Very-High Radix Combined Division and Square Root with Prescaling and Selection by Rounding

Tomás Lang
Dep. of Electrical and Computer Eng.
University of California at Irvine
e-mail: tlang@uci.edu

Paolo Montuschi
Dip. di Automatica e Informatica
Politecnico di Torino
e-mail: montuschi@polito.it

Abstract

An algorithm for square root with prescaling is developed and combined with a similar scheme for division. An implementation is described, evaluated and compared with other combined div/sqrt implementations.

1 Introduction

The implementation of very-high radix division and square root is difficult because of the complexity of the result-digit selection function. For division, a technique that has been extensively studied to reduce this complexity is the prescaling of the operands [3, 4, 11, 12, 21, 22]. In [7] it has been demonstrated that this technique can be used to achieve a faster implementation than other known dividers. Related schemes are presented in [1, 19, 24].

For square root, an algorithm with operand prescaling was presented in [13]. However, the resulting implementation is complex and is not compatible with the corresponding division unit. In this paper we present another algorithm for square root which can be easily combined with the division implementation of [7].

We then describe a combined implementation. Under the assumption that division is more frequent than square root, our main concern was not to increase the time of division. In fact, by using the additional multiplier required for square root also for the scaling step of division, we achieve a slight speed improvement. On the other hand, since the execution of the recurrence for square-root in one cycle would increase the cycle time, we chose an implementation in which this recurrence is done in two cycles.

A rough evaluation of the proposed implementation is performed, in terms of execution time and area, using as the basic unit the delay and area of a full adder. Finally, we perform a comparison with another high radix unit, with one unit that performs division and square-root by successive approximations, and with two units using lower-radix recurrences.

The combination of division and square root in a single unit has been described in [5, 19, 16, 17, 18, 20, 9, 10, 25]. Moreover, units that use algorithms with successive approximation also can perform both division and square root.

2 Square root with prescaling

In division, the prescaling consists of multiplying the divisor and the dividend by a factor so that the scaled divisor is close to 1. The corresponding approach to square root is to prescale the operand so that the scaled result is close to 1. However, as shown in [13] this requires a complicated postscaling of the result, and is not compatible with division. Here we develop a different algorithm, which consists of performing a modified scaled recurrence.

The standard recurrence is (already multiplied by 2^{-1} to make it compatible with division [6]),

$$w[j+1] = rw[j] - S[j]s_{j+1} - 2^{-1}s_{j+1}^2 r^{-(j+1)}$$

$$w[0] = 2^{-1}x$$

where $w[j]$ is the residual, $r = 2^b$ is the radix, $S[j] = \Sigma_{i=1}^{j} s_i r^{-i}$ is the partial result after step j, and x is the operand. The result-digit selection function

$$s_{j+1} = sel(rw[j], S[j])$$

has to satisfy the residual bound (for max. redundant digit set)

$$-S[j] + 2^{-1}r^{-j} < w[j] < S[j] + 2^{-1}r^{-j}$$

Because of the term on s_{j+1}^2 in the recurrence, the selection function for the initial digit is different than for the rest. That is, the most significant k bits of the square root are obtained as

$$S[1] = 2^{-k}s_1 = F(x)$$

Then, since $S[0] = 0$, the first iteration produces

$$w[1] = 2^k w[0] - 2^{-1}s_1^2 2^{-k}$$

and the rest of the iterations are

$$w[j+1] = rw[j] - S[j]s_{j+1} - 2^{-1}s_{j+1}^2 r^{-J}$$

where $J = j + g$ and $r^g = 2^k$.

As stated in the introduction, for high radix ($r > 8$), the selection function is complex and a direct implementation is not practical.

1063-6889/95 $4.00 © 1995 IEEE

2.1 Scaled recurrence

To reduce the complexity of the selection function, we perform a scaling of the residual so that

$$new(w[j]) = Mw[j] \tag{1}$$

resulting in the scaled recurrence (calling $w[j]$ the new residual)

$$w[j+1] = rw[j] - T[j]s_{j+1} - 2^{-1}t_{j+1}s_{j+1}r^{-J} \tag{2}$$

and the first iteration

$$w[1] = 2^{k-1}Mx - 2^{-1}t_1 s_1 2^{-k} \tag{3}$$

where $T[j] = MS[j]$ and $t_j = Ms_j$. The bound is

$$\begin{aligned} L1 = \quad & -T[j] + 2^{-1}Mr^{-J} < w[j+1] \\ < \quad & T[j] + 2^{-1}Mr^{-J} = U1 \end{aligned} \tag{4}$$

The factor M is chosen so that $T[j]$ is close enough to 1 and the result digit can be obtained by rounding an estimate of the shifted residual. That is,

$$T[j] \approx 1 \ \ and \ \ s_{j+1} = \lfloor \widehat{y} + 2^{-1} \rfloor \tag{5}$$

where $\widehat{y}$ is obtained by truncating the redundant (carry-save or signed-digit) representation of rw.

