Boosting Very-High Radix Division with Prescaling and Selection by Rounding

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

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

Abstract

An extension of the very-high radix division with prescaling and selection by rounding is presented. This extension consists in increasing the effective radix of the implementation by obtaining a few additional bits of the quotient per iteration, without increasing the complexity of the unit to obtain the prescaling factor nor the delay of an iteration. As a consequence, for some values of the effective radix, it permits an implementation with a smaller area and the same execution time than the original scheme. Estimations are given for 54-bit and 114-bit quotients.

1. Introduction

Division by digit recurrence produces one digit of the quotient per iteration. Consequently, for a given precision a higher radix results in fewer iterations and a potential faster execution. However, as the radix increases the digit-selection function becomes more complicated, which increases the cycle time and can overcome the reduction in execution time. Because of this, practical implementations are limited to up to radix-8 stages and radix-16 dividers have been implemented using overlapped radix-2 and radix-4 stages. Extensive literature exists on this subject; for specific references see for instance [5, 9, 14]

A way of achieving speedup with larger radices is to simplify the selection function by prescaling the divisor [4, 8, 10, 17]. In particular, prescaling and selection by rounding has been shown to produce a significant speedup [6]. We refer to this scheme as the VHR approach, which has been extended to square root [12] and $\sqrt{x/d}$ [1].

In principle, the speedup of the VHR approach should increase by increasing the radix. However, the increase in the radix increases the complexity of obtaining the prescaling factor and the size of the rectangular multiplier, which might produce an increase in the cycle time. Moreover, the area of the module to obtain the prescaling factor becomes the predominant contributor to the overall area, as illustrated in Table 1 [1].

[1] Here are shown improved values with respect to Table 4 in [6], since we used an improved linear approximation developed in [12] and [11].

Table 1. Execution time & area of VHR division

Quotient bits per iteration	9	11	14	18
Cycle time (t_{fa})	7.5	9.0	9.0	9.0
No. of cycles (54-bit quotient)	10	9	8	7
Execution time (t_{fa})	75	81	72	63
Area MAC (A_{fa})	520	710	740	970
Area presc. factor module (A_{fa})	130	220	470	1750

t_{fa}, A_{fa}: delay and area of full adder

In this paper we increase the effective radix by obtaining a few additional bits of the quotient per cycle, while maintaining the complexity of the prescaling factor calculation. This results in a reduction of the overall area for the same execution time. Specifically, we estimate that for 54 bits, in the implementation for 18 quotient bits per iteration the area of the prescaling factor module plus the area of the multiplier, can be reduced by 30%. Moreover, for the case of a quotient of 114 bits (extended precision) the boosting can achieve a speedup of about 10% for the same area. We describe the algorithm, show the architecture, and estimate the improvement obtained with respect to VHR division. We assume familiarity with [5] and [6].

2. Review of VHR division

As described in [6] in this approach the division is performed in the unit of Figure 1, as follows

- Cycle 1: determine the scaling factor M as an approximation of $1/d$ such that
$$1 - \frac{R-2}{4R(R-1)} < Md < 1 + \frac{R-2}{4R(R-1)}$$
where d is the divisor and R the radix.
- Cycle 2: obtain the scaled divisor $z = Md$ (carry-save).
- Cycle 3: obtain the initial residual $w[0] = Mx$ (carry-save), where x is the dividend. Assimilate z.
- Cycles 4 to $\lceil \frac{n}{log_2(R)} \rceil + 3$: obtain a quotient digit q_{j+1} by rounding and perform the recurrence, i.e.
$$q_{j+1} = round(\widehat{Rw[j]})$$

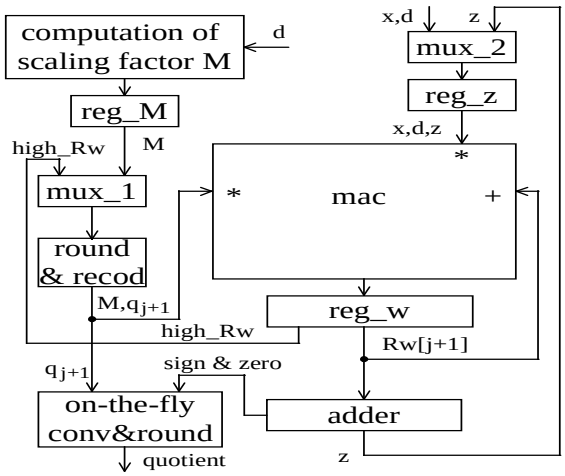

Figure 1. Architecture of VHR division of [6]

where $\widehat{Rw[j]}$ is the (redundant) shifted residual truncated at the t-th fractional bit, and

$$w[j+1] = Rw[j] - q_{j+1}z$$

The residual is left in carry-save form.

- Cycle $\lceil \frac{n}{log_2(R)} \rceil + 4$: postcorrection and rounding. The quotient is corrected if the last residual is negative and the quotient is rounded.

