Correctly Rounded Reciprocal Square–root by Digit Recurrence and Radix-4 Implementation*

Tomás Lang
Dept. Electrical and Computer Eng.
University of California at Irvine.
Irvine, CA 92697. USA
tlang@uci.edu

Elisardo Antelo
Dept. Electronic and Computer Eng.
University of Santiago
Santiago de Compostela. SPAIN
elisardo@dec.usc.es

Abstract

In this work we present a reciprocal square–root algorithm by digit recurrence and selection by a staircase function, and the radix–4 implementation. As similar algorithms for division and square–root, the results are obtained correctly rounded in a straightforward manner (in contrast to existing methods to compute the reciprocal square–root). Although apparently a single selection function can only be used for $j \geq 2$ (the selection constants are different for $j = 0$, $j = 1$ and $j \geq 2$), we show that it is possible to use a single selection function for all iterations. We perform a rough comparison with existing methods and we conclude that our implementation is a low hardware complexity solution with moderate latency, specially for exactly rounded results.

1. Introduction

The reciprocal square-root operation is important for applications, such as graphics and scientific computations. We here present a digit-recurrence algorithm and a radix-4 implementation. Digit recurrence is one of the preferred methods for division and square root. It is therefore natural to explore the possibility of a method of this type for reciprocal square root. Previously we had used this method for a very-high radix implementation for $\sqrt{x/d}$, which includes reciprocal square root [1]. However, this approach requires scaling operations, which make the implementation complex and significantly slower than the corresponding implementation for division. We here develop the low-radix case, using a "standard" selection function, instead of the selection by rounding used for very-high radix. We concentrate specifically on the radix-4 case, but the development is general so that radix-8 and radix-16 with overlapped radix-4 can be considered.

As discussed further below, these digit-recurrence implementations allow simple correct rounding, which is hard to achieve for other approaches.

One aspect we develop fully for the radix-4 implementation is to determine a single selection function for all iterations. This requires an extensive analysis, since as in square root the direct use of the selection function for large j to the first iterations does not produce convergence [5].

The implementations of reciprocal square root used today depend on the precision required[1]: for low precision (up to 10 bits) a direct table method, for medium precision (up to 16 bits) a bipartite table method [3] [13], for single precision floating-point (24 bits) a linear approximation method or an iterative method with quadratic convergence [2] [4] [7] [11] [14] [15] [17], and for double precision floating-point (53 bits) an iterative method with quadratic convergence or a table-driven polynomial approximation [2] [6] [8] [11] [16]. Except for the direct table approach, these methods provide an approximation for which it is hard to produce a correctly rounded result. To obtain exactly rounded results with these schemes, one possible method consist in computing the remainder $(1 - dP^2)$ (P is the approximation truncated to the rounding bit of $1/\sqrt{d}$), and then correct P if necessary.

We performed a rough comparison of representative instances of these methods (only for single and double precision) and our method (a radix–4 algorithm). We compared the designs for both exactly rounded results and for one ulp of precision. From the comparison we

*E. Antelo has been partially supported by Xunta de Galicia under project PGIDT99PXI20601A.

[1] Some of the references do not implement specifically the reciprocal square–root, but they can be adapted easily.

0-7695-1150-3/01 $10.00 © 2001 IEEE

concluded that our implementation is a low hardware solution with moderate latency, specially for exactly rounded results. The latency can be further reduced if a radix–8 or radix–16 instance is used. Further details of the comparison are given in Section 4.1.

Much of the notation and development is similar to that for division and square root. We assume that the reader is familiar with these algorithms [5].

2. Operands, result and recurrence

We want to obtain $P = \frac{1}{\sqrt{d}}$ for operand and result represented in floating point using the IEEE–754 standard, namely with significand in the range $[1, 2)$. To achieve this range of the result it is necessary either to modify the range of the operand or to postnormalize the result. Moreover, the exponent of the result is obtained by making even the exponent of the operand (and scaling the significand accordingly) and dividing the resulting exponent by two. Because of these considerations, there are several alternatives for the range to which the significand of the operand can be scaled, before performing the actual digit-recurrence algorithm. The best choice depends on the way the algorithm is performed; for the input operand we selected the range $[1/4, 1)$. We discuss the reasons of this choice at the end of this section. Specifically, calling the operand significand d we obtain a scaled operand significand z such that

$$z = \begin{cases} 2^{-1}d & (z \geq 1/2) \quad \text{if odd exponent} \\ 2^{-2}d & (z < 1/2) \quad \text{if even exponent} \end{cases} \tag{1}$$

As a result, $z \in [1/4, 1)$ and $P \in (1, 2]$[2].

In a radix-r digit recurrence one digit of the result is obtained each iteration. Consequently, the result is represented in radix $r = 2^b$ by

$$P = \sum_{i=0}^{n-1} p_i r^{-i} \tag{2}$$

As is usual in these type of algorithms, we use a signed–digit representation with $-a \leq p_i \leq a$ and $r/2 \leq a \leq (r-1)$ (not over–redundant). The corresponding redundancy factor is $\rho = a/(r-1)$.

In the type of recurrences we consider, the digits are obtained most–significant digit first. Consequently, after j iterations the partial result is

$$P[j] = P[0] + \sum_{i=1}^{j} p_i r^{-i} \tag{3}$$

[2]This is in the range of the IEEE standard, except for the case $P = 2$. However, this corresponds to $d = 1$ and even exponent. Consequently, to avoid a postnormalization this trivial case can be handled separately.

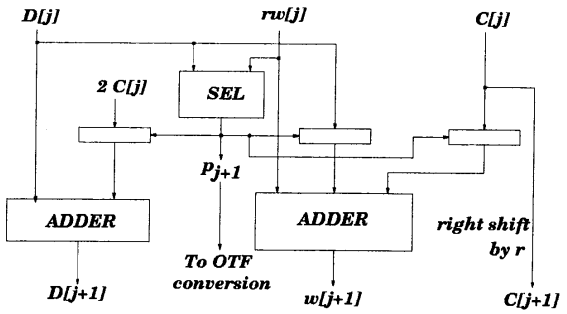

Figure 1. Block diagram.

