Radix-4 Square Root Without Initial PLA

MILOŠ D. ERCEGOVAC, MEMBER, IEEE, AND TOMAS LANG

Abstract—**A systematic derivation of a radix-4 square-root algorithm using redundant residual and result is presented. Unlike other similar schemes it does not use a table lookup or PLA for the initial step, resulting in a simpler implementation without any time penalty. The scheme can be integrated with division and also incorporates an on-the-fly conversion and rounding of the result, thus eliminating a carry-propagate step to obtain the final result. The result-digit selection uses 3 bits of the result and 7 bits of the estimate of the residual.**

Index Terms—**Digital arithmetic, digit-recurrence, on-the-fly conversion, radix-4 square root, redundant representation.**

I. Introduction

SEVERAL implementations of radix-4 square root have been presented in the literature [9], [6], [5], [10], [8]. In all these cases, a table lookup or a special PLA is included for the determination of the first few bits of the result, whereas another PLA implements the result-digit selection for the remaining radix-4 digits. In this paper, we present a systematic derivation of the algorithm and show that this initial PLA is not necessary; this results in a simpler implementation without any penalty in execution time. As in the other designs, the implementation can be combined with division: the result-digit selection and the recurrence form are identical in all steps.

To obtain a fast implementation, as done in [6], [5], [10], and [8], redundant 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.

Two alternative redundant adders are possible: carry-save adder or signed-digit adder, each having advantages and drawbacks. The advantage of using a carry-save adder is that the adder slice is just a full adder, whereas the signed-digit adder is more complex and slower [7]. On the other hand, since the result is produced in a signed-digit representation and the partial result is one input to the adder, if carry-save addition is used this partial result has to be converted to conventional representation, whereas it can be used directly for signed-digit addition. We choose the carry-save alternative because the result has to be converted anyhow to conventional representation and to do this we use an on-the-fly converter, which after each iteration produces the partial result in conventional form. Moreover, during this conversion on-the-fly rounding is performed [3].

Manuscript received October 26, 1989; revised March 6, 1990. This work was supported in part by NSF Grant MIP-8813340, Composite Operations Using On-Line Arithmetic for Application-Specific Parallel Architecture: Algorithms, Design, and Experimental Studies.

The authors are with the Department of Computer Science, School of Engineering and Applied Science, University of California, Los Angeles, CA 90024.

IEEE Log Number 9036137.

We describe a sequential implementation, in which the hardware of one iteration step is reused for each of the digits of the result. Of course, it is possible to have a combinational implementation, in which the iteration step is replicated. The selection between these alternatives is influenced by cost, speed, and throughput considerations.

The operation is defined by

$$s = x^{1/2} - \epsilon$$

where x is the operand, s is the result, and ϵ is the error. The algorithm is presented for *fractional* operand x and result s. For floating-point representation and normalized operand, it is necessary to scale the operand to have an even exponent (to allow the computation of the exponent). Consequently,

$$1/4 \leq x < 1 \quad 1/2 \leq s < 1.$$

If s has m fractional radix-4 digits then for a correct result the error is bounded so that

$$|\epsilon| < 4^{-m}.$$

In Section II, we describe the algorithm and in Sections III and IV we discuss its implementation.

II. Recurrence and Square Root Step

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

A. Recurrence and Bound

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

$$S[j] = \sum_{i=0}^{j} s_i 4^{-i} \qquad s_i \in \{-2, -1, 0, 1, 2\} \tag{1}$$

(the digit s_0 is needed to represent a result value $s \geq 2/3$, since the representation of s is in signed-digit form and the maximum value of s_i is 2).

The final result is then

$$s = S[m] = \sum_{i=0}^{m} s_i r^{-i}$$

and the result has to be correct for m-digit precision, that is,

$$|x^{1/2} - s| < 4^{-m}. \tag{2}$$

We define an error function ϵ so that its value after j steps is

$$\epsilon[j] = x^{1/2} - S[j]. \tag{3}$$

To have a correct final result this error has to be bounded. Since

$$s = S[j] + \sum_{i=j+1}^{m} s_i 4^{-i}$$

we get from (2) and (3)

$$\min\left(\sum_{i=j+1}^{m} s_i 4^{-i}\right) - 4^{-m} < \epsilon[j] < \max\left(\sum_{i=j+1}^{m} s_i 4^{-i}\right) + 4^{-m}.$$

Since the minimum (maximum) digit value is -2 (2), we get

$$-\frac{2}{3}4^{-j} \le \epsilon[j] \le \frac{2}{3}4^{-j}. \tag{4}$$

Introducing (3) in (4) and transforming to eliminate the square root operation (add $S[j]$ and obtain the square), we get

$$\frac{4}{9}4^{-2j} - \frac{4}{3}4^{-j}S[j] + S[j]^2 \le x \le \frac{4}{9}4^{-2j} + \frac{4}{3}4^{-j}S[j] + S[j]^2.$$

Subtracting $S[j]^2$ we obtain

$$\frac{4}{9}4^{-2j} - \frac{4}{3}4^{-j}S[j] \le x - S[j]^2 \le \frac{4}{9}4^{-2j} + \frac{4}{3}4^{-j}S[j]. \tag{5}$$

