High-Radix Division and Square-Root with Speculation

Jordi Cortadella and Tomás Lang

Abstract— **The speed of high-radix digit-recurrence dividers and square-root units is mainly determined by the complexity of the result-digit selection. We present a scheme in which a simpler function speculates the result digit, and, when this speculation is incorrect, a rollback or a partial advance is performed. This results in operations with a shorter cycle time and a variable number of cycles. The scheme can be used in separate division and square-root units, or in a combined one. Several designs were realized and compared in terms of execution time and area. The fastest unit considered is a radix-512 divider with a partial advance of six bits.**

Index Terms— **Digital arithmetic, digit recurrence, division, square-root, variable-time operation**

I. Introduction

MOST implementations for the division and square-root operations involve a recurrence in which one digit of the result is produced per iteration [1], [7], [12]. Consequently, to reduce the number of iterations, it is convenient to use a higher radix for the result digit. However, as the radix increases, the added complexity of the result-digit selection function increases the iteration delay and eliminates the advantage. Because of this, implementations have used radix-2 and radix-4 stages and higher radices are obtained by unfolding these stages [9], [13]. Moreover, reductions in time have been obtained by overlapping these unfolded stages [13], [14]. To have higher radix implementations, it is possible to use an arithmetic function for the result-digit selection [11]. Specifically, the result digit is obtained by multiplying an estimate of the residual by an estimate of the reciprocal of the divisor (or the square-root of the operand) and rounding the result. However, the calculation of the reciprocal approximation and the multiplication and rounding limits the speedup achieved.

For division, one way that has been proposed to reduce the complexity of the selection function is to prescale the divisor (and the dividend) and then perform the quotient-digit selection by rounding (see [7] for references). However, this method is not easily extensible to square-roots [10].

In this paper, we present and evaluate the idea of speculating the result digit using a function that is simpler (i.e., has smaller delay) than the result-digit selection function. This speculated digit is then used to continue with the algorithm. Another function determines whether the speculation is incorrect, and, in that case, the algorithm rolls back, the digit is corrected, and the process continues from there. In a variation of this scheme, we allow a partial advance of fewer bits than a full digit when an incorrect speculation is performed.

Because of the possible rollbacks and partial advances, the execution time is variable. Consequently, a unit of this type is suitable when the rest of the system can make use of this variable time. In particular, if the unit forms part of a general purpose processor with several functional units, it is necessary to have hardware support for control of dependencies. In the evaluation, we determine the average execution time assuming a uniform distribution of operands.

The effectiveness of an implementation depends on a variety of factors, such as the time and cost of the speculation function, the probability of correct speculation, the time and cost of error detection, and the time for correction.

We develop the method and evaluate alternative possibilities. We present some examples of implementations and compare with the implementation of conventional algorithms using the same technological constraints.

For clarity in the exposition, we first consider division and then extend the results to square-root and to a combined implementation.

A. Division Algorithm and Notation

We now review very briefly the well-known division algorithm, mainly to establish the notation used. We use the following standard recurrence:

$$w[j+1] = rw[j] - q_{j+1}d \qquad w[0] = x,$$

where $w[j]$ is the residual after the jth iteration, r is the radix, q_{j+1} is the new quotient digit, d is the divisor, and x is the dividend. We assume that x, d, and q are normalized fractions.

To have a fast iteration, a redundant adder is used. In this paper, we use a carry-save adder, although a similar development could be done for other redundant representations of $w[j]$.

The quotient digit is a signed digit $|q_{j+1}| \leq a$, with redundancy factor $\rho = a/(r-1)$. This requires that $w[0] \leq \rho d$, which is obtained by shifting the dividend. Moreover, for this case, the convergence of the algorithm requires the residual to be bounded so that

$$|w[j]| \leq \rho d. \tag{1}$$

The quotient digit q_{j+1} is determined by a quotient-digit selection function. This function depends on an estimate of

Manuscript received October 19, 1993; revised March 9, 1994. This work was supported in part by the Ministry of Education of Spain under CICYT, TIC 91-1036.