The cycle time is determined by the maximum of the recurrence and of the calculation of the prescaling factor. The main contributors to the area are the rectangular MAC and the module to calculate the prescaling factor. Different methods can be used for the calculation of the prescaling factor, which is an approximation of the reciprocal of the divisor [3, 7, 15, 16, 18]. In [6] we have shown the use of a linear approximation, which has been improved in [12] and [11]. The results obtained are illustrated in Table 1.

3. Boosting algorithm and architecture

As indicated in the Introduction, we propose to extend the algorithm so that a few additional bits of the quotient are obtained per cycle. Specifically, the overall radix R is decomposed into two factors such that

$$R = CB \quad with \;\; C = 2^c \quad and \quad B = 2^b$$

where now B corresponds to the radix of the "normal" VHR algorithm and c are the additional bits of the quotient digit obtained per cycle.

The prescaling is done according to the radix B so that

$$1 - \frac{B-2}{4B(B-1)} < z = Md < 1 + \frac{B-2}{4B(B-1)}$$

The recurrence is then

$$w[j+1] = CBw[j] - q_{j+1}z \tag{1}$$

with $q_{j+1} = Cp_{j+1} + s_{j+1}$, and where

- the digit p_{j+1} is selected according to the VHR radix-B algorithm;
- the digit s_{j+1}, which represents the extension of the few additional bits mentioned at the beginning of this section, is selected according to the radix-C boosting algorithm, presented here.

As described in [6], the digit set of the VHR digit is

$$-(B-1) \le p_j \le B \tag{2}$$

where the value B is required only in the first iteration. Moreover, for the digit s_{j+1} we have

$$-\max(\frac{3}{4}C - 1, 1) \le s_{j+1} \le C - 1 \tag{3}$$

as determined in Section 4.1.

The selection function has two components. For p_{j+1} we perform selection by rounding, as done in [6]. That is,

$$p_{j+1} = round(\widehat{Bw[j]}) \tag{4}$$

where $\widehat{Bw[j]}$ is the truncation of the shifted residual $Bw[j]$ to its $t-$th fractional bit. To allow this selection by rounding we need that [2]

$$-1 + \frac{1}{2B} + \frac{2^{-t}}{B} \le w[j] < 1 - \frac{1}{2B} \tag{5}$$

which, for $t = 2$ as in [6], results in

$$-1 + \frac{3}{4B} \le w[j+1] < 1 - \frac{1}{2B} \tag{6}$$

On the other hand, the selection of s_{j+1} is done as

$$s_{j+1} = sel(\widehat{w[j]}, \widehat{z}, \widehat{p_{j+1}}) \tag{7}$$

where $\widehat{w[j]}$, $\widehat{z}$ and $\widehat{p_{j+1}}$ are estimates of the corresponding quantities, as determined later.

The architecture to perform this algorithm, shown in Figure 2, is very similar to that of the original VHR division, with the addition of the selection function for s_{j+1}, the generation of $s_{j+1}z$, and the corresponding extension of the subtraction. The selection of s_{j+1} is performed using the estimates P, H and E, as discussed in the next section.

4. Selection function for s_{j+1}

To obtain the selection function for s_{j+1} we define $v[j]$, the remainder obtained after performing the iteration radix B. That is,

$$v[j] = Bw[j] - p_{j+1}z \tag{8}$$

[2] In [5] the improved analytical expression (5) has been introduced, and it has been shown that it influences neither the analytical and numerical results nor the implementation of [6].