That is, we have to compute $S[j]$ such that $x - S[j]^2$ is bounded according to (5). We now define a residual (or partial remainder) w so that

$$w[j] = 4^j(x - S[j]^2). \tag{6}$$

From (5) the bound on the residual is

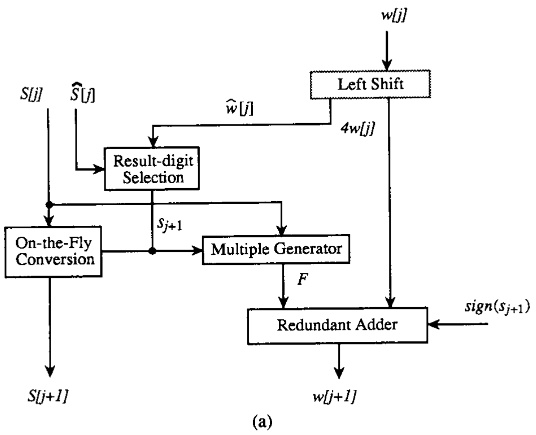
$$-\frac{4}{3}S[j] + \frac{4}{9}4^{-j} \le w[j] \le \frac{4}{3}S[j] + \frac{4}{9}4^{-j}. \tag{7}$$

Since from (1) $S[-1] = 0$, we get from (6) the initial condition

$$w[-1] = 4^{-1}x. \tag{8}$$

From (6) and (1) the basic recurrence of the square root algorithm is

$$w[j+1] = 4w[j] - 2S[j]s_{j+1} - s_{j+1}^2 4^{-(j+1)}. \tag{9}$$



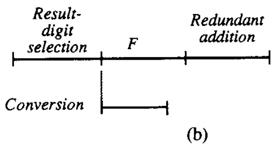


Fig. 1. (a) Square root step. (b) Timing.

B. Implementation of Square Root Step

The square root algorithm consists in performing m iterations of the recurrence (9). Moreover, each iteration consists of four subcomputations [Fig. 1(a)].

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 *Select*. 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 (7). 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,

$$s_{j+1} = Select(\hat{w}[j], \hat{S}[j]).$$

3) Formation of

$$F = -2S[j]s_{j+1} - s_{j+1}^2 4^{-(j+1)}. \tag{10}$$

4) Addition of F and $4w[j]$ to produce $w[j+1]$.

The four subcomputations are executed in sequence as indicated in the timing diagram of Fig. 1(b). Note that no time has been 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.

To have a fast recurrence step we use a carry-save adder (a signed-digit adder could also be used) and a result-digit selection 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. As indicated before, we use the symmetric digit-set

$$s_i \in \{-2, -1, 0, 1, 2\} \tag{11}$$

because it allows a simpler implementation of the adder input F (no multiple of three is required). 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.

Section III presents the result-digit selection and Section IV discusses the formation of F.

III. Result-Digit Selection Function

The selection function determines the value of the result digit s_{j+1} as a function of an estimate of the residual $w[j]$ and of an estimate of the partial result $S[j]$. Two fundamental conditions must be satisfied by a result-digit selection: *containment* and *continuity*. These conditions determine a *selection interval* for each value of s_{j+1}. We now develop these selection intervals.

A. Containment Condition and Selection Intervals

Let the bounds of the residual $w[j]$ be called $\underline{B}$ and $\bar{B}$, that is,

$$\underline{B}[j] \le w[j] \le \bar{B}[j]. \quad (12)$$

Define the *selection interval* of $4w[j]$ for $s_{j+1} = k$ 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, from the recurrence

$$L_k[j] \le 4w[j] \le U_k[j] \rightarrow \underline{B}[j+1] \le 4w[j] - 2S[j]k - k^2 4^{-(j+1)} \le \bar{B}[j+1]. \quad (13)$$

Consequently,

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

$$L_k[j] = \underline{B}[j+1] + 2S[j]k + k^2 4^{-(j+1)}. \quad (14)$$

Since $\bar{B}[j]$ and $\underline{B}[j]$ are the upper bound of the interval for $k = 2$ and the lower bound for $k = -2$, respectively, we get

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

Introducing these values in (14) we get

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

$$\underline{B}[j+1] - 2S[j] \times 2 + 2^2 4^{-(j+1)} = 4\underline{B}[j]. \quad (15)$$

This results in

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

$$\underline{B}[j] = -\frac{4}{3}S[j] + \frac{4}{9}4^{-j}. \quad (16)$$

To show that (16) is correct it is sufficient to replace in (15). Note that these bounds are identical to those obtained in (7). If this were not the case, we would use the tighter bounds.

From (14) and (16), the selection intervals are given by the expressions

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

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

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

$$U_k[j] = 2S[j]\left(k + \frac{2}{3}\right) + \left(k + \frac{2}{3}\right)^2 4^{-(j+1)} \quad (17a)$$

$$L_k[j] = 2S[j]\left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 4^{-(j+1)}. \quad (17b)$$

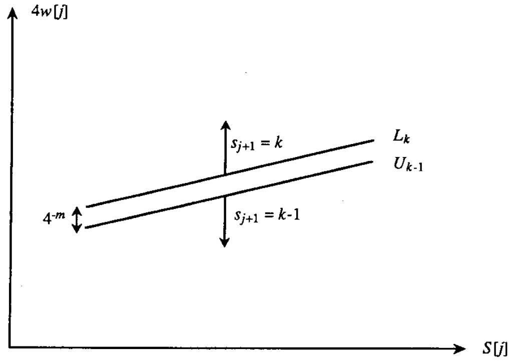


Fig. 2. Continuity.