The scaled iteration is performed as follows:

$$s_{j+1} = \lfloor \widehat{y} + 2^{-1} \rfloor$$

$$t_{j+1} = Ms_{j+1}$$

$$w[j+1] = rw[j] - T[j]s_{j+1} - 2^{-1}t_{j+1}s_{j+1}r^{-J}$$

$$T[j+1] = T[j] + t_{j+1}r^{-J}$$

With respect to the initial value $S[1]$, instead of obtaining it by a table look-up, we use the fact that

$$T = MS = M\sqrt{x} \approx 1$$

Consequently, Mx is an approximation of $\sqrt{x}$ so that

$$S[1] = 2^{-k}s_1 \approx Mx$$

To utilize the same hardware used for the selection of the other digits, we make

$$s_1 = \lfloor \widehat{2^k Mx} + 2^{-1} \rfloor \tag{6}$$

where $\widehat{2^k Mx}$ is obtained by truncating the redundant representation of $2^k Mx$.

2.2 Scaling interval and bits of $S[1]$

To complete the algorithm, we now determine the range of $M\sqrt{x}$ and the number of bits of $S[1]$ so that the algorithm converges when

A. s_1 is obtained by (6)

B. for $j \geq 2$, s_j is obtained by (5) and $|s_j| \leq r-1$.

Consequently, the following two bounds have to be satisfied:

1. The convergence bounds given by (4).
2. For ($j \geq 1$),

$$L2 = -1 + \frac{1}{2r} < |w[j]| < 1 - \frac{1}{2r} = U2 \tag{7}$$

To determine the scaling interval we follow a similar approach as in [7]. When the selection of s_{j+1} is done by rounding as described by (5) and (6), the next residual is bounded by

$$\begin{aligned} L = \quad & -2^{-1} + (1 - T[j])s_{j+1} - 2^{-1}Ms^2 r^{-J} \leq w[j+1] \\ \leq \quad & 2^{-1} + 2^{-t} + (1 - T[j])s_{j+1} - 2^{-1}Ms^2 r^{-J} = U \end{aligned}$$

where the term 2^{-t} is due to the truncation of the carry-save representation of $w[j]$ to t fractional bits.

Consequently, M is selected so that

$$(L > L1 \ \ and \ \ L > L2) \ \ and \ \ (U < U1 \ \ and \ \ U < U2)$$

The solution of these inequalities results in the required range of $T[j]$. However, $T[j]$ is not known to obtain the corresponding value of M. To base the calculation of M on x, we use the following relation

$$M\sqrt{x} - r^{-J} \leq T[j] \leq M\sqrt{x} + r^{-J} \tag{8}$$

The detailed computations are reported in [15]. We obtain the following results:

$t = 2$ (same as for division)

$k = b + 3$ and $s_1 \leq 8r$

$|s_j| \leq (r-1)$ for $j \geq 2$

$$1 - \left[\frac{1}{4} - \frac{23}{32r}\right]\frac{1}{8r\sqrt{x}} \leq M\sqrt{x} \leq 1 + \left[\frac{1}{2} - \frac{7}{8r}\right]\frac{1}{8r\sqrt{x}} \tag{9}$$

2.3 Computation of M

We now describe the computation of M so that the range of (9) is satisfied. As we did in [7] for division[1], M is computed by the following steps:

[1] The implementation has been somewhat improved in [14], we use here this improved version.

1. Obtain the pair of coefficients A_a and C_c (with a and c fractional bits, respectively) as a function of x_τ, the operand truncated to τ fractional bits.

2. Compute the carry-save representation of

$$P = C_c - A_a\delta_h$$

where $\delta_h = x_h - x_\tau$, with x_h being the operand truncated to h fractional bits.

3. Obtain the scaling factor M by rounding $\widehat{2^m P}$, that is

$$M = round(\widehat{2^m P}) \cdot 2^{-m},$$

where $\widehat{2^m P}$ is the truncation of the carry-save representation of $(2^m P)$ to the second fractional bit.

