Very High Radix Division with Selection by Rounding and Prescaling

Miloš D. Ercegovac† *Tomas Lang*‡ *Paolo Montuschi*⋆

† Computer Science Department,
University of California, Los Angeles
‡ Department of Electrical and Computer Engineering
University of California at Irvine
⋆ Dipartimento di Automatica e Informatica,
Politecnico di Torino

Abstract

A division algorithm in which the quotient-digit selection is performed by rounding the shifted residual in carry-save form is presented. To allow the use of this simple function, the divisor (and dividend) is prescaled to a range close to one. The implementation presented results in a fast iteration because of the use of carry-save forms and suitable recodings. The execution time is calculated and several convenient values of the radix are selected. Comparison with other high radix dividers is performed using the same assumptions.

1 Introduction

Often used division algorithms are based on recurrences producing one quotient-digit per iteration [6, 9]. To reduce the number of iterations, it is advantageous to use the highest possible radix for the quotient-digit representation. However, the complexity of the quotient-digit selection increases for higher radices, eliminating the advantage of the reduction in number of iterations [6]. To reduce the complexity of the quotient-digit selection it is possible to prescale the divisor (and the dividend, to preserve the value of the quotient) to a range close to unity [14, 5, 6, 10]. Using this approach, we develop a scheme that has a fast division step due to the following features:

- The quotient-digit selection is performed by rounding an estimate of the shifted residual.
- The calculation of the scaling factor is done using a linear interpolation approach, whose delay matches the delay of the recurrence.
- All operations (except the scaling of the divisor) are performed using a carry-save representation, thus avoiding time-consuming assimilations.

1.1 Related work

We mention division algorithms that are characterized by features similar to our approach. Svoboda [14] developed a radix-10 division scheme with scaling of divisor (and dividend) so that the quotient-digit is obtained as the most-significant digit of the partial remainder. A standard non-restoring division recurrence with carry-propagate adder is used. Tung [16] extended the scheme to an arbitrary radix and used a signed-digit adder. Krishnamurthy [10] summarizes generalizations of the range transformation techniques for division in conventional and signed-digit number systems thus allowing the use of redundant adders. Ercegovac [1] proposes a division algorithm with a simplified quotient-digit selection by solving $y = ay + b$ where y is in redundant digit-serial form, processed most-significant digit first, and the coefficients a and b are in digit-parallel form. Each output digit is obtained by rounding the shifted residual to the integer part. Scaling is required to transform the divisor into a range around 1. A radix-16 division algorithm using a continued-product method is presented in [3]. A higher radix division with range operands transformation, a quotient digit selection function with prediction, and a carry-save adder for computing partial residuals is discussed in [2]. A radix-4 implementation is described in [5].

Waser and Flynn [17] describe a version of Newton-Raphson iterative algorithm called byte division. It uses a byte multiplier (8×56) and an (8×8) multiplier

1063-6889/93 $03.00 © 1993 IEEE

to produce one radix-256 quotient digit per iteration. The algorithm uses an 8-bit reciprocal approximation (for scaling) and a carry-propagate adder for the iteration. Several other related schemes are described in Section 6 and compared with the scheme proposed here.

2 Quotient selection and divisor range

We now present the quotient-digit selection function by rounding and determine the required scaled-divisor range to achieve convergence.

To compute $Q = x/d$, with $1/2 \le x \le d < 1$, we use the recurrence

$$w[j+1] = rw[j] - q_{j+1}z \quad \text{with} \quad w[0] = Mx \quad (1)$$

where $w[j]$ is the residual after the j-th iteration, $r = 2^b$ is the radix, q_{j+1} is the new quotient digit, z is the scaled divisor and M is the scaling factor. That is,

$$z = Md \qquad \text{with} \qquad 1 - \alpha \le z \le 1 + \beta \quad (2)$$

To have a fast iteration we use a carry-save adder, although a similar development could be done for other redundant representations. We use a maximally redundant quotient-digit set, that is $|q_j| \le r - 1$. For this case, the convergence of the algorithm requires that the residual be bounded so that

$$|w[j]| \le z \quad (3)$$

We define

$$\Delta = rw[j] - q_{j+1} \quad (4)$$

so that (1) is transformed into

$$w[j+1] = \Delta + q_{j+1}(1 - z) \quad (5)$$

We now determine the range of Δ when the following quotient-digit selection by rounding is done:

$$q_{j+1} = round\ (\widehat{y}) \quad (6)$$

where $\widehat{y}$ is an estimate of $rw[j]$ obtained by truncating the carry-save representation up to t fractional bits. Then, from (4) and (6),

$$-1/2 \le \Delta \le 1/2 + 2^{-t} \quad (7)$$

We now use this range of Δ to determine the range of z that keeps $w[j+1]$ inside the required bound. First, from (5) and (7) we get

$$\begin{aligned} -1/2 - (r-1)|(1-z)| &\le w[j+1] \quad (8) \\ &\le 1/2 + 2^{-t} + (r-1)|(1-z)| \end{aligned}$$

Now applying the convergence requirement (3) we get (because of the term 2^{-t} in (8) the upper bound is the most critical):

$$\frac{1}{2} + 2^{-t} + (r-1)|(1-z)| \le z \quad (9)$$

In addition, it is necessary to constrain $w[j+1]$ so that a valid quotient digit ($\le r-1$) is obtained when using the rounding selection function of (6). For the upper bound this results in

$$rw[j+1] < r - \frac{1}{2} \quad (10)$$

Combining both requirements results in

$$\frac{1}{2} + 2^{-t} + (r-1)|(1-z)| < \min\left(z, 1 - \frac{1}{2r}\right) \quad (11)$$

To obtain bounds on z, we divide its range in three regions, as follows:

a) $z \ge 1$. Then, from (11) we get

$$1/2 + 2^{-t} + (r-1)(z-1) < 1 - \frac{1}{2r}$$

resulting in

$$1 \le z < 1 + \frac{1}{2r} - \frac{2^{-t}}{r-1} \quad (12)$$

b) $1 - 1/(2r) \le z < 1$. From (11) we get

$$1/2 + 2^{-t} + (r-1)(1-z) < 1 - \frac{1}{2r}$$

so

$$1 - \frac{1}{2r} + \frac{2^{-t}}{r-1} < z < 1 \quad (13)$$

c) $z < 1 - 1/(2r)$. From (11) we have

$$1/2 + 2^{-t} + (r-1)(1-z) < z$$

so

$$z > 1 - \frac{1}{2r} + \frac{2^{-t}}{r}$$

which is contradictory with the hypothesis.

Combining expressions (12) and (13) we get

$$1 - \frac{1}{2r} + \frac{2^{-t}}{r-1} < z < 1 + \frac{1}{2r} - \frac{2^{-t}}{r-1} \quad (14)$$

The smallest possible value of t in (14) is 2. A larger t increases the acceptable range of z, simplifying the scaling. However, since the variation of the range is small, it seems best to choose the smallest value of t. So we get

$$t = 2 \quad \Rightarrow \quad 1 - \frac{r-2}{4r(r-1)} < z < 1 + \frac{r-2}{4r(r-1)} \quad (15)$$

Observe that (15) is valid for radices $r > 2$.

3 Scaling

A scaling factor M is computed such that

$$z = Md \tag{16}$$

and z is in the range specified by (15). In addition, the scaled divisor and dividend are produced (i.e., Md and Mx). The scaling factor M can be viewed also as an approximation of the reciprocal of d.

Several techniques can be employed to implement the calculation of M as developed in [13]. In this paper we consider a linear interpolation approach (the L-approach) since, it provides the best results for the number of bits of interest [13]. In this approach, the scaling coefficient M is produced as follows:

1. Obtain the pair of coefficients $\hat{\gamma}_1$ and $\hat{\gamma}_2$ as a function of d_τ, the divisor truncated to its $\tau - th$ fractional bit.
2. Compute $M = -\hat{\gamma}_1 d_h + \hat{\gamma}_2$ (truncated to m bits), where d_h is the divisor truncated to its $h - th$ fractional bit.

The bit-lengths of the γ coefficients and the truncated versions of the divisor (d_τ, d_h) and of M are indicated in Fig. 1. More details are given in [7]. Md is computed in carry-assimilated form, whereas Mx is kept in carry-save form. These multiplications require a recoding to radix-4 of the carry-save representation of M (as described in the next section.) Moreover, the product Md requires a final assimilation of the result. Since Md belongs to the range specified by (15), it requires 1 integer plus $n + b + 4$ fractional bits. The same number of bits are required for Mx.