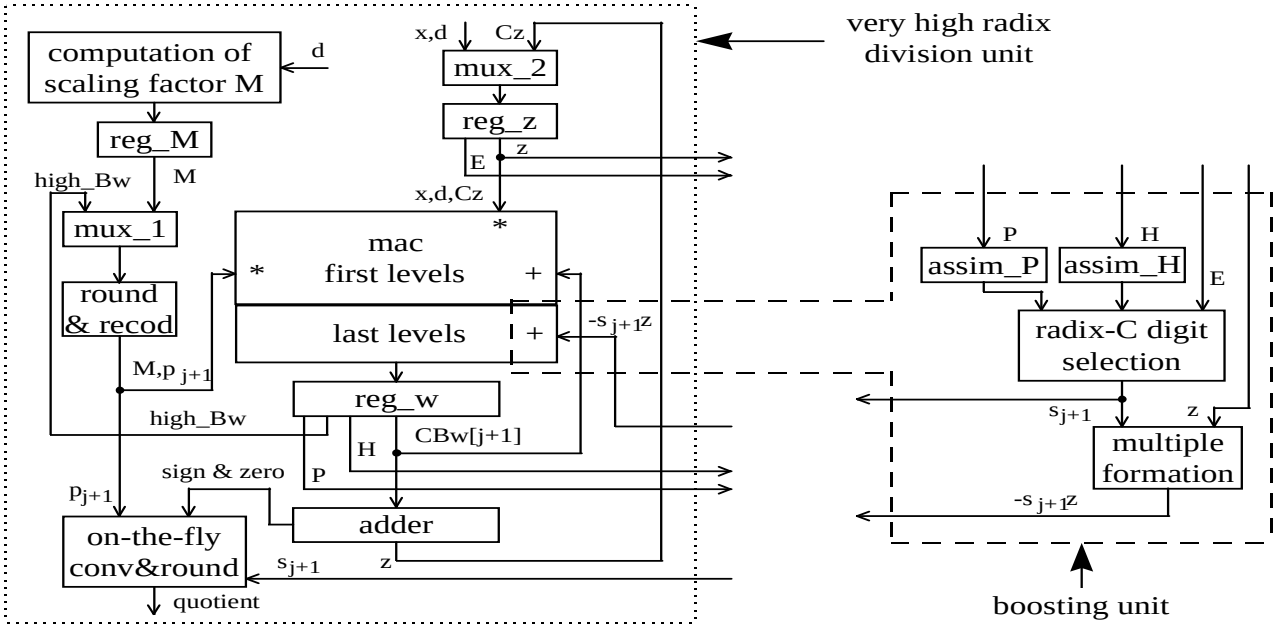

Figure 2. Proposed architecture

Then

$$w[j+1] = Cv[j] - s_{j+1}z \tag{9}$$

This means that the selection of s_{j+1} corresponds to a radix-C division with $w[j]$ replaced by $v[j]$ and the restrictions

$$-D_L = -1 + \frac{3}{4B} \leq w[j+1] < 1 - \frac{1}{2B} = D_H \tag{10}$$

and

$$1 - \frac{B-2}{4B(B-1)} < z < 1 + \frac{B-2}{4B(B-1)} \tag{11}$$

We first determine the digit set of s_{j+1} and then the selection function.

4.1. Choice of digit set for s_{j+1}

In this section we determine the digit set of s_{j+1}. To do this, we need to determine bounds on the value of $v[j]$. We decompose $v[j]$ as follows:

$$\begin{aligned} v[j] &= Bw[j] - p_{j+1}z \\ &= (Bw[j] - p_{j+1}) - p_{j+1}(z-1) \\ &= h[j] - p_{j+1}e \end{aligned} \tag{12}$$

where

$$h[j] = Bw[j] - p_{j+1} \quad and \quad e = z - 1$$

Because p_{j+1} is selected by rounding we get

$$-\frac{1}{2} \leq h[j] < \frac{1}{2} + 2^{-t} = \frac{3}{4} \tag{13}$$

for $t = 2$. Moreover, because of the prescaling

$$|e| < \frac{B-2}{4B(B-1)} = A \tag{14}$$

Consequently, the upper bound of $v[j]$ is

$$v[j] = h[j] - p_{j+1}e < \frac{3}{4} + (B-1)A = 1 - \frac{1}{2B} \tag{15}$$

Note that we do not consider the special case $p_1 = B$ since this produces $v[1] < 0$, as shown in [6].

For the lower bound the worst case is produced by using $p_{j+1} = B$, which can occur in the first iteration, leading to

$$v[j] = h[j] - p_{j+1}e > -\frac{1}{2} - BA = -\frac{3}{4} + \frac{1}{4(B-1)} \tag{16}$$

In summary, from (15) and (16), the domain of $v[j]$ is

$$-\frac{3}{4} + \frac{1}{4(B-1)} < v[j] < 1 - \frac{1}{2B} \tag{17}$$

Since the domain of $v[j]$ is not symmetric, this results in a nonsymmetric digit set for s_{j+1}. Let us define the digit set

$$-s_{min} \leq s_{j+1} \leq s_{max}$$

We start with the determination of the minimum s_{max}, which guarantees that (10) holds, i.e. $-1 + 3/(4B) \leq w[j+1] < 1 - 1/(2B)$. From (9) we have that the worst-case situation occurs when $v[j]$ is at its maximum value and z has the minimum value of its domain; in this case,

$$w[j+1] = Cv[j] - s_{max}z < C\left(1 - \frac{1}{2B}\right) - s_{max}(1-A)$$

Since $w[j+1] < 1 - 1/(2B)$ must be guaranteed, we get

$$C\left(1 - \frac{1}{2B}\right) - s_{max}(1-A) \leq 1 - \frac{1}{2B} \tag{18}$$

which results in

$$s_{max} \geq C - 1$$

Now s_{min} should guarantee that $w[j+1] \geq -(1-3/(4B))$. We have

$$w[j+1] = Cv[j] + s_{min}z \geq C(-\frac{1}{2} - p_{j+1}(z-1)) + s_{min}z \quad (19)$$

The computations show that the minimum value of (19) is achieved for $z = 1 - A$ and $p_{j+1} = -(B-1)$, i.e.