Derivation of the expressions for A and C as well as for the number of bits required for each variable are reported in [15]. The results are

•

$$C = \frac{216(4x_\tau - I)^2 x_\tau^{3/2}(8x_\tau^2 + 4x_\tau I - I^2)}{27648x_\tau^6 - 7344x_\tau^4 I^2 + 1620x_\tau^3 I^3 + 36x_\tau^2 I^4 - 27x_\tau I^5 + I^6} \quad (10)$$

$$A = \frac{216(4x_\tau - I)^3 x_\tau^{3/2}}{27648x_\tau^6 - 7344x_\tau^4 I^2 + 1620x_\tau^3 I^3 + 36x_\tau^2 I^4 - 27x_\tau I^5 + I^6} \quad (11)$$

where $I = 2^{-\tau}$;

- x_τ corresponds to the $\tau = (\lceil b/2 \rceil + 2)$ most-significant fractional bits of x;
- x_h corresponds to the next $(\lfloor b/2 \rfloor + 4)$ bits of x;
- A requires 2 integer bits and $\lfloor b/2 \rfloor + 3$ fractional bits;
- C requires 1 integer bit and $b + 8$ fractional bits; (however, since it can be proved from (10) that $C \geq 1$, it is not necessary to store the integer bit in the TABS module);
- M requires 1 integer bit plus $b + 5$ fractional bits.

3 Implementation issues

We now consider the implementation of the iteration so that its delay is similar to that of division. We will see that some modifications of the algorithm presented in the previous section are required.

A direct implementation of the iteration as presented in the previous section has the following critical path:

1. selection function and recoding of s_{j+1}
2. computation of $t_{j+1} = Ms_{j+1}$
3. computation of $2^{-1}t_{j+1}s_{j+1}r^{-J}$
4. addition to produce $w[j+1]$

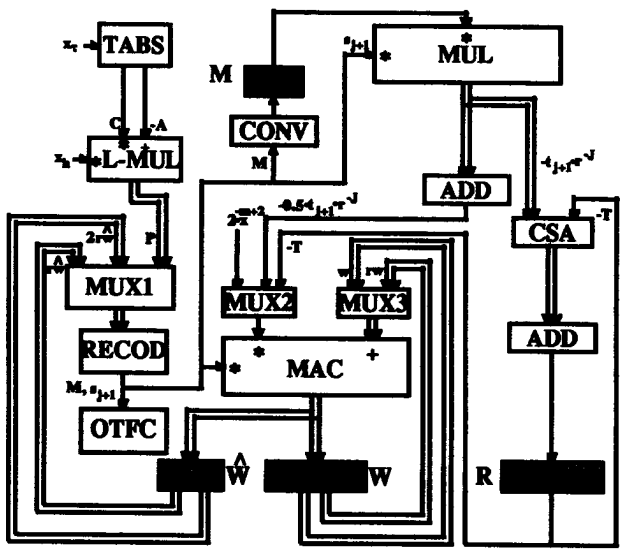

Figure 1: Architecture of the unit for square root

This corresponds to two multiplications (delay of each corresponding to the number of bits of the multiplier s_{j+1}) and one redundant addition. Since in division there is only one multiplication, the delay of the iteration for square root would be significantly larger. To reduce this delay, the solution we adopt is to perform each iteration of square root in two cycles (and one cycle for division). The two cycles operate as follows:

Cycle A

- selection function and recoding of s_{j+1}
- computation of $v[j+1] = rw[j] - T[j]s_{j+1}$
- first part of computation of $t_{j+1} = Ms_{j+1}$

Cycle B

- second part of computation of t_{j+1} (in assimilated form)
- computation of

$$w[j+1] = v[j+1] - 2^{-1}t_{j+1}s_{j+1}r^{-J}$$

- computation of $T[j+1] = T[j] + t_{j+1}r^{-J}$ in assimilated form

Moreover, initial cycles are required to:

Obtain M

Recode M and obtain $w[0] = 4Mx$, which is used to produce $w[1] = 8r(Mx/2) - 2^{-1}s_1t_1r^{-1})$