Scaling Unit. The hardware requirements for the scaling unit are (Fig. 1):

- Module for providing $\hat{\gamma}_1, \hat{\gamma}_2$ in 2's complement representation. This has $\lfloor b/2 \rfloor + 1$ inputs and $2b + 11$ outputs.
- One carry-save multiplier (the result is kept in carry-save form) to compute $-\hat{\gamma}_1 \times d_h$. The operands have $(b + 6)$ and $(b + 6)$ bits and it produces a result of $2b + 12$ bits. The multiplier incorporates a carry-save adder for computing $M = \{(-\hat{\gamma}_1 d_h) + \hat{\gamma}_2\}_{trunc-to-m}$.
- The hardware for recoding to radix-4 the carry-save representation of M.
- One carry-save multiplier of $(b+6) \times (n)$ bits, with a result of $(n + b + 5)$ bits, to compute the carry-save representations of Md and Mx; as indicated in the next section, for this operation we share the multiplier of $(b+6) \times (n+b+5)$ bits required for the iterations.
- One adder of $(n + b + 5)$ bits for computing the assimilation of Md coming out from the multiplier.

4 Overall Implementation

Besides the scaling unit, the overall scheme (Fig. 1) contains modules to execute the recurrence and on-the-fly conversion/rounding.

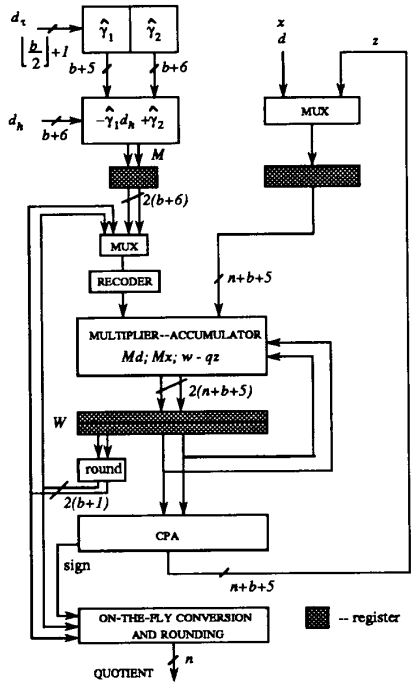

Figure 1: Block diagram of overall implementation

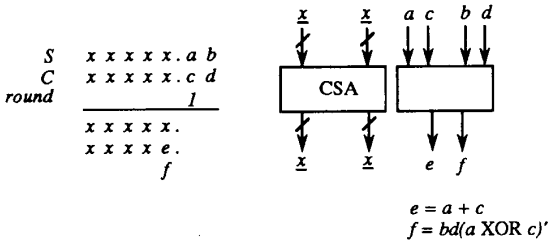

Figure 2: Rounding of the residual

The recurrence is implemented by the following modules (Fig. 1):

- Quotient-digit selection by rounding of the shifted residual (in carry-save form) truncated to the 2nd fractional bit. This is accomplished by a short carry-save adder and some additional logic (Fig. 2). The resulting quotient digit is in carry-save form with an additional bit in the least-significant position.

- Recoding the quotient digit to radix-4 representation (to be used as multiplier for the multiplication $q_{j+1}z$) is done in two steps:

1. Recode to signed-digit radix-4 with digit set (-2 to $+3$). This is possible since all groups of two bits of the digit q_{j+1} have values 0 to 6 (except least significant);
2. Recode this representation to signed digit radix-4 with digit set (-2 to $+2$).

For the first step, Fig. 3 shows the bits that are examined to obtain digit i of the recoded representation and the expression for the calculation of the radix-4 digit. For the second step it is sufficient to transform the values $+3$ and $+2$ into $4-1$ and $4-2$, respectively. An implementation that combines both steps is shown in Fig. 4.

- Multiplication of $-q_{j+1}$ by z and subtraction from $rw[j]$. For this, a multiplication of $(b+1) \times (n+b+6)$ bits is needed, with the inclusion of two terms to the adder array of the multiplier. This complete operation can be performed with the multiplier of $(b+6) \times (n+b+5)$ bits that is required for the scaling.
- On-the-fly conversion of the signed-digit quotient into a conventional representation, using the scheme presented in [4].

The post-correction for negative last residual and for rounding requires a sign detection of the last residual and the updating of the quotient. This updating is done as part of the on-the-fly conversion, as discussed in [4].