Note that we replace the digit p_0 by the initial value $P[0]$. Since $P \in [1, 2)$, we can only choose a single value for $P[0]$ for the whole range of P for the case $\rho = 1$. In other cases, we can choose $P[0] = 1$ for $P < 1 + \rho$ and $P[0] = 2$ for $P > 2 - \rho$. We make, [3]

$$P[0] = \begin{cases} 1 & \text{if odd exponent } (z \geq 1/2,\ P \leq \sqrt{2}) \\ 2 & \text{if even exponent } (z < 1/2\ ,P > \sqrt{2}) \end{cases} \tag{4}$$

The digit recurrence to compute P is as follows [1]

$$w[j+1] = rw[j] - P[j]p_{j+1}z - 2^{-1}p_{j+1}^2 r^{-(j+1)}z \tag{5}$$

with

$$w[0] = 2^{-1}(1 - zP[0]^2) = \begin{cases} (1/2)(1-d) & \text{if even} \\ (1/2)(1-d/4) & \text{if odd} \end{cases} \tag{6}$$

To simplify the description and the implementation we define the following variables:

$$D[j] = zP[j] \quad \text{and} \quad C[j] = 2^{-1}r^{-(j+1)}z \tag{7}$$

Using these variables, we obtain the recurrences

$$w[j+1] = rw[j] - p_{j+1}D[j] - p_{j+1}^2 C[j] \tag{8}$$

$$D[j+1] = D[j] + 2p_{j+1}C[j] \tag{9}$$

$$C[j+1] = r^{-1}C[j] \tag{10}$$

where

$$D[0] = zP[0] = (1/2)d \tag{11}$$

$$C[0] = 2^{-1}r^{-1}z = \begin{cases} 2^{-3}r^{-1}d & \text{if exponent even} \\ 2^{-2}r^{-1}d & \text{if exponent odd} \end{cases} \tag{12}$$

Figure 1 shows the block diagram of these recurrences.

The bound for $w[j]$ in terms of $D[j]$ is given by

$$-\rho D[j] + 2^{-1}\rho^2 r^{-j}z < w[j] < \rho D[j] + 2^{-1}\rho^2 r^{-j}z \tag{13}$$

[3]This can be done for $\rho \geq 2 - \sqrt{2}$.

Table 1. Alternative ranges

scaled operand	result	$P[0]$	postnor.	num. of iters.	$D[j]$	sel. bits
$[1,4)$	$(1/2,1]$	1	No	$\lceil (n+1)/b \rceil$	$[1,4)$	largest
$[1/2,2)$	$(1/\sqrt{2},\sqrt{2}]$	1	Yes	$\lceil (n+1)/b \rceil$	$[1/2,2)$	medium
$[1/4,1)$	$(1,2]$	1,2	No	$\lceil n/b \rceil$	$[1/2,1)$	smallest

In terms of these variables, p_{j+1} is selected so that $w[j+1]$ satisfies the bound of (13). Consequently, the selection function has as arguments $rw[j]$, $D[j]$, and z. As shown in Figure 1, we develop a selection function depending only on $rw[j]$ and $D[j]$. That is,

$$p_{j+1} = SEL(rw[j],\ D[j])$$

As indicated before, there are several choices for the ranges of operand and result. The characteristics of the implementation influenced by the choice are: (1) value of $P[0]$, (2) need of postnormalization and number of iterations, (3) handling of special cases, (4) range of $D[j]$, and (5) complexity of the selection function. In Table 1 we summarize these characteristics for three possible alternatives. From these alternatives we have chosen the scaled operand $1/4 \leq z < 1$ so that $1 < P \leq 2$ (for in depth explanation on the reasons for this choice see [10]). To obtain a normalized result it is only necessary to detect the case $d = 1$ and set $P = 1$.

2.1. Rounding

As for other digit-recurrence implementations, in this case it is simple to include correct rounding: it suffices to determine one additional result bit and assure that the remainder (last residual) is positive. Moreover, the zero remainder is used to determine an exact result. For the correct determination of the sign and zero of the last residual the datapath has to have about 2n fractional bits (see Figure 2). We discuss the effect of this on the area later.

3. Direct implementation and retiming

To reduce the delay, the iteration step is implemented with adder structures using redundant adders. If carry–save representation is used, the addition to produce $w[j+1]$ results in a 5–to–2 adder structure. Moreover, the selection is done using low–precision estimates $\widehat{rw}[j]$ (with t fractional bits) and $\widehat{D}[j]$ (with s fractional bits), which are obtained from the assimilation of some bits[4] of the carry-save representation. That is,

$$p_{j+1} = SEL(\widehat{rw}[j],\ \widehat{D}[j])$$

The resulting critical path corresponds to the sum of the following component delays: selection function, including the assimilation, multiple generation, and 5-to-2 carry-save addition.

To reduce the iteration delay, we retime the iteration[5]. The corresponding implementation is shown in Figure 2 (this is an instance of the radix–4 implementation discussed in Section 4). As shown, instead of having the selection at the beginning of the iteration, it is moved to the end and the selection of p_{j+2} is performed using estimates of $rw[j+1]$ and $D[j+1]$. This retiming permits the assimilation of $D[j+1]$ and a partial assimilation of $w[j+1]$ in parallel with the selection function. This has the following consequences:

- The 5–2 adder can be replaced by a 4–2 adder.
- Since the selection function depends on an estimate of $w[j+1]$, in the calculation of this estimate it is possible to use the partially assimilated $w[j]$ (with h fractional bits). Consequently, a 3-2 adder is used in the selection function path. The bound on h is given by $h \geq 1 + b + t + 2$ (which is the number of leading bits of $rw[j]$ used in the selection).
- Similarly, the selection function depends on an estimate of $D[j+1]$, which can be obtained also with a partial assimilation (of $x > s$ fractional bits).

From Figure 2 the possible critical paths are

- The calculation of p_{j+2}: delay of multiple generation + 3–2 adder + CPA(t)+ selection function OR delay of multiple generation + CPA(x) + selection function. The delay of this path is similar to that of division.
- The calculation of $w[j+1]$ (assimilated part): delay of multiple generation + 4–2 adder + CPA(h)

[4] The number of bits of these estimates to assure convergence is determined in Section 5.

[5] A similar retiming was used for division before the utilization of redundant adders (it was called a scheme with prediction) [12]. Recently, it has also been used to reduce the energy for division and square root [9].