$$w[j+1] = Cv[j] + s_{min}z > C(-\frac{3}{4} + \frac{1}{2B}) + s_{min}(1-A)$$

Since $w[j+1] \geq -(1 - 3/(4B))$,

$$C(-\frac{3}{4} + \frac{1}{2B}) + s_{min}(1-A) \geq -(1 - \frac{3}{4B})$$

$$s_{min} \geq \left(\frac{3}{4}C - 1 - \frac{1}{2B}\left(C - \frac{3}{2}\right)\right)\frac{1}{1-A} = S \quad (20)$$

Let us consider now the right hand side of (20). We have by expansion of $1/(1-A)$,

$$\begin{aligned} &\tfrac{3}{4}C - 1 - \tfrac{1}{2B}\left(C - \tfrac{3}{2}\right) < S \\ &< \left(\tfrac{3}{4}C - 1 - \tfrac{1}{2B}\left(C - \tfrac{3}{2}\right)\right)(1+A) \end{aligned}$$

Now, for $2 \leq C \leq B$ we have $0 < \frac{1}{2B}\left(C - \frac{3}{2}\right) < \frac{1}{2}$ and $\frac{3}{4}CA - \frac{1}{2B}(C - \frac{3}{2}) - A < 0$, (since $A < \frac{1}{4B}$); therefore,

$$\frac{3}{4}C - \frac{3}{2} < S < \frac{3}{4}C - 1$$

Since s_{min} should be an integer, we get as necessary and sufficient condition for the smallest value of s_{min}

$$s_{min} = \begin{cases} 1 & \text{if } C = 2 \\ \frac{3}{4}C - 1 & \text{if } C \geq 4 \end{cases}$$

In conclusion,

$$-\max(\frac{3}{4}C - 1, 1) \leq s_{j+1} \leq C - 1 \quad (21)$$

4.2. Selection intervals and selection constants

We first determine the selection intervals and then use them to determine conditions for the selection constants.

4.2.1 Selection intervals

Since the region of convergence of the algorithm is given by (10), we obtain, from (9), the following selection interval for s_{j+1}.

$$-D_L + s_{j+1}z \leq Cv[j] \leq D_H + s_{j+1}z \quad (22)$$

However, the boosting algorithm should operate concurrently with the VHR part, which implies that $v[j]$ is not known. Consequently, we develop the selection function from $h[j]$ instead of $v[j]$. Replacing $v[j]$ by $h[j] - p_{j+1}e$ in (22) we get the bounds of the selection interval $[L_k, U_k]$ such that if $L_k \leq Ch[j] \leq U_k$ then $s_{j+1} = k$ can be selected. We get

$$L_k = -D_L + Cp_{j+1}e + k(1+e)$$

$$U_k = D_H + Cp_{j+1}e + k(1+e)$$

4.2.2 Selection constants

Now, as in standard digit-recurrence division [5], the selection function is described by the selection constants m_k so that the value $s_{j+1} = k$ is selected if $m_k \leq Ch[j] < m_{k+1}$. Continuity requires that $L_k \leq m_k \leq U_{k-1}$. As is, m_k depends on p_{j+1} and e. Moreover, the selection requires knowledge of $h[j]$ with full precision. Since there is overlap between the selection intervals, it is possible to use estimates of these three quantities.

If we call P the estimate of p_{j+1} and E the estimate e, we get

$$\max(L_k(P,E)) \leq m_k(P,E) \leq \min(U_{k-1}(P,E)) \quad (23)$$

where the maximum and the minimum are determined for the range of values of p_{j+1} and e defined by the estimates P and E, respectively. This expression assumes $Ch[j]$ with full precision.

Estimate of $Ch[j]$

To avoid the knowledge of $Ch[j]$ with full precision, we use an estimate obtained from a few bits of the carry-save representation of $w[j]$, as follows. Since

$$h[j] = Bw[j] - p_{j+1}$$

and p_{j+1} is determined by rounding as described in (4), we obtain that

$$h[j] = -y_1 + \sum_{i=1}^{n} y_i 2^{-i}$$

where y_i is the i-th digit of the representation of $y[j] = Bw[j]$. That is, $y_1.y_1y_2\ldots$ is the two's complement representation of $h[j]$. Because of the way the rounding is performed y_1 and y_2 have values $0, 1$ and the rest is in carry-save form.

Consequently, the estimate H of $Ch[j]$ can be obtained from a truncated $Cy[j]$. Because of the two's complement representation we get

$$H \leq Ch[j] < H + \delta_H$$

Moreover, if the truncation is done at the f fractional bit,

$$\delta_H = 2^{-f+1} \quad (24)$$

Table 2. Sufficient conditions for selection

g (bits of E)	u (bits of P)	f (frac. bits of H)
c	$c+3$	3
$c+1$	$c+2$	3
$c+1$	$c+3$	2

As a consequence, when the estimate H of $Ch[j]$ is used, we get the selection intervals

$$\max(L_k(P,E)) \leq H \leq \min(U_k(P,E)) - \delta_H \quad (25)$$

and expression (23) is transformed to

$$\max(L_k(P,E)) \leq m_k(P,E) \leq \min(U^*_{k-1}(P,E)) - \delta_H \quad (26)$$

where the * indicates the value larger than U_{k-1}, with the granularity of H. To determine suitable values of $m_k(P,E)$, we need to specify the way the estimates P and E are computed.