Assimilate M and set $T[0] = 0$

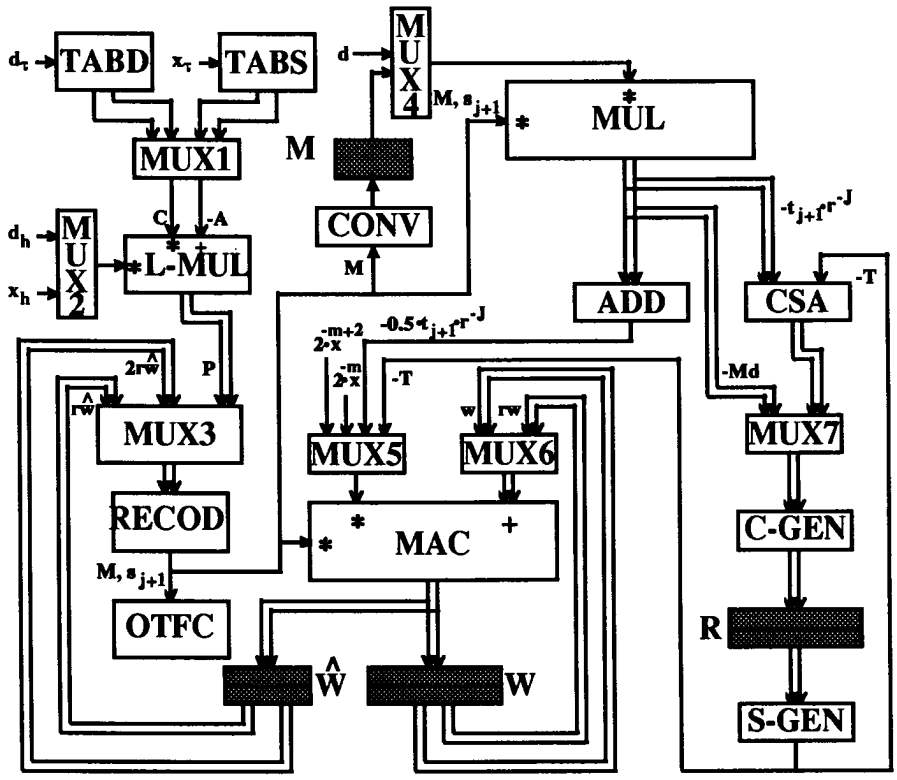

Figure 2: Architecture of the unit for combined division and square root

As we see in Section 4.2, this requires two cycles. Finally, one cycle is required for the detection of the sign of the last residual and the rounding.

The corresponding implementation is shown in Fig. 1. Note that to adapt to the signs of s_{j+1} and M, module MUL produces $-t_{j+1}$ and register R stores $-T$. Moreover, since s_{j+1} is an integer and M is a fraction, it is necessary to input $2^{-m+2}x$ to MAC.

4 Combined Implementation

We now describe an implementation of a module that performs both division and square root. This requires only minor modifications to the implementation for square root described in the previous section since the implementation of division as described in [7] essentially uses a subset of the modules utilized for square root. The resulting implementation is shown in Fig. 2. We now indicate the way division is performed and emphasize the modifications/additions with respect to the implementation of Fig. 1:

- The calculation of M for division uses the same approach as for square root. However, as discussed in [15], the values A and C are different so another module called TABD is needed. For division M also requires 1 integer plus $b+5$ fractional bits, and we have:

  –

$$C = \frac{2(d_\tau + I)}{2d_\tau(d_\tau + I) + (I/2)^2} \qquad (12)$$

$$A = \frac{2}{2d_\tau(d_\tau + I) + (I/2)^2} \qquad (13)$$

  – d_τ corresponds to the $\tau = (\lceil b/2 \rceil + 1)$ most-significant fractional bits of d; (however, only $\tau - 1$ enter the TABD module since the first fractional is always a 1);

  – d_h corresponds to the next $(\lfloor b/2 \rfloor + 4)$ bits of d

  – A requires 2 integer bits and $b/2+3$ or $(b+1)/2+1$ fractional bits, when b is even or odd, respectively;

  – C requires 1 integer bit and $b+3$ fractional bits; (however, since it can be proved from (12) that $C \geq 1$, it is not necessary to store the integer bit in the TABD module);

  Moreover, multiplexers MUX1 and MUX2 select between the parameters for division and square root.

- The calculation of Md is performed using MUL. This requires the inclusion of MUX4. The calculation of Mx is done simultaneously using MAC, so that one additional input is required for MUX5.

- The scaled dividend is stored in carry-save form in register W as $w[0]$. On the other hand, Md has to be assimilated and stored in the same register as T. To be able to perform the calculation of M, the multiplication of Md, and the assimilation in

two cycles, we perform the assimilation in a CLA adder divided in two parts: the generation of the carries (in module C-GEN) and the generation of the sum (in module S-GEN). Only the delay of C-GEN is included in the second cycle, whereas the delay of S-GEN is overlapped with the quotient-digit recoding in the next cycle.

Although this use of MUL for Md reduces by one the number of cycles for division, it increases somewhat the area (with respect to the alternative of using MAC for both Md and Mx) because the width of MUL has to be increased from $b+6$ bits (size of M) to 53 bits (size of d).

- The recurrence of division is performed in one cycle using the recoder, the divisor stored in R and MAC. The residual is stored in W and $\hat{y}$ in $\widehat{W}$.

4.1 Hardware requirements

The hardware requirements for the proposed shared unit are:

