7/12, 49/48 3/4	5/8, 53/48 1	2/3, 19/16 1
$L^*_0(S_i), \hat{U}_{-1}(S_{i+1})$ $m_i(0)$	-1, -29/48 -3/4	-13/12, -31/48 -1	-7/6, -11/16 -1	-5/4, -35/48 -1
$L^*_{-1}(S_i), \hat{U}_{-2}(S_{i+1})$ $m_i(-1)$	-15/6, -107/48 -9/4	-65/24, -115/48 -5/2	-35/12, -41/16 -11/4	-75/24, -133/48 -3

+ includes 16/16

Table 4. Single Result-digit Selection (for all values of j)

$[S_i, S_{i+1})$	8/16 9/16	9/16 10/16	10/16 11/16	11/16 12/16
$m_i(2)$	25/16	7/4	15/8	17/8
$m_i(1)$	1/2	1/2	1/2	3/4
$m_i(0)$	-1/2	-5/8	-3/4	-3/4
$m_i(-1)$	-13/8	-29/16	-2	-17/8
$[S_i, S_{i+1})$	12/16 13/16	13/16 14/16	14/16 15/16	15/16 16/16+
$m_i(2)$	9/4	5/2	21/8	23/8
$m_i(1)$	3/4	3/4	1	1
$m_i(0)$	-3/4	-1	-1	-1
$m_i(-1)$	-9/4	-5/2	-11/4	-23/8

+ includes 16/16

This is implemented by the three AND and three OR gates of Figure 4. Note that this replaces the much larger initial PLA of other radix-4 square root schemes [FAND87].

The result-digit selection uses three bits of \hat{S} and eight bits of w (four fractional bits and four integer bits). This is the same as for the other proposed implementations.

4. Generation of adder input F

The third input to carry-save adder has value

$$F[j] = -2S[j]s_{j+1} - s_{j+1}^2 4^{-j+1}$$

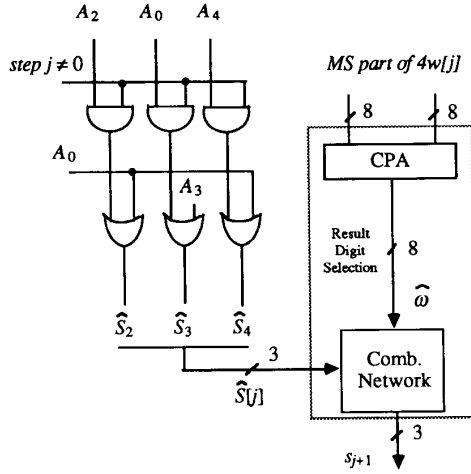


Figure 4. Result-digit selection implementation

To obtain $F[j]$ it is necessary to convert $S[j]$ to conventional radix-2 representation (since s_j is signed-digit). This conversion is done on-the-fly using a variation of the scheme presented in [ERCE87]. It requires that two conditional forms $A[j]$ and $B[j]$ are kept, such that

$$A[j] = S[j], \quad B[j] = S[j] - 4^{-j}$$

These forms are updated with each result-digit as follows:

$$A[j+1] = \begin{cases} A[j] + s_{j+1}4^{-(j+1)} & \text{if } s_{j+1} \geq 0 \\ B[j] + (4 - s_{j+1})4^{-(j+1)} & \text{otherwise} \end{cases}$$

$$B[j+1] = \begin{cases} A[j] + (s_{j+1} - 1)4^{-(j+1)} & \text{if } s_{j+1} > 0 \\ B[j] + (3 - s_{j+1})4^{-(j+1)} & \text{otherwise} \end{cases}$$

The implementation of this conversion requires two registers for A and B , appending of one digit, and loading. For controlling this appending and loading, a shift register K is used, containing a moving 1 (Figure 5).