B. Continuity Condition and Overlap Between Selection Intervals

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

$$U_{k-1} \ge L_k - 4^{-m}$$

where, as shown in Fig. 2, the term 4^{-m} appears because of the granularity of the representation of the residual with m radix-4 digits.

Moreover, to use estimates of $4w[j]$ and $S[j]$ for the result-digit selection, it is necessary to have an overlap between adjacent selection intervals. For the square root operation with digit-set $\{-2, -1, 0, 1, 2\}$ from (17) the overlap is

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

Note that the bounds, selection intervals, and the overlap depend on j.

C. Staircase Result-Digit Selection for Residual in Redundant Form

As indicated before, to have a fast recurrence step, a redundant adder is used. We now determine a staircase result-digit selection using an estimate of the partial result and an estimate of the shifted residual obtained by truncating the redundant form.

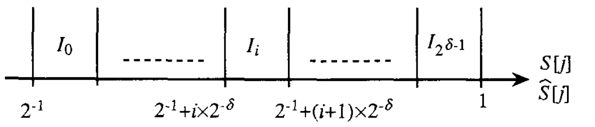
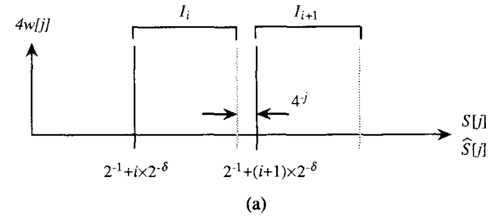


Fig. 3. Generic intervals.



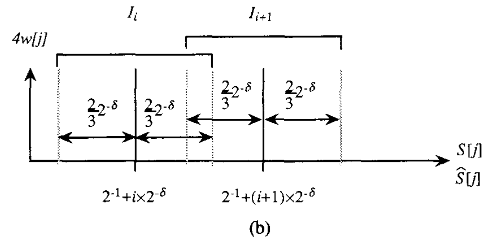


Fig. 4. Selection intervals. (a) Truncation of conventional form. (b) Truncation of signed-digit form.

As illustrated in Fig. 3, the values of the estimate of $S[j]$, called $\hat{S}[j]$, divide the range of $S[j]$ into intervals I_i (one interval per value of the estimate) so that

$$\text{If } \hat{S}[j] = 2^{-1} + i \times 2^{-\delta} \text{ then } S[j] \in I_i \qquad \text{for } 0 \le i \le 2^{\delta-1} \tag{19}$$

where δ is the number of fractional bits of the estimate $\hat{S}[j]$. Note that the value of $\hat{S}[j]$ for $i = 0$ is 2^{-1}, since the result is normalized, and that $\hat{S}[j] = 1$ for $i = 2^{\delta-1}$.

Fixed and variable estimate of S[j]: Since the result is produced one digit per iteration in signed-digit form, the following alternatives can be used to form the estimate of $S[j]$ [1] (this is in contrast to division, where the estimate is always obtained by truncating the conventional representation of the divisor):

a) Use the truncated conventional representation of $S[j]$. As discussed in Section IV, this conventional representation has to be formed on-the-fly anyhow since we use a carry-save adder for the recurrence. In this case, if $\hat{S}[j]$ is obtained by truncating $S[j]$ to δ fractional bits, then as shown in Fig. 4(a), I_i (the ith interval) is defined by

$$2^{-1} + i \times 2^{-\delta} \le S[j] < 2^{-1} + (i+1) \times 2^{-\delta}$$

$$0 \le i \le 2^{\delta-1}.$$

However, since $S[j]$ has only j fractional radix-4 digits, the upper bound of the interval is restricted so that I_i becomes

$$I_i = [2^{-1} + i \times 2^{-\delta}, 2^{-1} + (i+1) \times 2^{-\delta} - 4^{-j}). \tag{20}$$

b) Use directly the truncated signed-digit representation or (equivalently) the fixed value of $S[\lceil \delta/2 \rceil]$ (that is, the value of $S[j]$ immediately after at least δ fractional bits are produced). In this case [Fig. 4(b)], the ith interval is defined by

$$I_i = \left(2^{-1} + i \times 2^{-\delta} - \frac{2}{3} \times 2^{-\delta}, 2^{-1} + i \times 2^{-\delta} + \frac{2}{3} \times 2^{-\delta}\right). \tag{21}$$

The result-digit selection depends (slightly) on which of the two methods is used. However, since the difference is not significant, we will use method a) in our implementation.

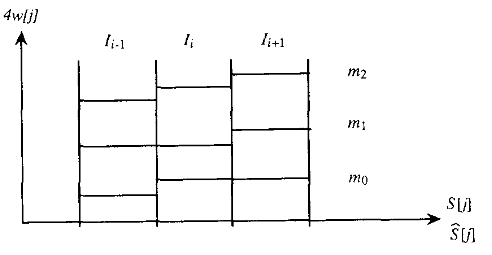


Fig. 5. Staircase result-digit selection.

Staircase Result-Digit Selection: Using the estimate of the result $\hat{S}[j]$, the result-digit selection is described by the set of *selection constants*

$$\{m_k(i) | 2^{-1} + i \times 2^{-\delta} \in \text{set of values of } \hat{S}[j],$$
$$k \in \{-2, -1, 0, 1, 2\}\}. \tag{22}$$

That is, there is one selection constant per value of $\hat{S}[j]$ and per value of the result digit. In terms of these selection constants, the *staircase result-digit selection* is defined by

$$s_{j+1} = k \text{ if } \hat{S}[j] = 2^{-1} + i \times 2^{-\delta} \text{ and}$$
$$m_k(i) \le \hat{w}[j] < m_{k+1}(i) \tag{23}$$

where $\hat{w}[j]$ is an estimate of the shifted residual $4w[j]$ obtained by truncating the redundant form to t fractional bits. This is illustrated in Fig. 5.