- Modules TABD and TABS for providing the coefficients required from the L-approach. TABD has $(\lceil b/2 \rceil)$ inputs and $(b+b/2+8)$ outputs when b is even, and $(b+(b+1)/2+6)$ outputs when b is odd; TABS has $(\lceil b/2 \rceil + 2)$ inputs and $(b + \lfloor b/2 \rfloor + 13)$ outputs;
- Carry-save multiplier with accumulation on one line (L-MUL) to compute P. The operands have $(\lfloor b/2 \rfloor + 4)$ and $(\lfloor b/2 \rfloor + 5)$ bits.
- The module (RECOD) for recoding and rounding the carry-save (unsigned) representation of P $(b+8)$ bits, to the radix-4 signed digit representation of M of $(b+6)$ bits; the same hardware is used also for the generation of the recoded digits s_{j+1}.
- One carry-save multiplier (MAC) of $(b+7) \times max(n+b+5, \lceil (n-3)/b \rceil b + 10)$ bits, with one line of accumulation and carry-save result of $(n+2b+10)$ bits. This multiplier performs also an accumulation of two lines only when the multiplier is s_{j+1} (i.e. of $(b+5)$ or $(b+1)$ two's complement weights).
- One carry-save multiplier (MUL) of $(b+7) \times (n)$ bits.
- One converter (CONV) for converting M from signed-digit radix-4 into non-redundant binary form.
- One two-step-adder (C-GEN, and S-GEN) for assimilating the carry save representation of $-Md$ and $-T$ of $(n+b+7)$ bits.
- One adder (ADD) for assimilating the two's complement carry save representation of t_{j+1} of $(2b+10)$ bits;
- Hardware for on-the-fly-conversion and rounding (OTFC) and registers for storing intermediate and final results.

4.2 Evaluation

We now present a rough evaluation of the implementation in terms of execution time and area. The objective of this evaluation is to determine the effect of the radix and to allow rough comparisons to other combined implementations. In what follows the number of bits of the operands is n and $b = log_2\ r$.

4.2.1 Execution time estimation

To determine the execution time we need the number of cycles and the cycle time. As discussed before, the following table shows the number of cycles for division and square root:

	Division	Square root
Scaling	2	2
Iterations	$\lceil n/b \rceil$	$2\lceil (n-3)/b \rceil$
Post-corr. & round	1	1
Total	$3+\lceil n/b \rceil$	$3+2\lceil (n-3)/b \rceil$

The cycle time of the combined square root and division operation using the implementation of Fig. 2 can be derived as the maximum delay of the paths in the timing diagrams of Fig. 3. The actual cycle time depends on the delay of the components. As we did in [7] we now evaluate this time in full-adder delays (t_{FA})[2] modules making the following assumptions on the delay of the components:

- Carry-save multiplier of $N1$ by $N2$ bits with accumulation of $N3$ operands. The delay is composed of three terms: a driver, the generation of the partial products, and the addition of the partial products plus the accumulation operands. Consequently,

$$t_{MA,CS,N1,N2,N3} = t_{driver} + t_{genmul} + t_{add}$$

  We estimate $t_{driver} = 1t_{FA}$ and $t_{genmul} = 0.5t_{FA}$. Since we recode the multiplier to radix-4 the number of operands to add is $\lceil N1/2 \rceil + N3$; the addition is done in a tree of 4-2 carry-save and 3-2 carry-save adders, with delays of $1.5t_{FA}$ and $1t_{FA}$, respectively.
- Recoding, including rounding, is about $1.5\ t_{FA}$.
- Carry-assimilate adders (and converters) of $N1$ bits use a carry-lookahead scheme with delay of $(1.5+0.3\lceil \log(N1) \rceil)t_{FA}$
- Register loading is $1.5t_{FA}$ and multiplexers (2-to-1 and 4-to-1 with decoded controls) have delay $.5t_{FA}$.

The only components whose delay cannot be readily described in t_{FA}'s are the TAB modules to determine the A and C coefficients. We have explored several alternatives as we did in [7] and have determined their delay and area using the family of standard cells from the ES2-ECPD10 library [8]. The delay and area of the best alternatives are summarized in Table 1. In Fig. 3

[2] For modules that are not composed directly from full adders, we use estimates obtained from implementations with standard gates [8].