J. Cortadella is with the Department of Computer Architecture, Polytechnic University of Catalonia, Barcelona 08071 Spain; e-mail: jordic@ac.upc.es.

T. Lang is with the Department of Electrical and Computer Engineering, University of California, Irvine, CA 92717 USA.

IEEE Log Number 9403089.

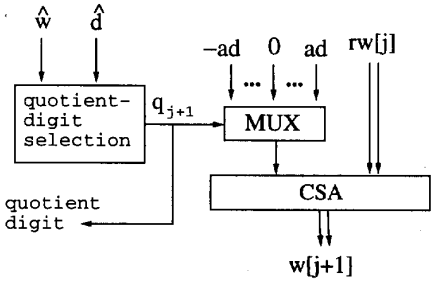


Fig. 1. Iteration step of digit-recurrence division.

TABLE I
NUMBER OF BITS REQUIRED FOR QUOTIENT-DIGIT SELECTION

radix	2	4	8	16
a	1	2	7	10
ρ	1	2/3	1	2/3
# bits divisor	0	3	3	6
# bits residual	4	7	8	10

the residual and on an estimate of the divisor; that is,

$$q_{j+1} = \text{sel}(\hat{w}, \hat{d}). \tag{2}$$

The conceptual implementation of one iteration step is shown in Fig. 1.

Two ways of implementing the quotient-digit selection have been proposed. In the most common, a table is constructed specifying for each value of $\hat{w}$ and $\hat{d}$ a quotient-digit value that satisfies the bound. The estimates are obtained by truncating the corresponding values. The number of bits of these estimates required for correct selection increases with the radix as shown in Table I. This lengthens the delay of the implementation, so that practical implementations are limited to radix 2, 4, and 8 stages.

A second and recent approach is practical for higher radices. In it, the quotient digit is obtained by multiplying an estimate of the residual by an approximation of the reciprocal of the divisor and rounding the result [11]. That is,

$$q_{j+1} = \text{round}\big(\hat{z}(r\hat{w})\big), \tag{3}$$

where $\hat{z}$ is an estimate of $1/d$. Although this can be implemented for very high radices, the delay of the multiplication, the rounding, and the decoding of the quotient digit in each iteration may result in a limited speedup with respect to low-radix designs.

II. Scheme for Division With Speculation of Quotient Digits

We now describe the basic scheme (without partial advance) and then introduce the partial advance.

A. Basic Scheme

The basic scheme is shown in Fig. 2. As mentioned in the introduction, the quotient digit is speculated (how this is done is described in the next section) and used in the recurrence to produce the speculated next residual as follows (speculation cycle):

$$w^s[j+1] = rw[j] - q^s_{j+1}d.$$

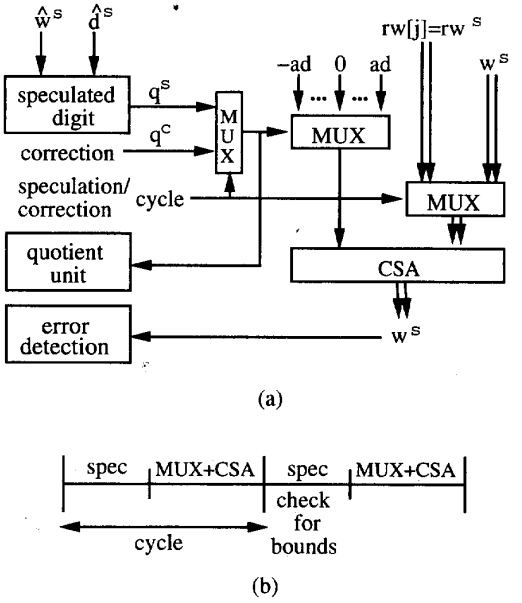


Fig. 2. Basic scheme. (a) Block diagram. (b) Timing diagram.