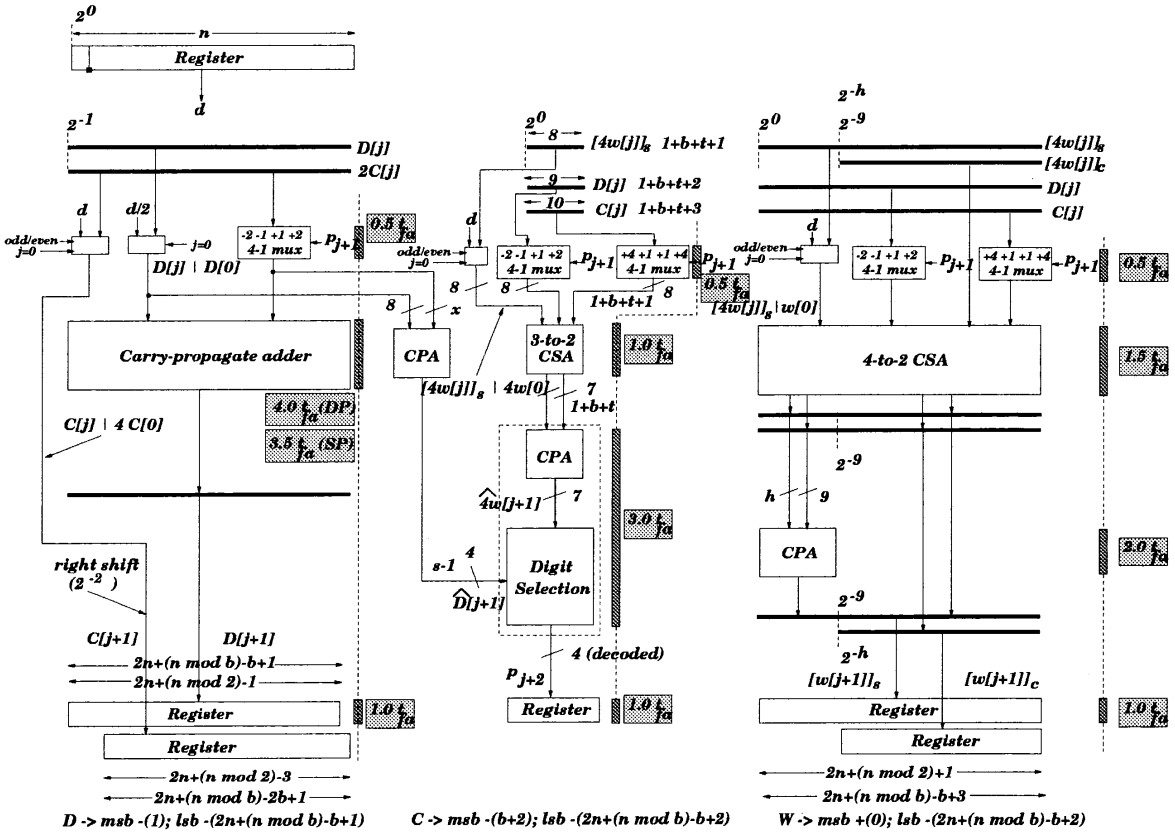

Figure 2. Block diagram of the radix–4 modified implementation (widths also specified for $r = 2^b$; msb (lsb) $\pm(a)$ means most (least) significant bit with weight $2^{\pm a}$).

- The calculation of $D[j+1]$: CPA.

In terms of the hardware required, if redundant adders are used, the 5-to-2 and 3-to-2 adders of Figure 1 have been replaced by a 4–to–2 adder plus one conventional carry-propagate adder. Moreover, two additional partial assimilations are included.

3.1. Complete execution

We now describe the complete execution of the operation using the implementation of Figure 2. It consists of the following cycles:

- Cycle 1: Initialization and computation of p_1

$$
\begin{aligned}
w[0] &= \begin{cases} (1/2)(1-d) & \text{if exponent even} \\ (1/2)(1-d/4) & \text{if exponent odd} \end{cases} \\
D[0] &= (1/2)d \\
C[0] &= \begin{cases} 2^{-3}r^{-1}d & \text{if exponent even} \\ 2^{-2}r^{-1}d & \text{if exponent odd} \end{cases} \\
P[0] &= \begin{cases} 1 & \text{if odd exponent} \\ 2 & \text{if even exponent} \end{cases} \\
p_1 &= SEL(\widehat{rw[0]}, \widehat{D[0]})
\end{aligned}
$$

- Cycles 2 to $N = \lceil n/b \rceil$ Iterations $(0 \le j \le N-2)$[6]

$$
\begin{aligned}
P[j+1] &= CONVERT(P[j], p_{j+1}) \\
w[j+1] &= rw[j] - D[j]p_{j+1} - C[j]p_{j+1}^2 \\
D[j+1] &= D[j] + 2p_{j+1}C[j] \\
C[j+1] &= r^{-1}C[j] \\
p_{j+2} &= SEL(\widehat{rw[j+1]},\ \widehat{D[j+1]})
\end{aligned}
$$

- Cycle $N+1$ Obtain last residual, correct and round

$$w[N] = rw[N-1] - D[N-1]p_N - C[N-1]p_N^2$$

if $w[N] < 0$ then $p_N = p_N - 1$

$P = RoundConvert(P[N-1], p_N)$ [5]

[6]The conversion is done on-the-fly [5].

Table 2. Example of execution

Number of bits of input operand (in interval [1,2)):8
d=1.0100101 × 2^4 = 0.0101001010000000 × 2^6
Initialization for on–the–fly conv.&round. $P = (2)_4$ $PM = (1)_4$

INITIALIZATION AND COMPUTATION OF p_1

WS[0]=1.1101101100000000 D[0] = 0.101001010000000 $\hat{D} = 20/32$, $4\hat{w} = -10/16$
WC[0]=0.0000000000000000 C[0] = 0.0000101001010000 $p_1 = -1$, $P = (1.3)_4$, $PM = (1.2)_4$

ITERATION j=0

WS[1]=0.0000011010101111 D[1] = 0.100100000110000 $\hat{D} = 18/32$, $4\hat{w} = 1/16$,
WC[1]=0.0000000000000001 C[1] = 0.0000001010010100 $p_2 = 0$, $P = (1.30)_4$, $PM = (1.23)_4$

ITERATION j=1

WS[2]=0.0001101010110000 D[2] = 0.100100000110000 $\hat{D} = 18/32$, $4\hat{w} = 6/16$,
WC[2]=0.0000000000010000 C[2] = 0.0000000010100101 $p_3 = 1$, $P = (1.301)_4$, $PM = (1.300)_4$

ITERATION j=2

WS[3]=1.1101100111000100 D[3] = 0.100100011010101 $\hat{D} = 18/32$, $4\hat{w} = -11/16$,
WC[3]=0.0000000000110111 C[3] = 0.0000000000101001 $p_4 = -1$

SIGN OF LAST REMAINDER AND ROUNDING:

WS[4]=1.1111100010000000 Sign of last remainder = - $PR = (1.3002)_4$
WC[4]=0.0000000011101101 EXPONENT COMPUTATION: $1/\sqrt{2^6} = 2^{-3}$
RESULT OF THE ALGORITHM: $(1.3002)_4 \times 2^{-3}$

Selection function (radix–4 algorithm). The selection is performed as follows: $p_{j+1} = k$ if $\hat{D} = D_L[i]$ ($\hat{D}$: 5 ms fractional bits of $D[j]$) and $m_k \leq 4\hat{w} < m_{k+1}$ ($4\hat{w}$: 7 msb of $4w[j]$).
$D_L[i]$ scaled by 32 and m_k scaled by 16.