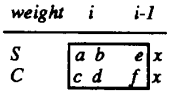

Figure 3: First step of recoding

5 Execution time

We now give an expression for the execution time of the division operation using the implementation of Fig. 1. We first determine the number of cycles and then the cycle time.

Number of cycles:

- One or more cycles for the computation of the scale factor M (we see in the next section that one cycle will suffice). The delay of this operation is denoted by t_M.

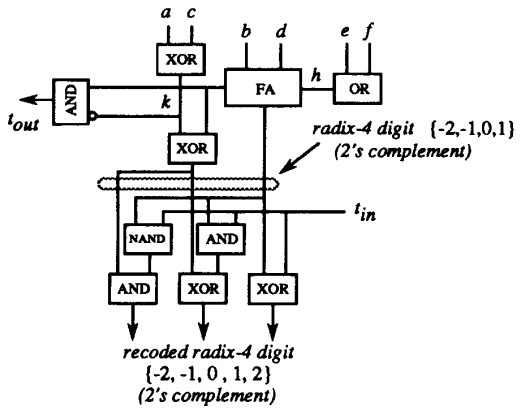

Figure 4: Implementation of recoding

- One cycle for the multiplication of d by M, producing the result in carry-save form. We denote this delay by t_{Md}.
- One cycle for the multiplication of x by M and (in parallel) for the assimilation of Md. The delay is denoted by t_{assimz}.
- $\lceil n/b \rceil$ cycles for the iterations. The corresponding delay is denoted t_{iter}.
- One cycle for the postcorrection and the rounding, with delay denoted by $t_{postcorr-round}$.

Consequently, assuming that one cycle is sufficient for the calculation of M, the number of cycles is

$$N_{cycles} = \left\lceil \frac{n}{b} \right\rceil + 4 \qquad (17)$$

where, as defined in Section 2, $r = 2^b$ is the radix.

Cycle time. The cycle time is determined by the maximum delay of the cycles described above. Since $t_{Md} < t_{iteration}$,

$$T_{cycle} = max(t_M, t_{assimz}, t_{iter}, t_{postcorr-round}) \qquad (18)$$

Moreover,

$$\begin{aligned} t_M &= t_\gamma + t_{MA,CS,b+6,b+6,1} + t_{reg} \\ t_{assimz} &= t_{ADD,n+b+5} + t_{reg} \\ t_{iter} &= t_{q-sel} + t_{MPX,2,1} + t_{recod} \\ &\quad + t_{MA,CS,b+6,n+b+5,0} + t_{reg} \\ t_{postcorr-round} &= t_{ADD,n+b+5} + t_{reg} \end{aligned}$$

where

- t_γ is the delay to compute $\widehat{\gamma_1}$ and $\widehat{\gamma_2}$;

- $t_{MA,CS,x,y,z}$ is the delay of an $x \times y$ carry-save multiplication with accumulation of z terms;
- $t_{ADD,x}$ is the delay of an addition of x bits;
- t_{q-sel} is the delay of the quotient-digit selection by rounding;
- $t_{MPX,2,1}$ is the delay of a 2-to-1 multiplexer;
- t_{recod} is the delay of recoding the carry-save representation of the quotient digit;
- t_{reg} is the delay to load a register.
- Finally, the delay for $t_{postcorr-round}$ corresponds to the time to detect the sign of the last residual, since the actual correction and rounding are performed as part of the on-the fly conversion [4]. Therefore, the delay can be approximated by $t_{ADD,n+b+5}$.

5.1 Execution time in full-adder delays

To determine the dependence of the execution time on the radix and to compare with other schemes, we define the execution time in terms of full-adder delays (t_{FA}). This unit is chosen because many of the components are formed of full adders and because it is relatively simple to express the delay of other components using this unit. Although this makes the evaluation and comparisons relatively independent of the technology, the results are rough because they depend on the accuracy of the assumptions made.