At the end of this speculation cycle, we decide whether the speculated digit is correct by determining whether the residual is inside the allowed bounds (see expression 1). If it is inside the bounds, then

$$w[j+1] = w^s[j+1],$$

and the next iteration is performed. On the other hand, if the residual is out of bounds, the quotient digit and the residual have to be corrected. A correction cycle is performed as follows:

$$w[j+1] = w^s[j+1] - q^c_{j+1}d.$$

In this case, the digit q^c_{j+1} has the same weight as the digit obtained in the speculation cycle, and thus the correct digit is $q_{j+1} = q^s_{j+1} + q^c_{j+1}$. The amount of correction performed, q^c_{j+1}, is discussed in the next section. Consequently, when there is an error (incorrect speculation), more than one cycle is needed to obtain the correct quotient digit and the correct next residual. As indicated in Fig. 2(b), the check for the bound is overlapped with the speculation of the next quotient digit.

In Fig. 3, we show the timing diagram of an example division, and, in Fig. 4, the quotient digits produced. As can be seen, the cycle time for the division with digit speculation is smaller than for the conventional case, because the computation of the speculated digit is faster than the computation of the correct digit. In the example, the first two speculated digits are correct; the third digit is incorrect, producing a $w^s[3]$, which is out of bounds. This results in a correction of q_3 by $+1$ and the corresponding correction of $w^s[3]$. This correction is sufficient, as indicated by the fact that the corrected residual is inside the bounds. Then the next speculated digit is correct, and, after that, another error is detected and corrected. In this case, two cycles are required for the correction, because the

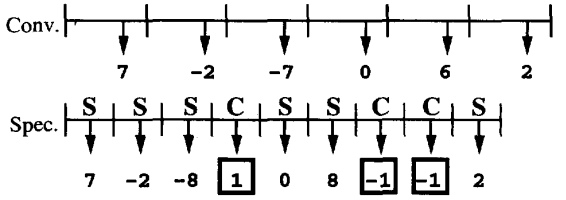


Fig. 3. Timing diagram for radix-16 conventional division (conv.) and quotient-digit speculation (spec). S: Speculation cycle, C: Correction cycle.

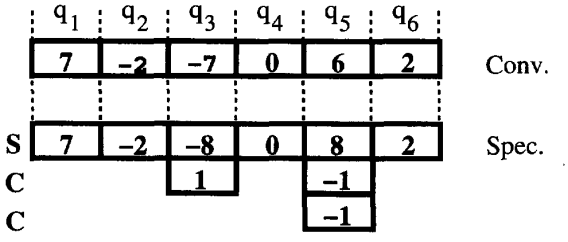


Fig. 4. Quotient-digit generation for conventional division and quotient-digit speculation.

residual obtained after the first cycle of correction is still out of bounds.

B. Scheme with Partial Advance

In the basic scheme presented, two situations can occur: Either we advance one digit of the quotient (when the speculation is correct), or we use the next cycle to correct the digit (no advance). So, at least one complete additional cycle is required to correct an incorrect speculation, even if the error is very small. The corresponding delay is reduced by the scheme with partial advance. In this scheme, when the error is small, a third situation is allowed, namely, the advance of a number of bits that is less than a whole digit. We call this *partial advance*.[1]

The amount of advance is selected so that the next residual is bounded. That is, it is possible to advance $\log_2 p$ bits if

$$|pw^s| \leq r\rho d.$$

The partial advance iteration is

$$w^s[j+1] = pw^s[j] - q^p_{j+1}d,$$

where q^p_{j+1} is the quotient digit speculated for partial advance.

With partial advance, there are three possible situations, as follows:

- Full advance if $|w^s| \leq \rho d$
- Partial advance of $\log_2 p$ bits if $\rho d < |w^s| \leq \frac{r}{p}\rho d$. In this case, as shown in Fig. 6(b), the digit q^p_{j+1} has an overlap of $\log_2 \frac{r}{p}$ bits with the previous digit.
- No advance if $|w^s| > \frac{r}{p}\rho d$.