For the case of the residual represented in carry-save form, the error introduced by using the estimate (with respect to the truncated residual in conventional two's complement representation) is

$$0 \le \text{error} < 2^{-t}$$

as shown in Fig. 6(a). Consequently, the result-digit selection has to satisfy the relations [see Fig. 6(b)] [4]

$$m_k(i) \ge \max(L_k(I_i))$$
$$m_k(i) + 2^{-t} \le \min(U_{k-1}(I_i)). \tag{24}$$

Note that the relations depend on j, the iteration number; consequently, the selection constants can, in general, be different for different j.

From these expressions, the minimum overlap required for a feasible result-digit selection is

$$\min(U_{k-1}(I_i)) - \max(L_k(I_i)) \ge 2^{-t}. \tag{25}$$

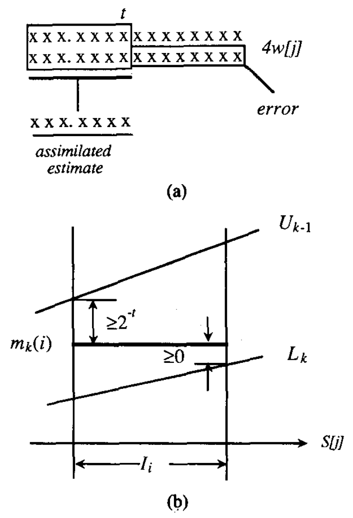


Fig. 6. (a) Error in the estimate—carry-save form. (b) Conditions on selection constants.

For method a) of formation of the estimate $\hat{S}[j]$, from (17) and (20) we get for min $(U_{k-1}(I_i))$ and max $(L_k(I_i))$

For $k > 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{1}{3}\right) + \left(k - \frac{1}{3}\right)^2 4^{-(j+1)}$$

$$\max(L_k(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{2}{3}\right) - \left(k - \frac{2}{3}\right)\left(\frac{26}{3} - k\right)4^{-(j+1)}. \quad (26a)$$

For $k \leq 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{1}{3}\right) + \left(\frac{1}{3} - k\right)\left(\frac{25}{3} - k\right)4^{-(j+1)}$$

$$\max(L_k(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 4^{-(j+1)}. \quad (26b)$$

These expressions are used to determine the result-digit selection. However, since they depend on j, a different selection function might result for different j. If we want to have a single selection function, we need to develop expressions that are independent of j. For min $(U_{k-1}(I_i))$, the term depending on j is always positive and approaching zero for large j; therefore, this term can be neglected. On the other hand, for max $(L_k(I_i))$, the term depending on j is negative for $k > 0$ and can be neglected, but is positive for $k \leq 0$ so it cannot be neglected and we have to use its maximum value (which occurs for $j = 0$). Consequently, the corresponding expressions independent of j are as follows:

For $k > 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{1}{3}\right)$$

$$\max(L_k(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{2}{3}\right). \quad (27a)$$

For $k \leq 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{1}{3}\right)$$

$$\max(L_k(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 4^{-1}. \quad (27b)$$

To determine whether a single selection function is possible we apply (25) to (27). The worst case is $i = 0$ and $k = -1$, resulting in

$$\begin{aligned}\min(U_{-2}(I_0)) - \max(L_{-1}(I_0)) &= -\frac{8}{3}(2^{-1} + 2^{-\delta}) + \frac{10}{3}(2^{-1}) - \frac{25}{9}(4^{-1}) \\ &= -\frac{13}{36} - \frac{8}{3}2^{-\delta} \geq 2^{-t}. \quad (28)\end{aligned}$$

Since there is no pair of values of t and δ that satisfy this inequality, no single selection function exists for all j. A possible alternative is to find a value J so that a single selection can be used for $j \geq J$ and then consider separately the cases for $j < J$.

For the case $j \geq J$, the same considerations given before produce

For $k > 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{1}{3}\right)$$

$$\max(L_k(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{2}{3}\right). \quad (29a)$$

For $k \leq 0$

$$\min(U_{k-1}(I_i)) = 2(2^{-1} + (i+1) \times 2^{-\delta})\left(k - \frac{1}{3}\right)$$

$$\max(L_k(I_i)) = 2(2^{-1} + i \times 2^{-\delta})\left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 4^{-(J+1)}. \quad (29b)$$

Introducing these expressions in (25) (for the worst case $i = 0$, $k = -1$) we get [similar to (28)]

$$\frac{1}{3} - \frac{25}{36}(4^{-J}) - \frac{8}{3}(2^{-\delta}) \geq 2^{-t}. \quad (30)$$