Estimate of p_{j+1}

Because of the selection by rounding, p_{j+1} is directly derived from the most significant $(b+2)$ integer and 2 fractional bits of $Bw[j]$. We estimate p_{j+1} by considering the most significant u binary weights of $Bw[j]$, i.e. those from weight 2^{b+1} to weight 2^{b+2-u}. So, if the estimate P is represented in (assimilated) two's complement representation while the rest of $Bw[j]$ remains in carry-save representation we get

$$P \leq Bw[j] < P + \delta_P$$

where

$$\delta_P = 2(2^{b+2-u}) \quad (27)$$

Consequently, from (4) it follows that

$$P \leq p_{j+1} \leq P + \delta_P \quad (28)$$

By combining (28) with (2), we get

$$P_L = \max(P, -(B-1)) \leq p_{j+1} \leq \min(P+\delta_P, B) = P_H \quad (29)$$

Estimate and range of e

Since e, obtained from z, is in two's complement representation we obtain

$$E_L = \max(E, -A) \leq e < \min(E + \delta_E, A) = E_H \quad (30)$$

Moreover, since $A = \frac{B-2}{4B(B-1)}$ the width of the range of e is slightly less than $2^{-(b+1)}$. Consequently,

$$\delta_E = 2^{-(b+1+g)}$$

where g is the number of "significant" bits of e used in the estimate.

The estimates P and E are then used to determine the $m_k(P,E)$ from expression (26). This has to be done taking into account the ranges and signs of P, E, and k; the detailed list of selection rules is reported in [13].

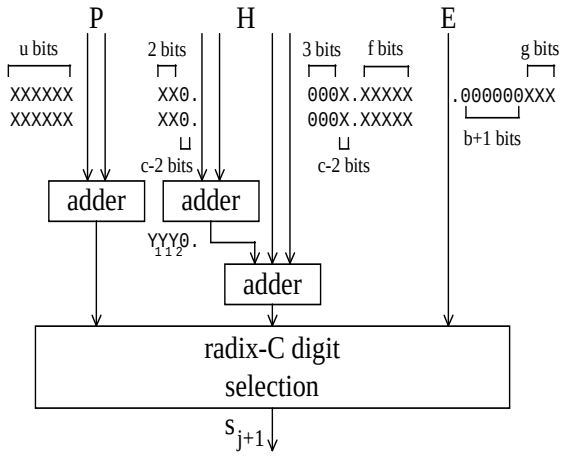

Figure 3. Implementation of s_{j+1} selection

4.2.3 Condition for the estimates

From (26), it follows that a necessary and sufficient condition for continuity for obtaining $m_k(P,E)$ of granularity of H is that for all P, E

$$(\min(U^*_{k-1}(P,E)) - \delta_H) - \max(L_k(P,E)) \geq 0 \quad (31)$$

Now, since from its definition, $U^*_{k-1} > U_{k-1}$, we have that the necessary and sufficient condition (31) evolves to the following sufficient condition

$$(\min(U_{k-1}(P,E)) - \delta_H) - \max(L_k(P,E)) \geq 0$$

This has to be evaluated for all combinations of signs. For example for the case $P > 0$, $E > 0$, and $k > 0$ we obtain

$$D_L + D_H - C(P_H E_H - P_L E_L)$$
$$-(1+E_L) - k(E_H - E_L) - \delta_H \geq 0$$

The worst case is for $P_H = B$, $P_L = B - \delta_P$, $E_H = A$, $E_L = 2^{-b-2} - \delta_E$, and $k = C-1$. Substituting these values and the δ's we get

$$1 - 2^{-u+c+1} - 2^{-g+c-1}(1+2^{-b}) - 3 \cdot 2^{-b-1}$$
$$-2^{-f+1} - 2^{-2b-2} + 2^{-u+c+2-g} + 2^{c-b-2}$$
$$+2^{-b-g} + 2^{c-2b-2} \geq 0 \quad (32)$$

A set of solutions of (32) is reported in Table 2. These solutions do not depend on the very high radix B, but only on the boosting radix C, when $C \leq B/8$.

5. Implementation

In this section we describe the implementation of the boosting VHR division. As indicated in Section 3, the architecture for the proposed scheme is the same as that presented in [6], except for the the following blocks:

1. The selection function of s_{j+1}

2. The generation of $s_{j+1}z$
3. The incorporation of $s_{j+1}z$ into the MAC
4. The production of $q_{j+1} = Cp_{j+1} + s_{j+1}$

Consequently, we now concentrate on the implementation of these blocks.