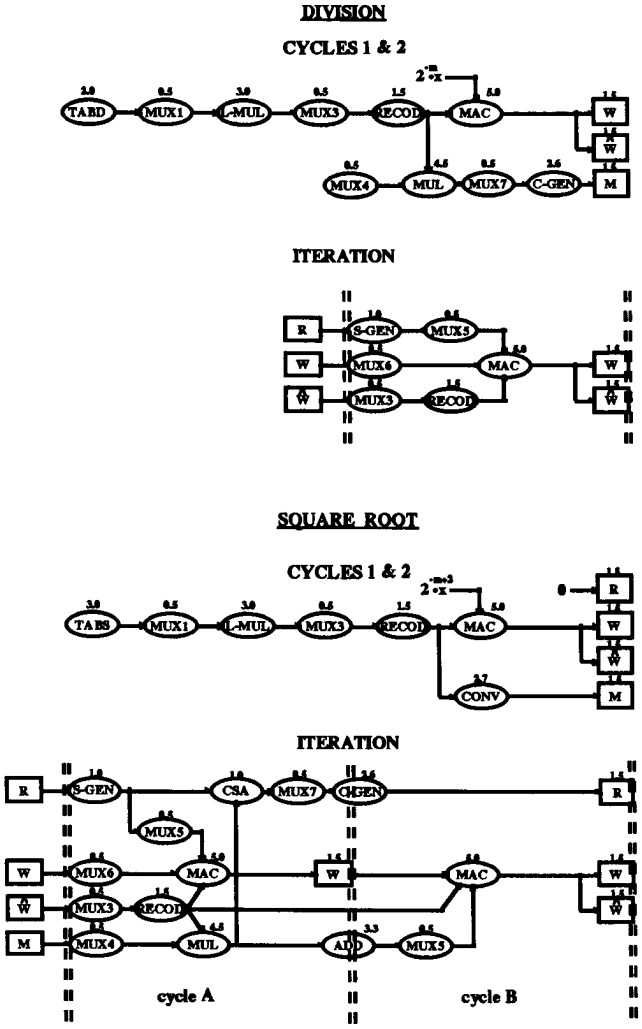

Figure 3: Timing diagrams for division and square root

we give the values of these delays for $r = 512$. Table 2 shows the cycle times for different values of b and the corresponding execution times for $n = 54$ bits.

4.2.2 Area estimation

As an estimate of the area we use the number of full adders in the multiplier modules and the equivalent area in full adders of the modules TABS and TABS. We use these modules since they should correspond to a large portion of the total area and similar modules are used in implementations of related algorithms so that comparisons are easy to perform. This area estimate is given in Table 2.

Table 1: Time and area for best TAB modules

b	9		11		14		18	
	D	S	D	S	D	S	D	S
t_{FA}	2	3	2	3	3	4	4	5
A_{FA}	60	310	140	670	340	1600	1650	7400

Table 2: Time and area for proposed implementation

b	9	11	14	18
Cycle time (t_{FA})	8.5	9.5	9.5	9.5
No. cycles DIV	9	8	7	6
Exec. time DIV (t_{FA})	75	75	65	55
No. cycles SQRT	15	13	11	9
Exec. time SQRT (t_{FA})	125	125	105	85
Area mult. (A_{FA})	1100	1300	1600	2000
Area TAB (A_{FA})	400	900	2000	9100
Area mult.+tab. (A_{FA})	1500	2200	3600	11100

5 Other Related Implementations

For comparison purposes, we now make a rough evaluation of the execution time of two other implementations for very-high radix units for shared division and square root. As a reference, we also include a radix-4 unit and a radix-8 unit with overlapping. To make uniform evaluations, we use as a unit the delay of a full adder, as done in Section 4.2. The evaluations are rough since data routing delays are not included nor do we perform some technology-dependent optimizations. As done in Section 4.2, for the very-high radix schemes we also give the area of the multipliers and initial tables, in full-adder units. We have normalized the implementations so that they use the same type of units, such as carry-save multipliers, recoders, and modules to generate scaling factors. More details of these comparisons are given in [15].

Scheme A: Unit using result-digit selection by residual scaling

In [1], [2] and [19] a radix-2^{17} division unit of this type is described, which is based on multiplying the residual by a short reciprocal of the divisor so that digit selection can be done by rounding. The unit uses a 18×69 rectangular multiplier with an additional adder port. This multiplier serves also as a 19×69 bit multiplier to perform the multiplication of the residual by the short reciprocal.

As mentioned, to compare with the method we are proposing we have adapted the implementation so that it uses the same features included in our scheme, when appropriate. In particular, we consider a radix-2^b 54-bit divider. Moreover, we utilize the same method for obtaining the short reciprocal (equivalent to the scaling factor in the method we propose). The number of cycles for division has been discussed in [7], and, for 54-bit result, is

$$N_{cycles,div} = 2 + 2\lceil 54/b \rceil$$

As indicated in [19], the computation of square root requires two more cycles than division. With respect to the cycle time, to the expression given in [7] for division it is necessary to include another carry-save adder because of the term depending on s^2_{j+1}, required for square root. This results in

$$t_A = t_{MPX,2,1} + t_{recod} + t_{MA,CS,b+9,n,0} + t_{6-2-csa} + t_{reg}$$

If the 6-2 adder is implemented by a cascade of one

Table 3: Time and area for scheme A

b	9	11	14	18
Cycle time (t_{FA})	11	12	12	12
Cycles DIV	14	12	10	8
Ex. time DIV (t_{FA})	155	145	120	95
Cycles SQRT	16	14	12	10
Ex. time SQRT (t_{FA})	175	170	145	120
Area (A_{FA})	1800	2500	4000	11500

level of 3-2 adders and one of 4-2 adders, the resulting delay is $t_{6-2-csa} = 2.5t_{FA}$.

With the assumptions of Section 4.2 we get the execution times and areas (for the multiplier plus the TABD and TABS modules) shown in Table 3.