TABLE I
SELECTION INTERVALS FOR $j \geq 3$ ($t = 4$)

i $\hat{S}[j]$	0 8/16	1 9/16	2 10/16	3 11/16	4 12/16	5 13/16	6 14/16	7 15/16, 16/16
$ML_2(i), m\hat{U}_1(i)$	3/2, 77/48	5/3, 29/16	11/6, 97/48	2, 107/48	13/6, 39/16	7/3, 127/48	15/6, 137/48	8/3, 49/16
$ML_1(i), m\hat{U}_0(i)$	3/8, 29/48	5/12, 11/16	11/24, 37/48	1/2, 41/48	13/24, 15/16	7/12, 49/48	5/8, 53/48	2/3, 19/16
$ML^*_0(i), m\hat{U}_{-1}(i)$	-2/3, -7/16	-3/4, -23/48	-5/6, -25/48	-11/12, -9/16	-1, -29/48	-13/12, -31/48	-7/6, -11/16	-5/4, -35/48
$ML^*_{-1}(i), m\hat{U}_{-2}(i)$	-5/3, -25/16	-15/8, -83/48	-25/12, -91/48	-55/24, -33/16	-15/6, -107/48	-65/24, -115/48	-35/12, -41/16	-75/24, -133/48

A solution is $\delta = 4$, $J = 3$, $t = 3$. However, when applying the conditions (24), one of the selection constants is $-29/16$ [4]; consequently, it is necessary to use $t = 4$. Table I shows the corresponding limits of the intervals. The following notation is used

$$m\hat{U}_{k-1}(i) = \min(U_{k-1}(I_i)) - \frac{1}{16}$$

$$ML_k(i) = \max(L_k(I_i)).$$

As shown in the expressions (29), for $k \leq 0$ the expressions for $ML_k(i)$ contain the term $(k - 2/3)^2 4^{-4}$. This term has the following values:

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

Since these values are small (compared to $2^{-t} = 1/16$), it is simpler to present in the table

$$ML^*_k(i) = \max(L_k(I_i)) - \left(k - \frac{2}{3}\right)^2 4^{-4}$$

instead of $ML_k(i)$, with the restriction that the selection constant $m_k(i)$ cannot be equal to $ML^*_k(i)$.

Now we have to consider the cases $j < 3$. One possible approach is to have an initial PLA to determine from a truncated x the value of S [3]. Another possibility is to analyze the cases $j < 3$ and then find a combined result-digit selection (which might depend on the value of j). As indicated in the Introduction, unlike other reported implementations, we follow this second approach because it potentially results in a simpler implementation.

For all cases, we use $\delta = 4$ and $t = 4$ to match with the case $j \geq 3$.

As indicated in expression (1), since the maximum value of the radix-4 digit is 2, it is necessary to have

$$s_0 = 1$$

to be able to represent values of $s \geq 2/3$. Consequently, $S[0] = 1$. Moreover, the values of s_1 are restricted to the set $s_1 \in \{0, -1, -2\}$ (to have $s < 1$).

Therefore, for $j = 0$ and $S[0] = 1$ we obtain

$$\hat{U}_{-1} = -2 \times 1 \times (1/3) + (1/9)(1/4) - 1/16 = -(101/144)$$

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

TABLE II
SELECTION INTERVALS FOR $j = 1$

i $\hat{S}[1] = S[1]$	0 1/2	4 3/4	8 1
$L_2(i), \hat{U}_1(i)(\times 144)$	208, 256	304, 376	400, 496
$L_1(i), \hat{U}_0(i)(\times 144)$	49, 91	73, 139	97, 187
$L_0(i), \hat{U}_{-1}(i)(\times 144)$	-	-140, -80	-188, -104
$L_{-1}(i), \hat{U}_{-2}(i)(\times 144)$	-	-335, -281	-455, -377

$$\hat{U}_{-2} = -2 \times 1 \times (4/3) + (16/9)(1/4) - 1/16$$
$$= -(329)/(144)$$

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

For $j = 1$ we can have $s_2 \in \{-2, -1, 0, 1, 2\}$. Moreover, since $s_1 \in \{-2, -1, 0\}$ and $S[0] = 1$ the only possible values of $S[1]$ are 1/2, 3/4, and 1. Since $\delta = 4$ (because of the case $j \geq 3$), the estimate $S[1]$ coincides with $S[1]$ and we need consider only the values of U_{k-1} and L_k for 1/2, 3/4, and 1 and not consider min or max values in intervals. The corresponding values of L_k and U_{k-1} are given in Table II. 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 $j = 2$, since we use $\delta = 4$ and the granularity of $S[2]$ is 1/16 again $\hat{S}[2] = S[2]$, so we use exact values of $S[2]$ instead of intervals for the computation of L_k and U_{k-1}. These values are shown in Table III.

We now need to obtain a single result-digit selection for $j = 0$, $j = 1$, $j = 2$, and $j \geq 3$. To do this we combine all previous tables into Table IV.

From Table IV we can see that for all entries except for m_{-1} (8) it is possible to have a single selection function for all values of j; that is, for each pair (k, i) it is possible to find a selection constant that is inside the allowed intervals for all j. The single nonconforming case is $k = -1$, $i = 8$, and $j = 0$, for which the selection interval is $[-1520/576, -1216/576]$ which does not overlap with the corresponding intervals for $j \neq 0$. We have chosen to solve this problem by changing the estimate for $j = 0$ ($\hat{S}[0]$) from its normal value 1 to either 12/16 or 13/16; this satisfies the only two cases which are significant for $j = 0$, namely $m_0(8)$ and $m_{-1}(8)$.