$D_L[i]$	16	17	18	19	20	21	22	23	24	25	26†	27	28	29	30	31
m_{-1}	-13	-14	-14	-15	-16	-17	-18	-18	-19	-19	-20	-21	-23	-23	-24	-25
m_0	-5	-5	-5	-6	-6	-6	-7	-7	-7	-8	-8	-8	-9	-9	-9	-10
m_1	3	4	4	4	4	4	4	5	5	7	7	6	5	6	8	8
m_1	12	13	14	14	15	15	16	17	17	18	19	19	20	21	21	22

† For $j = 0$ use the constants of the row $D_L[i] = 24/32$.

In Table 2 we show an example of the execution of the algorithm at the bit level for radix–4. The selection function is shown at the bottom of the table. This selection function is obtained in Section 5 (Table 9). The example is for an input operand of 8 bits. We verified the algorithm with an exhaustive simulation for input operands of 24 bits.

4. Radix-4 implementation

We now present the radix-4 implementation (see Figure 2). For this instance of the algorithm,

- From Section 5, $s = 5$, $t = 4$, $x = 8$ and $h = 9$.
- The generation of multiples $p_{j+1}D[j]$ and $2p_{j+1}C[j]$ are each implemented with a 4-to-1 (decoded) multiplexer.
- The generation of p_{j+1}^2 is performed by two OR gates and that of $p_{j+1}^2 C[j]$ by a 2-to-1 multiplexer.

4.1. Estimation of delay and area

We now estimate the delay and the hardware complexity of our design, and compare with other units. For the estimation of delays we use a rough model in terms of a full–adder delay. For the hardware complexity we just list the wide datapath elements.

The possible critical paths are as shown in Figure 2, including the delays of the components. We conclude that the critical path goes through the selection function and an estimate of the cycle time is 5.5 t_{fa}. In comparison with a division unit, the increase in the critical path corresponds to the delay of one 3–2 csa (1 t_{fa}). In terms of hardware complexity the reciprocal square–root unit uses a 4–2 csa instead of a 3–2 csa, and one additional multiple generator (multiplexer), both in the w datapath. Moreover the reciprocal square–root unit needs a datapath for $D[j]$ (two registers, multiple generator and cpa). For exactly rounded result, the reciprocal square–root unit needs about twice the width of the datapath.

In Table 3 we show the latency and hardware complexity of our architecture in comparison with other representative methods[7]. We compare for IEEE–754 single (SP) and double precision (DP) formats.

We evaluated the designs for exactly[8] and non exactly rounded results. We assumed that the hardware is reused when possible (specially for the case of exactly rounded results).

From the table we conclude that our implementation is a low hardware solution with moderate latency, specially for exactly rounded results. The exact rounding is straightforward in the proposed algorithm (as for additive division an square–root), whereas this results in a complex process for the other alternatives. The latency can be reduced further considering a higher radix (radix–8 or radix–16) using similar techniques as used before for division and square–root.

5. Selection function

We now derive the selection function for the scheme of Figure 2. This selection function is of the staircase form, as used for division and square root. We begin by determining the conditions for the radix-r case and then obtain the selection function for $r = 4$. Although apparently a single selection function can only be used for $j \geq 2$ (and the iterations $j = 0$ and $j = 1$ require special functions), we show that it is possible to use a single function for all j. This is based on the fact that for $j = 0$ and $j = 1$ the possible pairs $(w[j],\ D[j])$ are restricted.

5.1. Conditions for radix-r case

We call[9] $(U_k[j], L_k[j])$ the interval of $rw[j]$ for which $p_{j+1} = k$ can be selected. This interval is obtained from the recurrence and the bounds of $w[j+1]$, resulting in [10],

$$U_k[j] = (k+\rho)D[j] + 2^{-1}(k+\rho)^2 r^{-(j+1)} z \quad (14)$$

$$L_k[j] = (k-\rho)D[j] + 2^{-1}(k-\rho)^2 r^{-(j+1)} z \quad (15)$$

The staircase implementation has the following characteristics:

- The domain of the argument $D[j]$ is divided into intervals $[D_L(i), D_U(i))$, such that[10]

$$D_L(i) = \frac{1}{2} + i \times 2^{-s} \text{ and } D_U(i) = D_L(i) + 2^{-s} + 2^{-x}$$

- In interval i of $D[j]$ the selection function is defined such that $p_{j+1} = k$ if $m_k(i) \leq \widehat{rw}[j] < m_{k+1}(i)$[11]
- For convergence, the conditions for the comparison constants are[12]

$$\max_i(L_k[j]) \leq m_k(i) \leq \min_i(U_{k-1}[j]) - 2^{-t} \quad (16)$$

where the maximum and minimum are evaluated in interval i of $D[j]$.

From (16) we obtain a necessary condition that relates the parameters s, x, and t. Namely, for all i

$$\min_i(U_{k-1}[j]) - 2^{-t} - \max_i(L_k[j]) \geq 0 \quad (17)$$

5.2. Radix-4 with $a = 2$

We now concentrate on the radix-4 case and a quotient digit set with $a = 2$. For this case, (16) results in

$$\max_i\left(D[j](k-\frac{2}{3}) + \frac{1}{8}4^{-j}z(k-\frac{2}{3})^2\right) \leq m_k(i) \leq \min_i\left(D[j](k-\frac{1}{3})\right) - 2^{-t} \quad (18)$$

Notice that z appears in this expression. We now determine a bound for z in terms of $D[j]$. Since $D[j] = zP[j]$, we have[13]

[7]For each precision we compare with a fast method (linear interpolation for SP, and reference [6] for DP) and a common microprocessor implementation (reference [11]). There are other instances that could be used, but we estimate that similar conclusions would result.

[8]For the other designs we assumed a similar method of that for division and square–root to obtain exactly rounded results, adapted to the reciprocal square–root case. That is, P is truncated to the $n+1$ fractional bit, obtain P^2, multiply by d and compare with one (this way the sign of $1 - d\ P^2$ is obtained), adjust P and round. Since P^2 has about $2n$ bits, we assume that the multiplication $d\ P^2$ is performed in two steps of a $n \times n$ multiplication. Therefore three multiplications are needed for the rounding.

[9]Although for the retimed implementation we used $rw[j+1]$ and $D[j+1]$ to determine the digit, to simplify the presentation of this section we use the index j.