In principle, it is possible to advance a variable number of bits, depending on the amount of the error. However, to limit the complexity of the implementation, we consider only cases with a partial advance of a fixed number of bits. A block diagram for this scheme is shown in Fig. 5. Note that two different speculations are performed, depending on whether

[1] A variation on this idea was suggested to us by P. Montuschi.

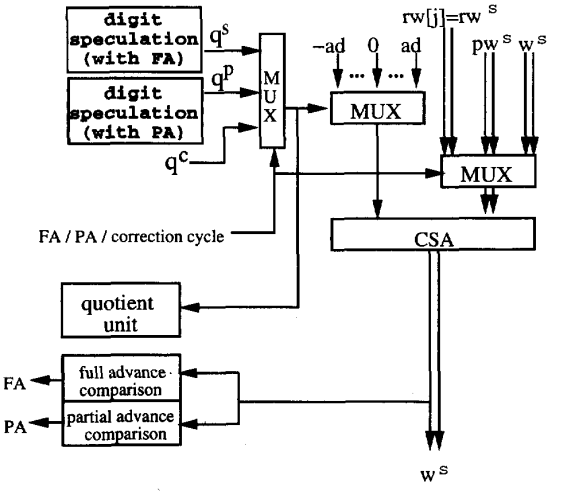
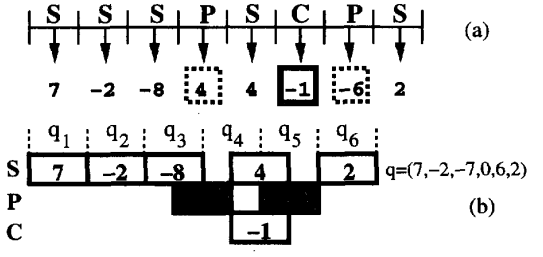


Fig. 5. Scheme with partial advance.



S: speculation cycle, C: correction cycle, P: partial advance

Fig. 6. Quotient-digit generation for radix-16 with partial advance (2 bits). (a) Conventional. (b) Partial advance.

there is a full advance ($\log_2 r$ bits) or a partial advance ($\log_2 p$ bits). The timing diagram of Fig. 6(a) and the quotient digit computations of Fig. 6(b) show an example with partial advance. The first two digits are correct, whereas the third is incorrect. Consequently, it is not possible to advance a whole digit. Two possibilities exist: No advance or partial advance. In this case, the error is sufficiently small so that a partial advance can be made. (We consider a partial advance of two bits, $\log_2 p = 2$.) The new speculated digit is 4, which overlaps with the incorrect digit. This new digit is correct. Then a new speculation ($q = 4$) is incorrect, and in this case, no advance is possible, because the error is too large. A correction cycle is required, and, after that, with the residual still out of bounds, another partial advance can be performed ($q = -6$). Finally, the last digit is correct.

III. Speculation of Quotient Digit

As indicated in the previous section, the idea is to reduce the delay of the quotient-digit selection by using a simpler function that gives a correct value with high probability. For our purposes, the complexity of the function is related to its delay, using a particular design style and technological constraints. As indicated in Fig. 7, the average execution time per quotient digit is equal to the product of the delay of a cycle by the average number of cycles per digit. More-