To determine the delay of the components, assumptions have to be made about their implementation; we have used assumptions that seem reasonable, but some alternatives would also be possible. In the cases in which there is no direct correspondence with the delay of a full adder, we have used the correspondence for a particular technology. Specifically, we utilized the family of standard cells from the ES2-ECPD10 library [8]. For instance, because of this we have estimated the delay of loading a register to be 1.5 t_{FA}. The delays of the components measured in this unit are as follows (more details in [7]):

- Adder assimilating two x-bit values

$$t_{ADD,x} = \lceil log_2(x) \rceil \cdot t_{FA} \tag{19}$$

- Carry-save multiplier of x by y bits with accumulation of z operands

$$t_{MA,CS,x,y,z} = 2 \left\lceil log_2 \left(\frac{min(x,y)}{2} + z \right) - 1 \right\rceil \cdot t_{FA} \tag{20}$$

Table 1: Time and area for best γ modules

b	9	11	14	18
Delay (t_{FA})	2	2	4	4
Area (A_{FA})	80	180	800	4400

Table 2: Time and area for proposed implementation

b	9	11	14	18
Cycle time (t_{FA})	10	12	12	12
No. cycles	10	9	8	7
Exec. time (t_{FA})	100	108	96	84
MULT (A_{FA})	570	700	900	1200
γ (A_{FA})	80	180	800	4400
MULT + γ (A_{FA})	650	880	1700	5600

The delay of a carry-save multiplier without accumulation is derived from (20) by making $z = 0$, whereas the delay of a multiplier with assimilation of the result is obtained by combining (19) and (20).

- Recoding (Fig. 4) is about 2 t_{FA}.
- Quotient-digit selection by rounding is 1 t_{FA} (Fig. 2).
- Register loading is assumed to be $1.5t_{FA}$.

The only component whose delay cannot be readily described in t_{FA}'s is the module to determine $\widehat{\gamma_1}$ and $\widehat{\gamma_2}$. We have explored several alternatives and have determined their delay and area using the family of standard cells from the ES2-ECPD10 library as discussed in [7]. In terms of the delay of a full adder (t_{FA}) and of its area (A_{FA}) the best alternatives for different values of b are summarized in Table 1.

5.1.1 Cycle time, execution time, and area

Using the delay of the components in terms of t_{FA} we now determine the cycle time, as expressed by (18). It can be computed that in all cases, the critical time corresponds to the iteration time. In Table 2 we show the cycle time and also the execution time for $n = 54$ and the corresponding area of the multipliers (only adders) and of the γ-module.

6 Other Related Implementations

For comparison purposes, we now make a rough evaluation of the execution time of other implementations for very-high radix dividers. As a reference, we also include the frequently used radix-16 divider with overlapped radix-4 stages. To make uniform evaluations, we use as a unit the delay of a full adder, as done in Section 5. The evaluations are rough since data routing delays are not included nor do we perform some technology-dependent optimizations. As done in Section 5, for the very-high radix schemes we also give the area, in full-adder units, of the multipliers and the corresponding γ modules. We consider the following schemes:

Radix-16 unit with overlapped radix-4 stages. A unit of this type was introduced in [15] and a recent implementation presented by Williams and Horowitz in [19]. We compare with the latter. Although the implementation reported is self timed, we evaluate a synchronous variation, to be able to compare similar design styles. We use the implementation with radix-4 stages since our calculations indicate that it is significantly faster than the radix-2 one. As indicated in [19], the delay per iteration can be approximated by

$$t_{WH} = 1/2(P + Q + R + S) \tag{21}$$

where $P = t_{CSA} + t_{ADD,7}$, $Q = t_{MPX,5,1} + t_{QSEL}$, $R = t_{CSA}$, and $S = t_{driver} + t_{MPX,5,1}$.

To estimate the delay of the radix-4 quotient selection we have implemented the module using the ES2-ECPD10 family of standard cells and obtained a delay equivalent to $2.5t_{FA}$. Introducing the other values according to the assumptions made in Section 5, we get

$$t_{WH} \approx 5t_{FA}$$

For a 54-bit mantissa, the execution time is

$$T_{WH} = 27t_{WH} + t_{postcor\ \&\ round} \approx 140t_{FA} \tag{22}$$

where $t_{postcor\ \&\ round} = 6t_{FA}$, that is, the delay of one addition of two 54-bit values (for the sign detection, because we have assumed before that the actual correction and rounding is done as part of the on-the-fly conversion).