Selection function of s_{j+1}
As described in Section 4, the selection function requires the assimilation of parts of the carry-save representation of $w[j]$ and then a function on the resulting bits. This implementation is shown in Figure 3. Note the two-level assimilation required for H, which is due to the fact that the rounding to produce p_{j+1} uses $Bw[j]$ up to the the second fractional bit. Moreover, the estimate E is obtained by a truncation of $e = z - 1$, which is produced by inverting the integer bit of z. Of course, this inversion is not actually necessary to feed the selection function. Moreover, from (14) and as explained in section 4.2.2 on the range of e, only the bit weights from weight $2^{-(b+2)}$ to weight $2^{-(b+1+g)}$ of E enter the selection function.

Generation of $s_{j+1}z$ and residual updating
In general, the generation of $s_{j+1}z$ requires a rectangular multiplier. This consists of two parts: generation of partial products and addition. The addition can be performed as part of the MAC tree, together with the updating of the residual. For instance, for radix-4 ($C = 4$) since $-2 \leq s_{j+1} \leq 3$ two terms are required, unless the multiple $3z$ is precomputed. Two terms are also required for $C = 8$.

Incorporation into the MAC tree
Because the time of the selection function for s_{j+1} is larger than that for p_{j+1} (done by rounding and recoding), to avoid that the boosting produces an increase in the cycle time, it is necessary to introduce the additional terms to lower levels of the MAC tree. This is possible if the original tree is not complete, as illustrated in Figure 4. Since the MAC in the VHR division is used both for the recurrence and for the prescaling, the number of available slots at different levels depends on the radix B and on the number of bits of the prescaling factor. Specifically, for the recurrence the number of inputs to the tree is $\lceil (b+1)/2 \rceil + 2$ and for the prescaling $\lceil (m+2)/2 \rceil$, where m is the number of fractional bits of the prescaling factor (since $1 \leq M < 2$ and M must be represented, for recoding purposes, in two's complement).

Production of q_{j+1}
Finally, it is necessary to produce $q_{j+1} = Cp_{j+1} + s_{j+1}$ in two's complement for the on-the-fly conversion. The implementation is simple and is not in the critical path.

6. Evaluation and comparison

We now give a rough estimation of the execution time and the area of the boosting VHR division and compare

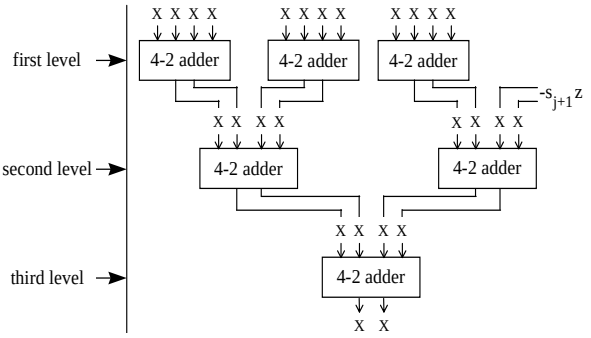

Figure 4. Levels of the MAC tree

with VHR division. As was seen for the VHR scheme [6], the radices that have to be considered are the lowest that achieve a reduction in the number of cycles. Consequently, we need to compare the VHR with radix R with the boost approach with $R = BC$. For the execution time we can give the following general considerations. The cycle time is determined by the maximum of the time to compute the prescaling factor and the time to perform an iteration of the recurrence. Moreover,

- the time to compute the scaling factor depends on R for the VHR approach and on B for the boosting approach. Consequently, this time might be reduced by using the boosting approach.
- the time of an iteration is the same for both schemes, as long as the delay of the selection function for s_{j+1} and the generation of the multiple is overlapped by the recoding and rounding, the multiplexer, the multiple generator, and the first levels of the MAC tree.

Consequently, the addition of the boosting can produce a speedup if the delay of the calculation of the scaling factor is the critical component. This depends on the way this calculation is performed. The implementation model we use shows that it is reasonable to assume that the delay of this calculation is not critical. Consequently, for the same radix R, we do not expect the boosting to produce a speedup.

With respect to the area, the main components are the MAC and the calculation of the prescaling factor. The area of the MAC increases somewhat when adding the boosting, because of the partitioning of the multiplier into two parts. On the other hand, the reduction in the area of the module to calculate the prescaling factor should be substantial, when going from R to radix $B = R/C$. This is the main advantage of the boosting technique. We now perform an estimate of this area reduction.

6.1. Choice of C

Acceptable values of C are determined by the following considerations:

Table 3. Available slots* in the MAC tree

b	9-10	11-12	13-14	15-16	17-18	19-20	21-26
$\lceil (b+6)/2 \rceil$	8	9	10	11	12	13	14-16
slots	0	3	2	2	2	1	0

* in second level

- A larger value of C produces a larger reduction in the number of iterations.
- For larger C the selection function for s_{j+1} is more complicated, resulting in a larger delay and area.
- A larger C requires larger values of s_{j+1} and therefore a larger number of additional terms to be added in the MAC.