[10]Note that in the retimed implementation (Figure 2) we compute the estimate of $D[j+1] = D[j] + 2p_{j+1}C[j]$ by performing a truncated addition up to x fractional bits and truncation of the result to t fractional bits. Therefore the intervals overlap by 2^{-x}.

[11]Actually, in the retimed implementation (see Figure 2) we compute the estimate of $rw[j+1] = r(rw[j] - D[j]p_{j+1} - C[j]p_{j+1}^2)$ by (1) assimilating $rw[j]$ up to bit of weight $2^{-(h-1)}$; (2) performing a truncated addition to produce an estimate with t fractional bits.

[12]The derivation of this condition is similar to that for division, as described for instance in [5].

[13]The exact solution of the inequality complicates the presentation, so that we used the bound $z \leq 1$. Moreover, we have verified that the same selection function is obtained using $z = 1/4$ (lower bound of z) and $z = 1$ (upper bound of z).

Table 3. Our design in context: latency and hardware complexity.

Latency (SP)		Latency (DP)	
Design		Design	
quadratic conv. [11]	15 cycles of fpmul. $15 \times (7.0(ttree) + 1.0(treg))\ t_{fa} = 120\ t_{fa}$ e.r. $\rightarrow$ Add 3 mults. (12 cycles) $27 \times (7.0(ttree) + 1.0(treg))\ t_{fa} \approx 215\ t_{fa}$	[11]	23 cycles of fpmul. $23 \times (7.0(ttree) + 1.0(treg))\ t_{fa} \approx 185\ t_{fa}$ e.r. $\rightarrow$ Add 3 mults (12 cycles) $35 \times (7.0\ (ttree) + 1.0(treg))\ t_{fa} \approx 280\ t_{fa}$
Linear Interp. [4]	$tT(12) + tmac(15 \times 15 + 26)$ $2.5 + 10.5 = 13\ t_{fa}$ e.r. $\rightarrow$ Add $3 \times (tmac(25 \times 25 + 50) + treg)$ $13\ t_{fa} + 3 \times (11.0 + 1.0)\ t_{fa} \approx 50\ t_{fa}$	ref. [6]	$tT(14) + tm(15 \times 56) + 2 \times tm(14 \times 14) +$ $ta(3-2) + ta(4-2) + tm(43 \times 44)$ $3.0 + 9.0 + 2 \times 4.5 + 1.0 + 1.5 + 12.0 \approx 35\ t_{fa}$ e.r. $\rightarrow$ Add $3 \times (tmac(54 \times 54 + 108) + \text{treg})$ $35\ t_{fa} + 3 \times (12.0 + 1.0)\ t_{fa} \approx 75\ t_{fa}$
Our	$12 \times (tmux + ta(3-2) + tsel + treg)$ $12 \times (0.5 + 1.0 + 3.0 + 1.0)\ t_{fa} \approx 65\ t_{fa}$ e.r. $\rightarrow$ Add one iter. $13 \times (0.5 + 1.0 + 3.0 + 1.0)\ t_{fa} \approx 70\ t_{fa}$		$26 \times (tmux + ta(3-2) + tsel + treg)$ $26 \times (0.5 + 1.0 + 3.0 + 1.0)\ t_{fa} \approx 145\ t_{fa}$ e.r. $\rightarrow$ Add two iter. $28 \times (0.5 + 1.0 + 3.0 + 1.0)\ t_{fa} \approx 155\ t_{fa}$

Keys: $tT(a)$: delay of a table of a inputs; $tm(a \times b)$: delay of multiplier of $a \times b$ bits.
$ta(3-2), ta(4-2)$: delay of a 3-2, 4-2 csa; *fpmul*: floating–point mult. of four stages.
$tmac(a \times b + c)$: delay of a multiplier–accumulator with a multiplier of a bits, multiplicand of b bits and and accumulation of c bits.
tsel: delay of a short cpa and selection function of 11 input bits. *treg*: delay of a register.
e.r.: Exactly Rounded results.

Hardware (SP)		Hardware (DP)	
Design		Design	
quadratic conv. [11]	10 % of a 76×76 fp–mult. e.r. $\rightarrow$ Add mux, reg. and change mult. for mac	[11]	10 % of a 76×76 fp–mult. e.r. $\rightarrow$ Same as SP
Linear Interp.[4]	Tables: $2^{12} \times 26$, $2^{12} \times 15$ mac: $15 \times 15 + 26$ e.r. $\rightarrow$ Change mult. for mac: $25 \times 25 + 50$ 2 reg., 3 mux	ref. [6]	Tables: $2^{14} \times 15$, $2^{14} \times 56$ mult: 56×15, 14×14, 44×43 csa: 2 $(4-2)$, 7 $(3-2)$ e.r. $\rightarrow$ Change mult. for mac: $54 \times 54 + 108$ 2 reg., 3 mux
Our	Table: $2^{11} \times 4$ datapath($\sim$ 24 bit):3 4–1mux, 4–2csa, cpa, 4 reg. e.r. $\rightarrow$ double width (48 bits)		Table: $2^{11} \times 4$ datapath($\sim$ 53 bit):3 4–1mux, 4–2csa, cpa, 4 reg. e.r. $\rightarrow$ double width (105 bits)

Table 4. Conditions for selection constants.

$k \geq 1$	$2^{-t}\left\lceil 2^t((D_U(i)(k-\frac{2}{3}) + \frac{1}{8}4^{-j}(D_U(i)+\frac{2}{3}4^{-j})^2(k-\frac{2}{3})^2)\right\rceil \leq m_k(i) \leq 2^{-t}\left\lfloor 2^t(D_L(i)(k-\frac{1}{3}) - 2^{-t})\right\rfloor$
$k \leq 0$	$2^{-t}\left\lceil 2^t(D_L(i)(k-\frac{2}{3}) + \frac{1}{8}4^{-j}(D_L(i)+\frac{2}{3}4^{-j})^2(k-\frac{2}{3})^2)\right\rceil \leq m_k(i) \leq 2^{-t}\left\lfloor 2^t((D_U(i))(k-\frac{1}{3}) - 2^{-t})\right\rfloor$

Necessary condition: $\frac{1}{6} - \frac{25}{72}4^{-j}(\frac{1}{2}+\frac{2}{3}4^{-j})^2 - \frac{4}{3}(2^{-s}+2^{-x}) - 2^{-t} > 0$

$$D[j] \geq \sqrt{z} - (2/3)4^{-j}z > \sqrt{z} - (2/3)4^{-j}$$

Therefore $z < (D[j] + (2/3)4^{-j})^2$, and substituting in (18) we obtain

$$\max_i\left(D[j](k-\frac{2}{3}) + \frac{1}{8}4^{-j}(D[j]+\frac{2}{3}4^{-j})^2(k-\frac{2}{3})^2\right) \leq m_k(i) \leq \min_i\left(D[j](k-\frac{1}{3})\right) - 2^{-t} \quad (19)$$

Because of the max and min functions, we now consider separately the cases $k \geq 1$ and $k \leq 0$. Moreover, since the selection constants are integer multiples of 2^{-t}, we limit the interval to integer multiples of 2^{-t} and give the intervals in 2^{-t} units. The resultant expressions are shown in Table 4. The necessary condition was obtained from the worst case: $k = -1$ and $D_L(i) = 1/2$.