Scheme B: Unit based on Newton-Raphson approximation

Such a scheme is used, for example in the Weitek W4164 and W4364 chips. The data sheet [23] indicates that the double-precision division takes seven cycles, which include the initial approximation of the reciprocal, the successive approximations, and the final rounding. On the other hand, the computation of double precision square root requires 8 cycles [23]. The cycle time is probably determined by a multiplication-accumulation (including recoding and register loading), so that we estimate the cycle delay to be

$$\begin{aligned} t_B &= t_{recod} + t_{MA,CA,64,64,1} + t_{reg} \\ &= (1 + 11.6 + 1.5)t_{FA} \approx 14t_{FA} \end{aligned}$$

where t_{recod} is the recoding of non-redundant multiplier (for this we assume a delay of $1t_{FA}$). Therefore, the duration of the whole computation is

$$\begin{aligned} T_{B,DIV} &= 7 \cdot 14t_{FA} \approx 100t_{FA} \\ T_{B,SQRT} &= 8 \cdot 14t_{FA} \approx 110t_{FA} \end{aligned}$$

We estimate that the area of the hardware for digit selection and updating of the residual is about the area of the multiplier, which has been estimated as $2500A_{FA}$.

Scheme C: Radix-4 unit

This unit is described in [6] and shares the same hardware for division and square root. As indicated in [6], the delay per iteration can be approximated by

$$t_C = t_{dsel} + t_{buffer} + t_{MUX} + t_{CSA} + t_{reg}$$

With the delays given in Section 4.2.1 and translating the delay given in [6] for the digit selection as $t_{dsel} = 2.5t_{FA}$ we get

$$t_C = 6t_{FA}$$

For a 54-bit mantissa, 27 cycles plus one for rounding are required so that the execution time is

$$T_C = 28t_C \approx 170t_{FA}$$

We do not estimate the area since in this case no multipliers are used and therefore it is not directly comparable with the estimate made for the very high-radix schemes.

Scheme D: Radix-8 with overlapping stages

In [9] a method is described for the design of a radix-8 shared division and square root unit, with overlapped radix-4 stage and radix-2 stage. As can be obtained from [9], the corresponding cycle time is

$$\begin{aligned} t_D &= t_{CSA1} + t_{DSM} + t_{MPX,2,1} + t_{driver} + \\ &\quad t_{MPX,5,1} + t_{CSA2} + t_{reg} \end{aligned}$$

where t_{DSM} is the delay of the radix-4 digit selection module. This module is similar to that of Scheme C, but has two more inputs because of the tighter bounds required. We estimate $t_{DSM} = 3t_{FA}$.

Moreover, we estimate that the delay of the block CSA1 equals t_{FA}, whereas the delay of the second CSA is $t_{CSA2} = 0.5t_{FA}$. Then, the cycle time is

$$t_D = (0.5 + 3.0 + 0.5 + 1 + 0.5 + 0.5 + 1.5)t_{FA} = 8t_{FA}$$

For a 54-bit mantissa, division requires 18 iterations plus one cycle for rounding, resulting in

$$T_{D,DIV} = 19t_D \approx 150t_{FA}$$

On the other hand, for square root there are 16 cycles for iterations plus one for initialization of the first 6 bits plus one for postcorrection, so that

$$T_{D,SQRT} = 18t_D \approx 145t_{FA}$$

Table 4 summarizes the characteristics of the schemes we have compared.

6 Conclusions

We have presented an algorithm for very-high radix square root in which the result selection is obtained by rounding an estimate of the residual. This is made possible by scaling the recurrence by a factor M in such a way that $MS[j]$ is close to 1. This approach has the advantage that it is possible to combine the square root with the division algorithm presented in [7]. The combined implementation presented minimizes the time of division, so that the square root recurrence is performed in two cycles.

We perform a rough evaluation of the implementation and compare with other related schemes. From this comparison, we conclude that for division the scheme we present (for $r = 512$) produces a speed-up of 1.3 with respect to the faster of the other schemes (Scheme B), whereas for square root the speed-up is 0.9. Moreover, the implementation presented here has an estimated area of 60% of Scheme B.

Acknowledgments

We thank Roberto Tempo and Piera Del Col.

References

[1] W. S. Briggs and D. W. Matula, "Method and Apparatus for Performing Division Using a Rectangular Aspect Ratio Multiplier," U.S. Patent No.5 046 038, September 1991.