Divider in Weitek W4164 and W4364. In [18] no detail is given concerning the algorithm. However, since a 64 × 64 multiplier is used, it should be of the successive-approximation type [9]. The data sheet indicates that the double-precision division takes seven cycles, which include the initial approximation of the reciprocal, the successive approximations, and the final rounding. The cycle time is probably determined by a multiplication-accumulation (including recoding and register loading), so that we estimate the cycle delay to be

$$\begin{aligned} t_{weitek} &= t_{recod} + t_{MA,CA,64,64,1} + t_{reg} \\ &= 1 + 17 + 1.5 = 19.5t_{FA} \end{aligned} \tag{23}$$

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

$$T_{WEITEK} = 7 \cdot 19.5 \approx 135t_{FA} \tag{24}$$

The basic hardware requirements of this architecture are

- one carry-assimilated multiplier/accumulator of 64 by 64 bits;
- the module required for the initial approximation.

Matula's scheme. In [11] and [12] a radix-2^{17} division unit is described, which is based on multiplying the residual by a short reciprocal of the divisor so that digit selection can be done by truncation. 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.

To compare with the method we are proposing, we determine the execution time for 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 is determined as follows:

- One cycle to determine the short reciprocal.
- Two cycles per iteration: one to multiply the residual by the short reciprocal and one to multiply the divisor by the quotient digit and accumulate the new residual.
- One cycle for postcorrection and rounding.

Consequently, for 54-bit result the number of cycles is

$$N_{cycles} = 2 + 2\left\lceil \frac{54}{b} \right\rceil \tag{25}$$

The cycle time is determined by the multiplication (including the multiplexer for selecting the multiplier and the multiplier recoding) and the register loading. That is,

$$\begin{aligned} T_{cycle} &= t_{MPX,2,1} + t_{driver} + t_{recod} \\ &\quad + t_{MA,CS,b+6,n,0} + t_{ADD,n+b+6} + t_{reg} \end{aligned} \tag{26}$$

Table 3: Time and area for Matula's scheme

b	Cycle time (t_{FA})	Number of cycles	Execution time (t_{FA})	Area (A_{FA})
9	15	14	210	550
11	17	12	200	750
14	17	10	170	1500
18	17	8	135	5400

With the assumptions of Section 5 we get the execution times and areas (for the multiplier plus the γ module) shown in Table 3. The basic hardware requirements are

- One γ module.
- One carry-assimilated multiplier of $(b+6)$ by $(b+6)$ (to calculate the short reciprocal).
- One carry-assimilated multiplier/accumulator of $(b+6)$ by 54 bits.

Wong and Flynn scheme. In [20] two division algorithms are presented, namely a *basic* and an *advanced method*, which differ only in the way the short reciprocal (for scaling) is evaluated. To make consistent comparisons with the proposed scheme, we utilize the same method for computing the scaling factor. The authors present three different choices of radix; moreover, they also study the possibility of performing carry-save multiplications. The number of cycles is determined as follows (in this description we use the notation given in [20]):

- One cycle to determine the short reciprocal $1/Y_h$. At the same time perform $X_h \times Y$.
- One cycle to calculate $Y \times 1/Y_h$. At the same time the first iteration is performed.
- $\lceil n/b - 1 \rceil$ cycles for the iterations (we use here b instead of the m of [20]($b = m - 2$)).
- One cycle to assimilate the quotient (because of the overlapping of the quotient segments produced by consecutive iterations, on-the-fly conversion is not practical).
- One cycle for postcorrection and rounding.

The cycle time is determined by the multiplication with assimilation required in cycle 1. That is

$$T_{cycle} = t_{MPX,2,1} + t_{driver} + t_{recod} + t_{MA,CS,b+6,n,0} + t_{ADD,n+b+5} + t_{reg}$$

This results in the values given in Table 4. The main modules required are:

- One γ module and one multiplier with assimilation of $(b+6)$ by $(b+6)$ bits.

Table 4: Time and area for Wong & Flynn's scheme

b	Cycle time (t_{FA})	Number of cycles	Execution time (t_{FA})	Area (A_{FA})
9	15	9	135	1150
11	17	8	135	1450
14	17	7	120	2450
18	17	6	100	6600

Table 5: Summary of time/area characteristics

W & H	140/ -			
Weitek	135/ -			
b	9	11	14	18
Matula	210/550	200/750	170/1500	135/5400
W & F	135/1150	135/1450	120/2450	100/6600
Our	100/650	110/880	95/1700	85/5600

- One multiplier with assimilation of $(b+6)$ by n bits, to compute X_h times Y and $1/Y_h$ times Y.
- One carry-assimilated adder of $(b+4)$ bits to compute X_h.
- One carry-save multiplier of $(b+6)$ by $(n+b+5)$ bits with single accumulation, to perform the iterations.
- One carry-save multiplier of $(b+4)$ by $(b+6)$ bits to compute the quotient segments.
- One 4-2 carry-save adder to accumulate the quotient.
- One carry-assimilated adder of $(56+b)$ bits for assimilation of the quotient and one 54-bit sign detection network.