Determination of possible parameters

The parameters are obtained from the necessary condition (see Table 4). We see that there is no solution for all j, so that apparently it is necessary to have different selection functions for different j. We also observe that there is a solution for $j \geq 2$. Consequently, we consider first the case $j \geq 2$ and then the cases $j = 0$ and $j = 1$.

For $j \geq 2$ the condition is satisfied with $s = 4$, $t = 4$ and $x = 7$. However, for these values there are no selection constants with granularity 2^{-t} [14]. Consequently, we choose $s = 5$, $t = 4$ and $x = 8$ [15]. Applying the conditions for the selection function we obtain the ranges of Table 5.

5.2.1 Selection for $j = 0$ and $j = 1$

As indicated in the previous section, apparently the same selection function for $j \geq 2$ cannot be used for $j \leq 1$. However, we now show that it is actually possible to use the same selection function for all j. This is based on the fact that for $j \leq 1$ only a reduced set of pairs of values $(w[j],\ D[j])$ are possible. We consider separately the cases $j = 0$ and $j = 1$.

Case $j = 0$. The initial value of w is (with $D[0] = zP[0]$)

$$w[0] = \begin{cases} 2^{-1}(1 - D[0]) & \text{if } P[0] = 1 \text{ (exponent even)} \\ 2^{-1}(1 - 2D[0]) & \text{if } P[0] = 2 \text{ (exponent odd)} \end{cases}$$

Consequently, the possible set of pairs of values of $(w[0],\ D[0])$ are described by the two lines of Figure 3(a). For $j = 0$ the intervals of $D[0]$ do not overlap because $D[0]$ is not in redundant representation.

Moreover, the selection intervals for $j = 0$ are[16]

$$U_k[0] = (k + 2/3)D[0] + 2^{-3}(k + 2/3)^2 D[0]/P[0]$$

$$L_k[0] = (k - 2/3)D[0] + 2^{-3}(k - 2/3)^2 D[0]/P[0]$$

These intervals are also shown in the figure. Note that, although it seems that there should exist two regions for each interval (for the two values of $P[0]$), this is not the case because the allowed region of $(w[0],\ D[0])$ is always positive (negative) for $P[0] = 1$ ($P[0] = 2$).

Because of the limitation in the possible pairs of values of $(4w[j], D[j])$, the acceptable interval for a selection constant might be extended. Specifically, the standard interval $\max_i(L_k) \leq m_k(i) \leq \min_i(U_{k-1})$ is extended to the adjacent (continuous) regions in which there is no possible value of $4w[j]$ in the interval i of $D[j]$ (see Figure 3(b)). Using this fact we obtain the acceptable intervals of m_k given in Table 6.

Case $j = 1$. We have

$$w[1] = 4w[0] - p_1 D[0] - 2^{-3}p_1^2 D[0]/P[0]$$

Moreover,

$$w[0] = 2^{-1}(1 - D[0]P[0])$$

[14] We verified that for these values of s and t a large value of j is required. Consequently, this is not a good solution.

[15] We minimize the possible value of t to minimize the delay.

[16] We give the expressions in terms of $D[0]/P[0]$ instead of in terms of d since the selection is performed using $D[0]$

Table 5. Ranges of m_k for $j \geq 2$ ($D_L[i]$ scaled by 32 and m_k scaled by 16).

$D_L[i]$	16	17	18	19	20	21	22	23
m_{-1}	-13	-14	-14	-15	-16	-17,-16	-18,-17	-18
m_0	-5,-4	-5	-5	-6,-5	-6,-5	-6,-5	-7 to -5	-7,-6
m_1	3,4	4	4,5	4,5	4,5	4,6	4,6	5,6
m_2	12	13	13,14	14	15	15,16	16,17	17,18

$D_L[i]$	24	25	26	27	28	29	30	31
m_{-1}	-19,-18	-20,-19	-21,-20	-22 to -20	-23 to -21	-23, -22	-24 to -22	-25 to -23
m_0	-7,-6	-8,-6	-8,-6	-8,-6	-9,-6	-9,-7	-9,-7	-10,-7
m_1	5,7	5,7	5,7	5,8	5,8	6,8	6,9	6,9
m_2	17,19	18,19	19,20	19,21	20,22	21,23	21,24	22,24

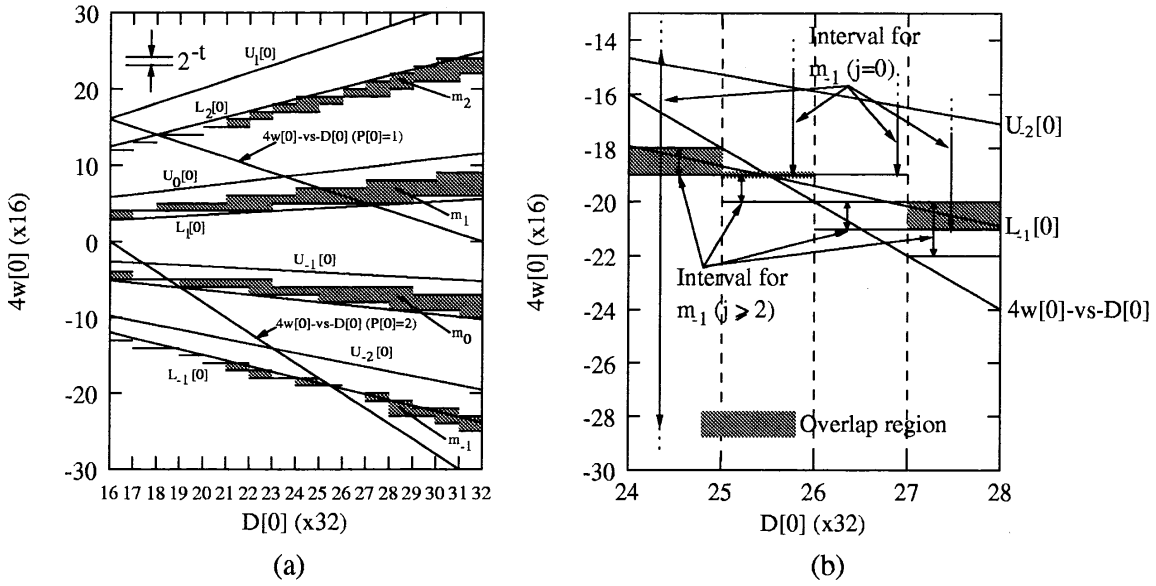

Figure 3. $4w[0]$ **versus** $D[0]$ **(Dashed areas: intersection intervals between the cases** $j = 0$ **and** $j \geq 2$**).**