In terms of these forms, the value of F and the corresponding bit-strings are

	$F[j]$	
s_{j+1}	Value	Bit-string
1	$-2A[j] - 4^{-(j+1)}$	$\bar{a} \cdots \bar{a} \bar{a} 111$
2	$-4A[j] - 4 \times 4^{-(j+1)}$	$\bar{a} \cdots \bar{a} 1100$
-1	$2B[j] + 7 \times 4^{-(j+1)}$	$b \cdots b b 111$
-2	$4B[j] + 12 \times 4^{-(j+1)}$	$b \cdots b 1100$

where $a \cdots aa$ and $b \cdots bb$ are the bit-strings representing $A[j]$ and $B[j]$, respectively.

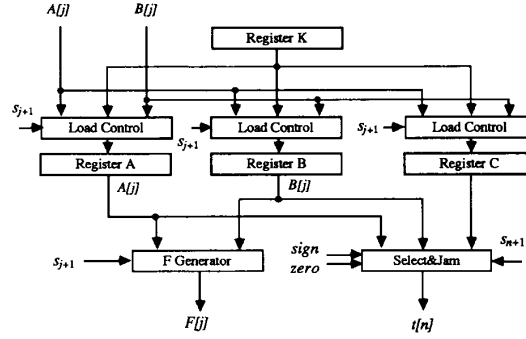


Figure 5. Network for generating F and rounding

5. On-the-fly conversion and rounding

The result digit obtained from the result-digit selection logic is in signed-digit form. As mentioned in the previous section, the partial result is converted on-the-fly to conventional form to use it in the formation of F . Consequently, the final result in conventional form is obtained from register A .

In addition, rounding of the result might be required. The most used type of rounding, rounding-to-nearest, is usually done as follows [FAND87]. First, $n+1$ digits of the result are computed for an n -digit rounded result. Then, a restoration step is performed to obtain a positive residual; to achieve this, the sign of the last residual is determined and the result decremented by one in the least significant position if the sign is negative. Since the representation of the partial residual is redundant (carry-save), the sign has to be obtained from this redundant representation; the process is similar in delay, but simpler in amount of hardware, to a carry-propagate addition that converts the residual to conventional representation. The sign of the residual is then used to decrement the $n+1$ -digit result; this can be done by a subtraction or by using the decremented form available from the on-the-fly conversion. Finally, the (unrounded) result is rounded by, possibly, incrementing it by 1. This incrementation requires a carry-propagate addition. The process is costly both in hardware and in time.

To simplify the hardware required for rounding and increase its speed, in [ERCE89b] we describe three on-the-fly rounding methods that are combined with the conversion. They are as follows:

1) *Rounding to nearest.* In this case the first steps of computing an additional digit and finding the sign of the residual are also required. However, neither the restoration step nor the actual rounding require a carry-propagate addition because they can be performed on-the-fly if a third form is computed during the conversion. The method can be summarized as follows. The rounded result with n digits called $t[n]$ is

$$t[n] = \begin{cases} C[n] & \text{if } (s_{n+1} - \text{sign}) = 2 \\ A[n] & \text{if } -1 \leq (s_{n+1} - \text{sign}) \leq 1 \\ B[n] & \text{if } (s_{n+1} - \text{sign}) \leq -2 \end{cases}$$

where $A[n]$ is the converted result with n digits, $B[n] = A[n] - 2^{-n}$, $C[n] = A[n] + 2^{-n}$, and s_{n+1} is the $(n+1)$ -th signed-digit of the result. Moreover, to have unbiased rounding to nearest it is necessary to set to zero the least significant bit of the result when $|s_{n+1}| = 2$ and the last residual (remainder) is zero.

The forms A and B are produced for the conversion as discussed in the previous section. To be able to do the rounding we need to produce also the form C . It is updated as follows:

$$C[j+1] = \begin{cases} A[j] + (s_{j+1} + 1)4^{-(j+1)} & \text{if } s_{j+1} \geq -1 \\ B[j] + 3 \times 4^{-(j+1)} & \text{if } s_{j+1} = -2 \end{cases}$$

This updating is also done by appending and loading, as shown in Figure 5.

In addition, it is necessary to have a network to detect the sign and zero of the remainder from its redundant representation. Such a network is discussed in [ERCE89b].

2) *Rounding without sign detection.* As a second method for rounding we consider the case in which the sign of the remainder is not detected. This results in a simpler and faster implementation, but with a somewhat larger error. As described in [ERCE88b], the rounding rule to produce the minimum error possible (for the digit set $\{-2, \dots, 2\}$) is $r[n] = A[n]$.