Table 4: Summary of time/area characteristics $[t_{FA}/A_{FA}]$

UNIT	$b = 9$			$b = 11$			$b = 14$			$b = 18$		
	DIV	SQRT	AREA	DIV	SQRT	AREA	DIV	SQRT	AREA	DIV	SQRT	AREA
our	75	125	1500	75	125	2200	65	105	3600	55	85	11100
A	155	175	1800	145	170	2500	120	145	4000	95	120	11500

UNIT	DIV	SQRT	AREA
B	100	110	2500(1)
C	170	170	-
D	150	145	-

(1): estimated area of multiplier only (not available area of lookup tables)

[2] W. S. Briggs T. B. Brightman and D. W. Matula, "Method and Apparatus for Performing the Square Root Function Using a Rectangular Aspect Ratio Multiplier," U.S. Patent No.5 060 182, October 1991.

[3] M. D. Ercegovac, "A Division With Simple Selection of Quotient Digits," Proc. 6th IEEE Symposium on Computer Arithmetic, 1983, pp.94-98.

[4] M. D. Ercegovac and T. Lang, "Simple Radix-4 Division with Operands Scaling," IEEE Trans. Comput., Vol.C-39, pp.1204-1208, September 1990.

[5] M. D. Ercegovac and T. Lang, "Module to Perform Multiplication, Division and Square Root in Systolic Arrays for Matrix Computations," Journal of Parallel and Distributed Computing, Vol. 11, No.3, March 1991, pp.212-221.

[6] M. D. Ercegovac and T. Lang, "Division and Square Root: Digit-Recurrence Algorithms and Implementations," Kluwer Academic Publishers, U.S.A., 1994.

[7] M. D. Ercegovac, T. Lang and P. Montuschi, "Very-High Radix Division with Prescaling and Selection by Rounding," IEEE Transactions on Computers, Vol. C-43, No.8, August 1994, pp.909-918.

[8] European Silicon Structures, ES2 ECPD10 Library Databook, April 1991.

[9] J. Fandrianto, "Algorithm for High Speed Shared Radix 8 Division and Radix 8 Square-Root," Proc. 9th IEEE Symposium on Computer Arithmetic, Santa Monica, CA, pp.68-75, September 1989.

[10] J. B. Gosling and C. M. S. Blakeley, "Arithmetic Unit with Integral Division and Square Root," IEE Proceedings, Part E, Vol. 134, pp.17-23, January 1987.

[11] J. Klir, "A Note on Svoboda's Algorithm for Division," Information Processing Machines, (Stroje na Zpracovani Informaci), 1963, No.9, pp.35-39.

[12] E.V. Krishnamurthy, "On Range-Transformation Techniques for Division, " IEEE Trans. Comput., vol. C-19, No. 2, Feb. 1970, pp. 157-160.

[13] T. Lang and P. Montuschi, "Higher Radix Square Root with Prescaling," IEEE Trans. Comput., Vol. C-41, No.8, Aug. 1992, pp.996-1009.

[14] T. Lang and P. Montuschi "Improved Methods to a Linear Interpolation Approach for Computing the Prescaling Factor for Very High Radix Division," I.R. DAI/ARC 6-94, Politecnico di Torino.

[15] T. Lang and P. Montuschi, "On the Design of Very-High Radix Combined Division and Square Root with Prescaling and Selection by Rounding," I.R. DAI/ARC 7-94, Politecnico di Torino.

[16] S.E. McQuillan, J.V. McCanny and R. F. Woods, "High Performance VLSI Architecture for Division and Square Root," Electronic Letters, Vol.27, No.1, pp.19-21, January 1991.

[17] S.E. McQuillan and J.V. McCanny "VLSI Module for High Performance Multiply, Square Root and Divide," IEE Proceedings, Part E, Vol.139, No.6, pp.505-510, June 1992.

[18] S.E. McQuillan, J.V. McCanny and R. Hamill, "New Algorithms and VLSI Architectures for SRT Division and Square Root," Proc. of the 11th IEEE Symposium on Computer Arithmetic, July 1993, Windsor, Ontario, Canada, pp.80-86.

[19] D.W. Matula, "Design of a Highly Parallel IEEE Floating Point Arithmetic Unit," Symposium on Combinatorial Optimization Science and Technology (COST), at RUTCOR/DIMACS, April 1991.

[20] P. Montuschi and L. Ciminiera, "Reducing Iteration Time When Result Digit is Zero for Radix-2 SRT Division and Square Root with Redundant Remainders," IEEE Transactions on Computers, Vol. C-42, No.2, February 1993, pp.239-246.

[21] A. Svoboda, "An Algorithm for Division," Inf. Proc. Mach., Vol.9, pp.25-32, 1963.

[22] C. Tung, "A Division Algorithm for Signed-Digit Arithmetic," IEEE Trans. Comput., Vol.C-17, 1970, pp.887–889.

[23] Weitek, W4164 and W4364 Floating Point Processors, Technical Overview, Oct. 1990.

[24] D. C. Wong and M. J. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations," IEEE Trans. Comput., Vol. 41, Aug. 1992, pp.981–995.

[25] J. H. P. 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, January 1987.