$$D[1] = D[0] + 2^{-2}p_1 D[0]/P[0]$$

resulting in

$$w[1] = 2 - \frac{2P[0]^2 + p_1 P[0] + 2^{-3}p_1^2}{P[0] + 2^{-2}p_1} D[1]$$

Consequently, the possible values of the pair ($w[1]$, $D[1]$) correspond to linear segments for the values of $P[0]$ and p_1. As shown for the case $j = 0$, the possible pairs are ($P[0] = 1$, $p_1 = 0,\ 1,\ 2$) and ($P[0] = 2$, $p_1 = 0,\ -1,\ -2$), resulting in six line segments[17]. Note that because $D[1]$ is in redundant representation (carry–save) the intervals of $D[1]$ overlap.

The selection intervals are

$$U_k[1] = (k+2/3)D[1] + 2^{-5}(k+2/3)^2 D[1]/(P[0]+2^{-2}p_1)$$

$$L_k[1] = (k-2/3)D[1] + 2^{-5}(k-2/3)^2 D[1]/(P[0]+2^{-2}p_1)$$

Note that the selection interval (as a function of $D[1]$) depends on the pair ($P[0]$, p_1). Using the same conditions as for $j = 0$ we show in Table 7 the possible range of values of m_k for $j = 1$.

5.2.2 Selection function for all j

The selection for all j is obtained from the intersection of the ranges for $j \geq 2$, $j = 0$ and $j = 1$. This results in the Table 8 As is shown in the table, the only

[17] Although the range of $D[0]$ for each line segment is limited (because the choice of $p_1 = k$ is limited to a range of $D[0]$), to simplify the analysis we have considered the whole range of $D[0]$ to determine each line segment

Table 6. Ranges of m_k for $j = 0$ ("-" means don't care, $D_L[i]$ scaled by 32 and m_k scaled by 16).

$D_L[i]$	16	17	18	19	20	21	22	23
m_{-1}	$\leq$ -2	$\leq$ -4	$\leq$ -6	$\leq$ -8	$\leq$ -10	$\leq$ -12	$\leq$-14	$\leq$-15
m_0	≤ -2	≤ -3	-	≥ -6	≥ -8	≥ -10	≥ -12	≥ -14
m_1	≤ 15	≤ 14	≤ 13	≤ 12	≤ 11	≤ 10	≤ 9	≤ 8
m_2	-	-	≥ 14	≥ 13	≥ 12	≥ 11	≥ 10	≥ 9

$D_L[i]$	24	25	26	27	28	29	30	31
m_{-1}	-	$\geq$ -19	$\geq$ -19	$\geq$ -21	$\geq$ -24	$\geq$ -26	$\geq$-28	$\geq$-30
m_0	≥ -16	≥ -18	≥ -20	≥ -22	≥ -24	≥ -26	≥ -28	≥ -30
m_1	-	-	-	≥ 5	≥ 4	≥ 3	≥ 2	≥ 1
m_2	≥ 8	≥ 7	≥ 6	≥ 5	≥ 4	≥ 3	≥ 2	≥ 1

Table 7. Ranges of m_k for $j = 1$ ($D_L[i]$ scaled by 32 and m_k scaled by 16).

$D_L[i]$	16	17	18	19	20	21	22	23
m_{-1}	$\leq$ -9	-14 to -12	-16 to -6	-24 to -13	-16 to -14	-19 to -5	-26 to -11	-33 to -17
m_0	-5,-4	-8 to 1	-16 to -4	-6 to 7	-12 to 1	-19 to -5	-7 to -5	-10 to 7
m_1	0 to 8	3 to 5	2 to 13	-5 to 7	4 to 6	2 to 17	-4 to 12	-10 to 7
m_2	12,13	9 to 19	2 to 15	14 to 27	8 to 22	2 to 18	16 to 35	13 to 31

$D_L[i]$	24	25	26	27	28	29	30	31
m_{-1}	-19 to -17	-22 to -3	-28 to -8	-34 to -13	-40 to -18	-46 to -21	-24 to 3	-27 to -1
m_0	-16 to 2	-22 to -3	-28 to -6	-8 to 15	-12 to 11	-17 to 7	-22 to 3	-27 to -1
m_1	5 to 7	3 to 23	-2 to 19	-7 to 15	-12 to 11	-17 to 9	≥ 6	≥ 4
m_2	8 to 27	3 to 23	-2 to 22	≥ 19	≥ 16	≥ 12	≥ 8	≥ 4

case in which the intersection is empty (no overlap) is for $D_L[i] = 26/32$ and m_{-1}. This is due to the point $(26/32, -20/16)$ in the graph $D[0] - vs - 4w[0]$, which corresponds to the case $j = 0$ and even exponent. To solve the problem, we note that the selection would be correct with the selection constants of the row corresponding to $D_L[i] = 24/32$, for even (negative part of the graph) or odd (positive part of the graph) exponent. Therefore when $\hat{D} = 26/32$ and $j = 0$, we transform $\hat{D}$ into 24/32 (setting to zero just the third fractional bit) before input to the selection function.

5.2.3 Range of $D[j]$

As mentioned before, the selection function might produce a $D[j+1]$ which is out of the range $[0.5, 1)$. Since this is inconvenient for the implementation we determine in [10] the requirements on the selection constants so that this out-of-range situation does not occur. We then prune the selection function we have obtained so that it satisfies these requirements. The resulting selection function is given in Table 9. The ranges which were reduced by the pruning are indicated by *. However, since $\hat{D}$ is an estimation of $D[j]$, with a negative error of $2^{-5} + 2^{-8}$, we may obtain $\hat{D} = 15/32 < 1/2$, as input to the selection function. To simplify the implementation of the selection function it is convenient to have the bit with weight 2^{-1} always set to one. Therefore, since a value of $\hat{D} = 15/32$ corresponds to the bound $1/2 \leq D[j] < 1/2 + 2^{-5} + 2^{-8}$, we saturate $\hat{D}$ to 16/32 in this case, before input to the selection function, allowing a correct selection. This scheme is performed by detecting a zero in the weight 2^{-1} of $\hat{D}$, and inverting all the bits in that case.