Table 5 summarizes the characteristics of the schemes we have compared.

7 Conclusion

We described a very-high-radix division unit based on the standard division recurrence. We scale the divisor (and the dividend) to a range close to one so that the quotient-digit selection can be performed by rounding a truncated residual.

To determine a good choice for the radix, we performed an evaluation of the execution time. This required the modeling of the delays of the components. We determined a range of possible execution times and of area requirements. The conclusions obtained depend on the assumptions made about the implementation of the components; however, expressions are given to allow the evaluation using other assumptions.

We have used the same modeling assumptions to perform a rough evaluation of the delay of other very-high radix implementations and of a radix-16 implementation with overlapped radix-4 stages. We found that the fastest divider is about 2.5 times faster than the slowest and that the implementation presented here is the fastest by about 20%. The implementation presented here with $b = 9$ seems to be a good tradeoff between speed and area.

The divider is considered here as one unit. However, it consists of two separate components: the computation of the scaling factor and the rest. Since the former depends only on the divisor, if the divisor is available before the dividend, it is possible to perform this scaling factor calculation in advance, resulting in a reduction of the division time. An additional reduction can be achieved if several divisions use the same divisor value, because the whole divisor scaling does not have to be repeated. Finally, the unit can be pipelined with two stages, increasing in this way the throughput.

Acknowledgment. Part of this work was performed while Tomas Lang was with UPC, Barcelona, Spain. We thank Jordi Cortadella for providing information that helped us in the evaluation.

References

[1] M. D. Ercegovac, "A General Hardware-Oriented Method for Evaluation of Functions and Computations in a Digital Computer," Dept. of Computer Science, TR-750, U. of Illinois at Urbana-Champaign, 1975.

[2] M. D. Ercegovac and T. Lang, "A Division Algorithm with Prediction of Quotient Digits," Proc. 7th IEEE Symp. on Comp. Arithmetic, 1985, pp.51-56.

[3] M. D. Ercegovac, "Radix-16 Evaluation of Certain Elementary Functions," IEEE Trans. Comput., Vol.C-22, No.6, June 1973, pp.561–566.

[4] M. D. Ercegovac and T. Lang, "On-the-Fly Rounding," IEEE Trans. Comput., Vol. 41, No.12, Dec. 1992, pp. 1497–1503.

[5] M. D. Ercegovac and T. Lang, "Simple Radix-4 Division with Operands Scaling", IEEE Trans. Comput., Vol. C-39, No.9, Sept. 1990, pp. 1204-1207.

[6] M. D. Ercegovac and T. Lang, *Digit-Recurrence Algorithms and Implementations for Division and Square Root*, to be published, Kluwer Academic Publishers, 1993.

[7] M. D. Ercegovac, T. Lang, and P. Montuschi, "Very-High Radix Division with Selection by Rounding and Prescaling," Tech. Report, Elect. and Comp. Eng. Dept., University of California at Irvine, 1993.

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

[9] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*, John Wiley and Sons, New York, 1978.

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

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

[12] D.W. Matula, "Short Reciprocal Division," submitted for publication, 1992.

[13] P. Montuschi, T. Lang, and M. D. Ercegovac, "Prescaling Algorithms for Very-High Radix Division" I.R. DAI/ARC 6-92

[14] A. Svoboda, "An Algorithm for Division," Information Proc. Machines, 1963, No.9, pp.25–34.

[15] G. S. Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Selection Stages," Proc. 7th Symp. on Comp. Arithmetic, pp. 64–71, June 1985.

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

[17] S. Waser and M. J. Flynn, "Introduction to Arithmetic for Digital Systems Designers," Holt, Rinehart, and Winston, New York, 1982.

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

[19] T.E. Williams and M.A. Horowitz, "A 160ns 54-bit CMOS Division Implementation Using Self-Timing and Symmetrically Overlapped SRT Stages," Proc. 10th Symp. on Comp. Arithmetic, 1991, pp. 210–217.

[20] 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.