In addition, to reduce the number of bits required for the estimate $\hat{S}[j]$, we convert $\hat{S}[j] = 1$ into $\hat{S}[j] = 15/16$ (for $j \neq 0$), that is we fold interval $i = 8$ onto $i = 7$; this can

TABLE III
SELECTION INTERVALS FOR $j = 2$

i $\hat{S}[2] = S[2]$	0 8/16	1 9/16	2 10/16	3 11/16	4 12/16	5 13/16	6 14/16	7 15/16	8 16/16
L_2, U_1(×576)	784, 922	880, 1042	976, 1152	1072, 1262	1168, 1372	1264, 1482	1360, 1592	1456, 1702	1552, 1812
L_1, U_0(×576)	193, 325	217, 373	241, 421	265, 469	289, 517	333, 565	357, 613	381, 661	405, 709
$L_0, \hat{U}_{-1}$(×576)	-380, -254	-428, -278	-476, -302	-524, -326	-572, -350	-620, -374	-668, -398	-716, -422	-754, -444
$L_{-1}, \hat{U}_{-2}$(×576)	-935, -815	-1045, -911	-1155,-1007	-1265, -1103	-1375, -1199	-1485, -1295	-1595, -1391	-1696, -1487	-1815, -1583

TABLE IV
SELECTION INTERVALS AND SELECTION CONSTANTS
FOR ALL VALUES OF j

i $\hat{S}[j]$	0 8/16	1 9/16	2 10/16	3 11/16	4 12/16	5 13/16	6 14/16	7 15/16	8 16/16
$ML_2(i), m\hat{U}_1(i)$ (×576)									
$j \geq 3$	864, 924	960, 1044	1056, 1164	1152, 1284	1248, 1404	1334, 1524	1440, 1644	1536, 1764	1536, 1764
$j = 2$	784, 922	880, 1042	976, 1152	1072, 1262	1168, 1372	1264, 1482	1360, 1592	1456, 1702	1552, 1812
$j = 1$	832, 1024				1216, 1504				1600, 1984
$j = 0$									-
$m_2(i)$	3/2	7/4	2	2	9/4	5/2	5/2	11/4	3
$m_2(i)$×576	864	1008	1152	1152	1296	1440	1440	1584	1728
$ML_1(i), m\hat{U}_0(i)$ (×576)									
$j \geq 3$	216, 348	240, 396	264, 444	288, 492	312, 540	336, 588	360, 636	384, 684	384, 684
$j = 2$	193, 325	217, 373	241, 421	265, 469	289,517	333, 565	357, 613	381, 661	405, 709
$j = 1$	196, 364				292, 556				388, 748
$j = 0$									-
$m_1(i)$	1/2	1/2	1/2	1/2	3/4	3/4	1	1	1
$m_1(i)$×576	288	288	288	288	432	432	576	576	576
$ML^*_0(i), m\hat{U}_{-1}(i)$ (×576)									
$j \geq 3$	-384, -252	-432, -306	-570, -300	-528, -324	-576, -348	-624, -372	-672, -396	-720, -420	-720,-420
$j = 2$	-, -	-428, -278	-476, -302	-524, -326	-572, -350	-620, -374	-668, -398	-716, -422	-754, -444
$j = 1$	-				-560, -320				-752, -416
$j = 0$									-704, -404
$m_0(i)$	-1/2	-5/8	-3/4	-3/4	-3/4	-1	-1	-1	-1
$m_0(i)$×576	-288	-360	-432	-432	-432	-576	-576	-576	-576
$ML^*_{-1}(i), m\hat{U}_{-2}(i)$ (×576)									
$j \geq 3$	-960, -900	-1080, -996	-1200,-1092	-1320,-1188	-1440,-1284	-1560,-1380	-1680,-1476	-1800,-1596	-1800,-1596
$j = 2$	-,-	-1045,-911	-1155,-1007	-1265,-1103	-1375,-1199	-1485,-1295	-1595,-1391	-1696,-1487	-1815,-1583
$j = 1$	-				-1340,-1124				-1820,-1408
$j = 0$									-1520,-1216
$m_{-1}(i)$	-13/8	-7/4	-2	-17/8	-9/4	-5/2	-11/4	-23/8	*
$m_{-1}(i)$×576	-936	-1008	-1152	-1224	-1296	-1440	-1584	-1656	

be done because the selection intervals are satisfied. Therefore, the estimate used in the selection $\hat{S} = (\hat{S}_1, \hat{S}_2, \hat{S}_3, \hat{S}_4)$ is obtained as follows:

$$(\hat{S}_1, \hat{S}_2, \hat{S}_3, \hat{S}_4) = \begin{cases} (1, 1, 0, -) & \text{if } (j = 0) \\ (1, 1, 1, 1) & \text{if } (A_0 = 1) \text{ and } (j \neq 0) \\ (1, A_2, A_3, A_4) & \text{if } (j \neq 0) \end{cases}$$

where $(A_0, A_1, A_2, A_3, A_4)$ are the most significant bits of A, the conventional representation of $S[j]$ (as discussed in Section IV), and for $j = 0$, $A_0 = 1$ and $A_2 = A_3 = A_4 = 0$.