6. Conclusions

We have presented a reciprocal square–root algorithm by digit recurrence and selection by a staircase function. The algorithm permits to obtain exactly rounded results in a straightforward manner, in contrast to existing methods. We described in detail the radix–4 implementation. Specifically, we developed a detailed analysis to show that a single selection function can be used for all iterations. We also analyze the latency hardware complexity trade–off, by comparing with existing alternatives. We concluded that our implementation represents a low hardware alternative with moderate latency. Moreover, the latency can be reduced if radix–8 or overlapped radix–4 (radix–16) schemes are developed following the method used for the radix–4 implementation.

References

[1] E. Antelo, T. Lang and J.D. Bruguera, "Computation of $\sqrt{x/d}$ in a very high radix combined division/square–root

Table 8. Intersection ranges of m_k for all iterations. ($D_L[i]$ scaled by 32 and m_k scaled by 16).

$D_L[i]$	16	17	18	19	20	21	22	23
m_{-1}	-13	-14	-14	-15	-16	-17,-16	-18,-17	-18
m_0	-5,-4	-5	-5	-6,-5	-6,-5	-6,-5	-7 to -5	-7,-6
m_1	3,4	4	4,5	4,5	4,5	4 to 6	4 to 6	5,6
m_2	12	13	14	14	15	15,16	16,17	17,18

$D_L[i]$	24	25	26	27	28	29	30	31
m_{-1}	-19,-18	-19	‡ no overlap	-21, -20	-23 to -21	-23, -22	-24 to -22	-25 to -23
m_0	-7,-6	-8 to -6	-8 to -6	-8 to -6	-9 to -6	-9 to -7	-9 to -7	-10 to-7
m_1	5 to 7	5 to 7	5 to 7	5 to 8	5 to 8	6 to 8	6 to 9	6 to 9
m_2	17 to 19	18,19	19,20	19 to 21	20 to 22	21 to 23	21 to 24	22 to 24

‡ For $j = 0$ use the constants of the row $D_L[i] = 24/32$. For $j \geq 1$ use the constants -21 or -20.

Table 9. Ranges of m_k for all iterations with $1/2 \leq D[j] < 1$ ($D_L[i]$ scaled by 32 and m_k scaled by 16).

$D_L[i]$	16	17	18	19	20	21	22	23
m_{-1}	-13	-14	-14	-15	-16	-17,-16	-18,-17	-18
m_0	* -5	-5	-5	-6,-5	-6,-5	-6,-5	-7 to -5	-7,-6
m_1	3,4	4	4,5	4,5	4,5	4 to 6	4 to 6	5,6
m_2	12	13	14	14	15	15,16	16,17	17,18

$D_L[i]$	24	25	26	27	28	29	30	31
m_{-1}	-19,-18	-19	‡ no overlap	-21, -20	-23 to -21	-23, -22	-24 to -22	-25 to -23
m_0	-7,-6	-8 to -6	-8 to -6	-8 to -6	-9 to -6	-9 to -7	-9 to -7	-10 to -7
m_1	5 to 7	* 7	* 7	* 6 to 8	5 to 8	6 to 8	* 8,9	* 8,9
m_2	17 to 19	18,19	19,20	19 to 21	20 to 22	21 to 23	21 to 24	22 to 24

‡ For $j = 0$ use the constants of the row $D_L[i] = 24/32$. For $j \geq 1$ use the constants -21 or -20.

unit with scaling", IEEE Transactions on Computers, vol 47, no .2, pp. 152–161, 1998.

[2] R.C. Agarwal, F.G. Gustavson, M.S. Schmookler, "Series approximations methods for divide and square root in the Power3 processor", in Proc. 14th IEEE Symposium on Computer Arithmetic, pp. 116–123, 1999.

[3] D.Das Sarma and D.w. Matula, "Faithfull bipartite ROM reciprocal tables", in Proc. 12th IEEE Symposium on Computer Arithmetic, pp. 17–28, 1995.

[4] D.Das Sarma and D.w. Matula, "Faithfull interpolation in reciprocal tables", in Proc. 13th IEEE Symposium on Computer Arithmetic, pp. 82–91, 1997.

[5] M.D. Ercegovac and T. Lang, "Division and square root: digit–recurrence algorithms and implementations", Kluwer Academic Pub., 1994.

[6] M.D. Ercegovac, T. Lang, J.–M. Muller and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers", IEEE Transactions on Computers, vol. 49, no. 7, pp. 628–637, 2000.

[7] H. Hassler and N. Takagi, "Function evaluation by table look–up and addition", in Proc. 12th IEEE Symposium on Computer Arithmetic, pp. 10–16, 1995.

[8] V.K. Jain, L. Lin, "High–speed double precision computation of nonlinear functions", in Proc. 12th IEEE Symposium on Computer Arithmetic, pp. 107–114, 1995.

[9] A. Nannarelli and T. Lang, "Low–power divider", IEEE Transactions on Computers, vol. 48, no. 1, pp. 2–14, 1999.

[10] T. Lang and E. Antelo, "Correctly rounded reciprocal square–root by digit recurrence", Internal Report. Dept. Electrical and Computer Eng., University of California at Irvine. Link: http://www.ece.uci.edu/numlab.

[11] S. Oberman, "Floating point division and square root algorithms and implementations in the AMD–K7 microprocessor", in Proc. 14th IEEE Symposium on Computer Arithmetic, pp. 106–115, 1999.

[12] J.E. Robertson, "A new class of digital division methods", IRE Transactions on Electronic Computers, vol. EC–7, pp. 218–222, 1958.

[13] M.J. Schulte and J.E. Stine, "Approximating elementary functions with symmetric bipartite tables", IEEE Trans. on Computers, vol 48, no 8, pp. 842–847, 1999.

[14] M.J. Schulte and K.E. Wires, "High–speed inverse square roots", in Proc. 14th IEEE Symposium on Computer Arithmetic, pp. 124–131, 1999.

[15] N. Takagi, "Generating a power of an operand by a table look–up and a multiplication", in Proc. 13th IEEE Symposium on Computer Arithmetic, pp. 126–131, 1997.

[16] W.F. Wong and E. Goto, "Fast hardware–based algorithms for elementary function computations using rectangular multipliers", IEEE Transactions on Computers, Vol. 43, no. 3, March 1994.

[17] W.F. Wong and E. Goto, "Fast evaluation of the elementary functions in single precision", IEEE Transactions on Computers, vol 44, no. 3, pp. 453–457, 1995.