Consequently, in this case $(n+1)$ -th bit of the result does not have to be computed, neither is there need for sign detection, detection of zero, nor computation of C . The rounding is unbiased but with an error bounded by $\pm(2/3)4^{-n}$, which is larger than the error of rounding-to-nearest $((1/2)4^{-n})$.

3) *Rounding with estimate of sign of remainder.* As a compromise between the previous two methods it is possible to use an estimate of the sign of the remainder and then use the rounding rules of method 1 above. The estimate is computed using a few most-significant bits of the redundant remainder as described in [ERCE89b].

6. Overall implementation and timing

The overall implementation at the block-diagram level is shown in Figure 6. The cycle time is

$$\begin{aligned} T_{\text{cycle}} = & t_{\text{digit_select}} \quad \{8\text{-bitCPA} + 12\text{-input network}\} \\ & + t_{F\text{-generate}} \quad \{4\text{-to-1 multiplexer}\} \\ & + t_{\text{CSA}} \quad \{3\text{-to-2 carry-save adder}\} \\ & + t_{\text{load}} \quad \{\text{register loading}\} \end{aligned}$$

This is comparable to the cycle time of a radix-4 division with carry-save adder. Note the absence of a PLA for the initial step.

Acknowledgments This research has been supported in part by the NSF Grant No. MIP-8813340 *Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architectures: Algorithms, Design, and Experimental Studies*.

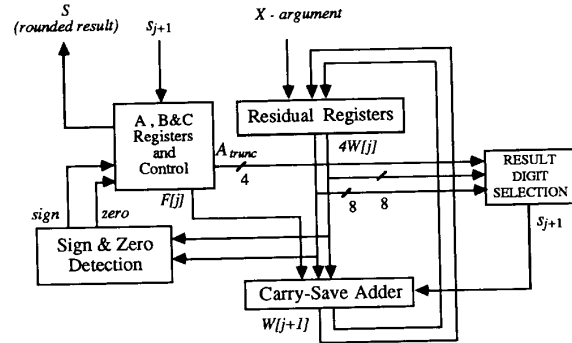


Figure 6. Block diagram of the square root scheme (mantissa part)

References

- [CORT88] J. Cortadella and J.M. Llaberia, "Evaluating $A+B=K$ conditions in constant time", Proc. of the International Conference on Circuits and Systems, Helsinki, 1988.
- [ERCE87a] M.D. Ercegovac and T. Lang, "On-the-Fly Conversion of Redundant into Conventional Representations", IEEE Transactions on Computers, Vol. C-36, No.7, July 1987, pp.895-897.
- [ERCE89a] M.D. Ercegovac and T. Lang, *Square Root Algorithms and Implementations*, monograph in preparation, 1989.
- [ERCE89b] M.D. Ercegovac and T. Lang, "On-the-fly Rounding for Division and Square Root", this proceedings.
- [FAND87] J. Fandrianto, "Algorithm for High Speed Shared Radix-4 Division and Radix-4 Square Root," Proc. 8th Symposium on Computer Arithmetic, 1987, pp. 73-79.
- [GOSL87] J.B. Gosling and C.M.S. Blakeley, "Arithmetic unit with integral division and square-root", IEE Proceedings, Vol. 134, pt. E, no.1, January 1987, pp. 17-23.
- [TAYL85] G.S. Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages", IEEE Proc. of 7th Symposium on Computer Arithmetic, 1985, pp. 64-73.
- [VINE65] M. B. Vineberg, "A Radix-4 Square-Rooting Algorithm", Report No. 182, Department of Computer Science, University of Illinois, Urbana-Champaign, June 1965.
- [WILL87] T.E. Williams et al., "A Self-Timed Chip for Division", Proc. Stanford VLSI Conference, (Ed. Losleben), MIT Press, 1987, pp.75-95.
- [ZURA87] J.H. Zurawski and J.B. Gosling, "Design of a High-Speed Square Root, Multiply, and Divide Unit," IEEE Transactions on Computers, vol. C-36, January 1987, pp. 13-23.