The resulting selection function is given in Table V and its implementation shown in Fig. 7. From the possible selection constants, those which minimize the number of fractional bits in their representation are chosen. Since the selection constants are of the form $D \times 2^{-3}$ (D integer), 3 fractional bits of the estimate of the shifted residual are used in the result-digit

TABLE V
RESULT-DIGIT SELECTION FOR ALL j

i	0	1	2	3	4	5	6	7
$\hat{S}[j]$	8/16	9/16	10/16	11/16	12/16	13/16	14/16	15/16
$m_2(i)$	3/2	7/4	2	2	9/4	5/2	5/2	11/4
$m_1(i)$	1/2	1/2	1/2	1/2	3/4	3/4	1	1
$m_0(i)$	-1/2	-5/8	-3/4	-3/4	-3/4	-1	-1	-1
$m_{-1}(i)$	-13/8	-7/4	-2	-17/8	-9/4	-5/2	-11/4	-23/8

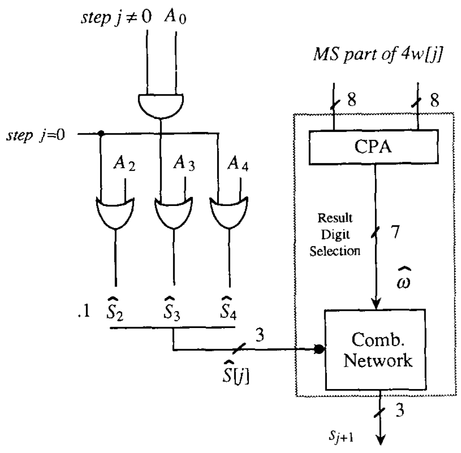


Fig. 7. Result-digit selection implementation.

selection. Moreover, since $t = 4$, 4 fractional bits of the shifted carry-save residual are used, resulting in a total of 8 bits (4 integer bits and 4 fractional bits).

Alternative implementations exist for the combinational network of the result-digit selection. One possibility is to use a ten-input/three-output PLA; on the other hand, some preprocessing can be done to reduce the size of the PLA, as proposed for example in [5].

IV. Generation of Adder Input F

As part of the implementation of the recurrence (9) it is necessary to form the adder input F with value

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

Since the digit of the result is produced in signed-digit form, the partial result $S[j]$ is also in this form. However, for the case we are considering, which uses a carry-save adder, the input F has to be in two's complement representation. Consequently, $S[j]$ is converted to this form on-the-fly using a variation of the scheme presented in [2]. It requires that two conditional forms $A[j]$ and $B[j]$ are kept, such that

$$A[j] = S[j]$$

$$B[j] = S[j] - 4^{-j}.$$

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

$$A[j+1] = \begin{cases} A[j] + s_{j+1}4^{-(j+1)} & \text{if } s_{j+1} \geq 0 \\ B[j] + (4 - |s_{j+1}|)4^{-(j+1)} & \text{otherwise.} \end{cases} \tag{31a}$$

$$B[j+1] = \begin{cases} A[j] + (s_{j+1} - 1)4^{-(j+1)} & \text{if } s_{j+1} > 0 \\ B[j] + (3 - |s_{j+1}|)4^{-(j+1)} & \text{otherwise.} \end{cases} \tag{31b}$$

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. This implementation is shown in Fig. 8. In terms of these forms, the value of F is given by the following expressions:

For $s_{j+1} > 0$

$$\begin{aligned} F[j] &= -2S[j]s_{j+1} - s_{j+1}^2 4^{-(j+1)} \\ &= -(2A[j] + s_{j+1}4^{-(j+1)})s_{j+1}. \end{aligned} \tag{32a}$$

For $s_{j+1} < 0$

$$\begin{aligned} F[j] &= 2S[j]|s_{j+1}| - s_{j+1}^2 4^{-(j+1)} \\ &= 2(B[j] + 4^{-j})|s_{j+1}| - s_{j+1}^2 4^{-(j+1)} \\ &= (2B[j] + (4 - |s_{j+1}|)4^{-(j+1)})|s_{j+1}|. \end{aligned} \tag{32b}$$

Note that these expressions are obtained by concatenation and multiplication by a radix-4 digit. The resulting bit-strings are given in Table VI, where $a \cdots aa$ and $b \cdots bb$ are the bit-strings representing $A[j]$ and $B[j]$, respectively (shifted one position). The location of the trailing string is also controlled by the moving 1 of register K.

Fig. 8 also shows a block to perform on-the-fly rounding, as described in [3].

V. Overall Algorithm, Implementation, and Timing

The overall algorithm is as follows (not including rounding):

```
begin Square root
 Initialization
  $w[0] = x - 1$      *since $w[-1] = S[-1] = 0$ and
                       $s_0 = 1$ (Load $x$ and make 1 the sign
                       position)*
  $A[0] \leftarrow 1.000...000$     *$S[0] = 1$*
  $B[0] \leftarrow 0.000...000$     *$B[0] = A[0] - 1$*
  $K[0] \leftarrow 0.100...000$
 Iterations
  for $j = 0$ to $m$
   begin
     $s_{j+1} =$ SELECT($\hat{w}[j]$, $\hat{S}[j]$)     *see Table V*
     $F[j] = f(A[j], B[j], s_{j+1})$        *see Table VI*
     $w[j+1] \leftarrow 4w[j] + F[j]$
     $A[j+1] \leftarrow g_a(A[j], B[j], s_{j+1})$
                                   *see expression (31a)*
     $B[j+1] \leftarrow g_b(A[j], B[j], s_{j+1})$
                                   *see expression (31b)*
     $K[j+1] \leftarrow$ shift-right ($K[j]$)
   end for
 Result
  $s = S[m] = A[m]$
end Square root
```

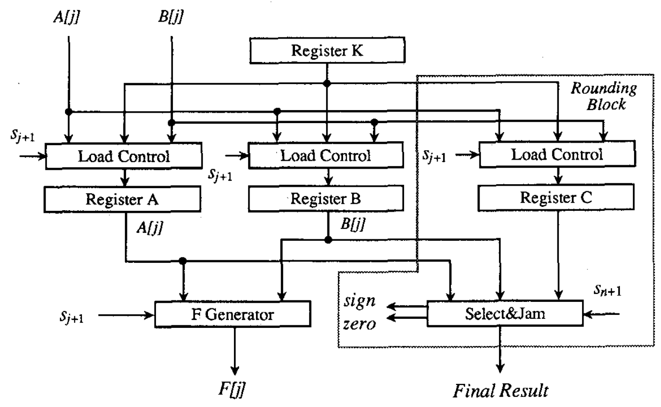


Fig. 8. Network for generating F and rounding.