The combination of these considerations have lead us to estimate that acceptable values of C are 4 and 8. For $C = 4$ two additional terms have to be added if $3z$ is not precomputed, whereas only one term is required if it is precomputed. The radix-8 case requires two additional terms.

6.1.1 Selection for $C = 4$

To estimate the delay and area of the selection function we have performed a synthesis for $C = 4$. As indicated in Section 4, the digit set is $-2 \leq s_{j+1} \leq 3$. Moreover, the design of the selection function requires the choice of parameters f, g, and u, and the determination of the selection constants m_k. The actual implementation is done using a multilevel gate network where the adders required for the assimilation should be included in the delay optimization. Several selection functions exist, for different choices of parameters [13]. As an example, we have performed a synthesis of the selection function corresponding to $g = 3$, $u = 4$ and $f = 3$, using Compass Passport $0.6\mu m$ standard cell library [2]. The resulting delay is of $3.6ns.$, which corresponds to less than $4t_{fa}$. The number of gates is 320, corresponding to about $30A_{fa}$.

The delay of less than $4t_{fa}$ allows us to enter the tree at the second level, without increasing the cycle time, since the delays of recoding and rounding, the multiplexer, the multiple generation, and the first level of the MAC tree, according to the assumptions of [6], amount to a total of $4.5t_{fa}$. Consequently, we need to use a radix B such that there are one (if $3z$ is precomputed) or two slots available in the second level of the tree.

Although we have not performed a synthesis for $C = 8$, we estimate that it might be possible to use also this radix without increasing the cycle, if the additional terms enter the tree at the second level.

6.2. Allowable values of B

As indicated above the allowable values of B are those that leave one or two empty slots in the second level of the MAC tree. As described in section 5 this depends on the recurrence and also on the number of bits of the prescaling factor, which depends on the way this factor is computed [3, 7, 15, 16, 18]. Using the L-approach described in [12] (and, more in general in [11]) we obtain that the number of inputs to the tree required by the VHR division is $\lceil (b+6)/2 \rceil$. Consequently, the number of empty slots is shown in Table 3. Therefore, since for $C = 4$ and $C = 8$ we need two slots, in the sequel we consider $11 \leq b \leq 18$.

Table 4. Numbers of cycles

no. cycles	10	9	8	7
$log_2 R$	9	11	14	18

(a) 54-bit quotient

no. cycles	17	16	15	14	13	12	11	10
$log_2 R$	9	10	11	12	13	15	17	19

(b) 114-bit quotient

Table 5. Area of MAC + presc.-factor module

$log_2 R$	11	14	18
no. cycles	9	8	7
exec. time	80	70	60
VHR	930	1200	2800
VHR+boost ($C = 4$)	-	1200	2000
Area ratio		1.0	0.70

(a) 54-bit quotient

$log_2 R$	11	12	13	15	17	19
no. cycles	15	14	13	12	11	10
exec. time	135	125	115	110	100	90
VHR	1500	1500	1700	2400	3300	5400
VHR+boost ($C = 4$)	-	-	1800	2100	2800	3600
Area ratio			1.0	0.85	0.85	0.65
VHR+boost ($C = 8$)	-	-	-	1900	2100	2900
Area ratio				0.80	0.65	0.55

(a) 114-bit quotient

6.3. Suitable values of $R = BC$

As described in [13], the suitable values of R depend on the required precision of the quotient; namely, the suitable values are the smallest that produce a given number of cycles. These values for $n = 54$ and $n = 114$ bits (double and extended precision, respectively) are as reported in Table 4.

6.4. Execution time and area

Table 5 shows estimates of the execution time and area of VHR division and of the version with boosting, using the models presented in [12]. As can be seen from the table, for 54 bits the boosting technique is only effective for $log_2 R = 18$, in which case the area ratio (VHR+boost)/(VHR) is 0.7. On the other hand, for 114 bits it is effective for $log_2 R = 16, 17$ and 19, producing ratios between 0.85 and 0.65. Moreover, for 114 bits we estimate that additional area reductions can be achieved by using $C = 8$.

Figure 5 shows the tradeoff between area and speedup, using as reference values for delay and area the radix-2 implementation as reported in [5], i.e. area equal to $460A_{fa}$

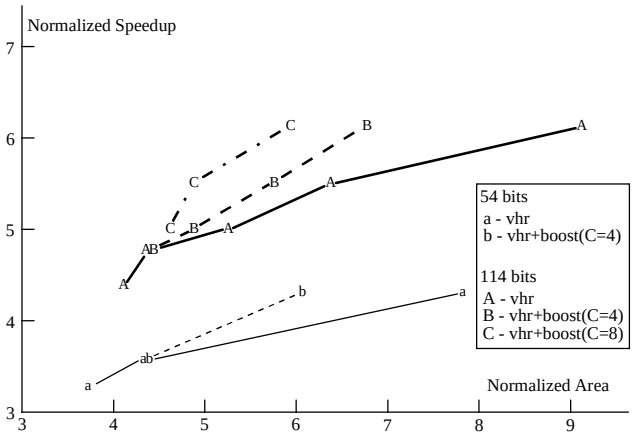

Figure 5. Speedup vs. Area comparisons

and $780A_{fa}$ for 54 and 114 bits, respectively, and delay equal to $260t_{fa}$ and $550t_{fa}$ again for 54 and 114 bits, respectively. For 54 bits we estimate a VHR+boost ($C = 4$) unit 4 times faster than the "classical" radix-2 architecture, requiring 6 times its hardware; in this case, the estimated area saving is about 30%, with respect to the VHR unit with the same delay. On the other hand, for 114 bits both very-high radix implementations produce speedups of up to 6; the reduction in area of the implementations of the boosting algorithm being of 25% (for C=4) and of 35% (for C=8), with respect to the standard VHR implementation. Moreover, from Table 5b we observe that for 114 bits using an area of about $3000A_{fa}$, we can design either a VHR radix-2^{17} unit with total delay of $100t_{fa}$ or a VHR+boost radix-2^{19} unit (with $C = 8$) with total delay 10% smaller.

7. Conclusions

We have presented an algorithm and implementation that increases the effective radix of the very-high radix division approach presented in [6]. This is accomplished by obtaining a few additional bits of the quotient per iteration without increasing the complexity of the module to obtain the scaling factor, nor the iteration delay.

We show that for some values of the effective radix this approach results in a significant reduction in the area of the module to compute the prescaling factor with respect to the original scheme. As a consequence, it is possible to achieve values of the execution time with a smaller unit.

We expect this approach to be useful for other related operation such as square root and $\sqrt{x/d}$.

Acknowledgment

We thank Alberto Nannarelli for the help in the synthesis of the proposed unit.

References

[1] E. Antelo, T. Lang, and J. Bruguera. Computation of $\sqrt{x/d}$ in a very high radix combined division/square-root unit with scaling. *IEEE Trans. Comput.*, C-47(2):152–161, February 1998.

[2] Compass Design Automation. *Passport - 0.6 Micron, 3-Volt, High Performance Standard Cell Library*. Compass Design Automation, Inc., 1994.

[3] D. DasSarma and D. Matula. Faithful bipartite rom reciprocal tables. In *Proc. of the 12th IEEE Symposium on Computer Arithmetic*, pages 12–25, Bath, England, July 1995.

[4] M. Ercegovac and T. Lang. Simple radix-4 division with operands scaling. *IEEE Trans. Comput.*, C-39(9):1204–1208, September 1990.

[5] M. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Press, New York, NJ, 1994.

[6] M. Ercegovac, T. Lang, and P. Montuschi. Very high radix division with prescaling and selection by rounding. *IEEE Trans. Comput.*, C-43(8):909–917, August 1994.

[7] M. Ito, N. Takagi, and S. Yajima. Efficient initial approximation for multiplicative division and square root by a multiplication with operand modification. *IEEE Trans. Comput.*, C-46(4):495–498, April 1997.

[8] J. Klir. A note on Svoboda's algorithm for division. *Information Processing Machines, (Stroje na Zpracovani Informaci)*, 9:35–39, 1963.

[9] I. Koren. *Computer Arithmetic Algorithms*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[10] E. Krishnamurthy. On range-transformation techniques for division. *IEEE Trans. Comput.*, C-19(2):157–160, February 1970.

[11] 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, Dipartimento di Automatica e Informatica*, 1994.

[12] T. Lang and P. Montuschi. Very high radix combined division and square root with prescaling and selection by rounding. In *Proc. of the 12th IEEE Symposium on Computer Arithmetic*, pages 124–131, Bath, England, July 1995.

[13] P. Montuschi and T. Lang. An algorithm for boosting very high radix division with prescaling and selection by rounding. *I.R. DAI/ARC 4-98, Dipartimento di Automatica e Informatica*, 1998.

[14] S. Oberman and M. Flynn. Division algorithms and implementations. *IEEE Trans. Comput.*, C-46(8):833–854, August 1997.

[15] M. Schulte and J. Stine. Symmetric bipartite tables for accurate function approximation. In *Proc. of the 13th IEEE Symposium on Computer Arithmetic*, pages 175–183, Asilomar, CA, July 1997.

[16] E. Schwarz and M. Flynn. Hardware starting approximation method and its application to the square root operation. *IEEE Trans. Comput.*, C-45(12):1356–1369, December 1996.

[17] A. Svoboda. An algorithm for division. *Inf. Proc. Mach.*, 9:25–32, 1963.

[18] N. Takagi. Generating a power of an operand by a table lookup and a multiplication. In *Proc. of the 13th IEEE Symposium on Computer Arithmetic*, pages 126–131, Asilomar, CA, July 1997.