TABLE VI
GENERATION OF $F[j]$

	$F[j]$		
s_{j+1}	Value	Value	Bit-string
0	0	0	$0\cdots00000$
1	$-2S[j]-4^{-(j+1)}$	$-2A[j]-4^{-(j+1)}$	$\bar{a}\cdots\overline{aa}111$
2	$-4S[j]-4\times4^{-(j+1)}$	$-4A[j]-4\times4^{-(j+1)}$	$\bar{a}\cdots\bar{a}1100$
-1	$2S[j]-4^{-(j+1)}$	$2B[j]+7\times4^{-(j+1)}$	$b\cdots bb111$
-2	$4S[j]-4\times4^{-(j+1)}$	$4B[j]+12\times4^{-(j+1)}$	$b\cdots b1100$

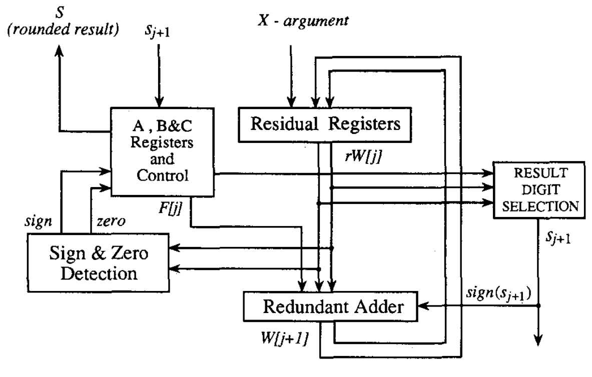


Fig. 9. Block diagram of the square root scheme (mantissa part).

The overall implementation at the block-diagram level is shown in Fig. 9. The cycle time is

$$\begin{aligned} T_{\text{cycle}} &= t_{\text{result\_digit\_select}} && \{\text{8-bit CPA + 10-input comb. net}\} \\ &+ t_{F\_\text{generate}} && \{\text{4-to-1 multiplexer}\} \\ &+ t_{\text{CSA}} && \{\text{3-to-2 carry-save adder}\} \\ &+ t_{\text{load}} && \{\text{register loading}\}. \end{aligned}$$

This is comparable to the cycle time of a radix-4 division with carry-save adder.

VI. Conclusions

We have shown a radix-4 square root algorithm that does not require an initial PLA. More precisely, this PLA is replaced by the four gates of Fig. 7. This is of theoretical interest and also produces a simpler implementation. It should be noted that the number of iterations required in this algorithm is the same as in those that use an initial PLA. That is, the fact that the initial bits of the result are obtained from the PLA does not reduce the number of iterations, since the residual has to be obtained from the iterations. Consequently, the simplification in implementation is obtained without time penalty.

Acknowledgment

We thank P. Montuschi for helpful suggestions.

References

[1] L. Ciminiera and P. Montuschi, "Higher radix square rooting," Intern. Rep., Politecnico di Torino, Dipartimento di Automatica e Informatica, I.R. DAI/ARC 4-87, Dec. 1987.

[2] M. D. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Trans. Comput.*, vol. C-36, no. 7, pp. 895-897, July 1987.

[3] ——, "On-the-fly rounding for division and square root," in *Proc. 9th Symp. Comput. Arithmetic*, 1989, pp. 169-173.

[4] ——, "Division and square root algorithms and implementations," monograph in preparation.

[5] J. Fandrianto, "Algorithm for high speed shared radix-4 division and radix-4 square root," in *Proc. 8th Symp. Comput. Arithmetic*, 1987, pp. 73-79.

[6] J. B. Gosling and C. M. S. Blakeley, "Arithmetic unit with integral division and square-root," *IEE Proc.*, vol. 134, pt. E, no. 1, pp. 17-23, Jan. 1987.

[7] S. Kuninobu *et al.*, "Design of high speed MOS multiplier and divider using redundant binary representation," in *Proc. 8th Int. Symp. Comput. Arithmetic*, 1987, pp. 80-86.

[8] P. Montuschi and L. Ciminiera, "On the efficient implementation of higher radix square root algorithms," in *Proc. 9th Symp. Comput. Arithmetic*, Sept. 1989, pp. 154-161.

[9] M. B. Vineberg, "A radix-4 square-rooting algorithm," Rep. 182, Dep. Comput. Sci., Univ. of Illinois, Urbana-Champaign, June 1965.

[10] J. H. Zurawski and J. B. Gosling, "Design of a high-speed square root multiply and divide unit," *IEEE Trans. Comput.*, vol. C-36, pp. 13-23, Jan. 1987.

Miloš D. Ercegovac (M'85), for a photograph and biography, see the June 1990 issue of this Transactions, p. 740.

Tomas Lang, for a biography, see the June 1990 issue of this Transactions, p. 740.