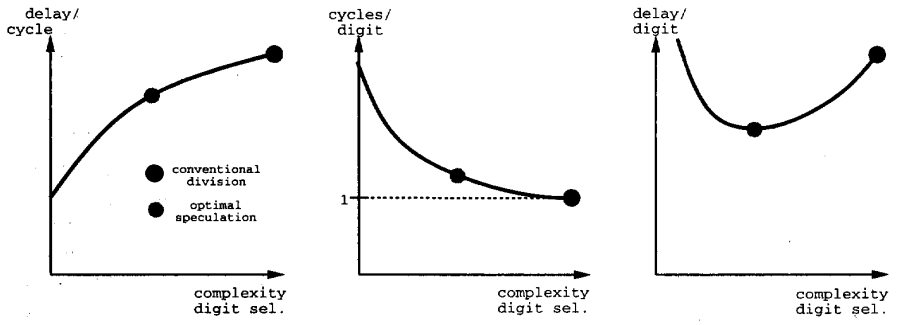


Fig. 7. Delay/digit vs. complexity of q-digit selection.

over, the first component increases with the complexity of the quotient-digit speculation function, whereas the second component decreases, because fewer no-advance and partial advance cycles are required. The speculation function should be selected to produce an execution time close to the minimum of Fig. 7.

The process of finding the simplest (lowest delay) speculation function for a given probability of correct speculation is a complex problem. It essentially consists of finding a function that has the same value as a correct quotient-digit selection function for a given fraction of the argument values, produces a small error when incorrect, and has minimum complexity. Since no general strategy exists to obtain the optimal function, we have explored the following alternatives:

1) Use a function with fewer variables, that is, a function with fewer bits of the residual estimate and of the divisor estimate.
2) Reduce the number of output values; that is, some quotient values are not generated. When one of these nongenerated values is required, an incorrect value is generated and correction action (no advance or partial advance) is required.
3) Instead of the general function (2), use an approximation of the arithmetic function of expression (3).

We now discuss these approaches further.

A. Reducing the Number of Variables

In this case, we use a speculation function described as

$$q_{j+1}^{s} = \text{spec}(\hat{w}^{s}, \hat{d}^{s}),$$

where $\hat{w}^s$ and $\hat{d}^s$ have fewer bits than the $\hat{w}$ and $\hat{d}$ of expression (2), respectively. This reduction in number of variables should reduce the delay in the implementation. Moreover, if the least-significant bits of $\hat{w}$ and $\hat{d}$ are not included, the probability of correct speculation should be high and the value of the error should be small. The question is how to find such a speculation function, combining a low delay and a reasonable probability of speculating the correct value. There are many parameters in this choice of function, including characteristics of the implementation. Therefore, it is not practical to search for an optimal solution, and we have to be content with finding one that is satisfactory. To simplify the process, we perform it in two stages; namely, we first determine, for various numbers of bits of the estimates, the functions that give the highest probability of correct selection, and then we use these functions in implementations.

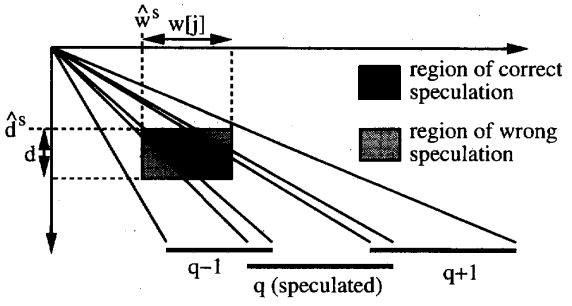


Fig. 8. P–D diagram to determine the probability of correct speculation.

For a given number of bits of the estimates, the function that produces the highest probability of correct speculation can be determined theoretically or empirically. A possible theoretical approach follows (see Fig. 8).

- For a particular value of $\hat{w}^s$, determine the range of values of $w[j]$. This defines a vertical strip in Fig. 8.
- For a particular value of $\hat{d}^s$, determine the range of values of d. This defines a horizontal strip in Fig. 8.
- Determine the areas corresponding to the intersection of the rectangle formed by the two strips above, and the selection intervals for different values of q_{j+1}. Select for q_{j+1}^s the value that corresponds to the largest area.

This approach has the disadvantage that it assumes that all values of the residual are equally likely. To avoid such an assumption, which might be even less true with speculation than with conventional division, we determined the speculation function empirically.

A possible empirical determination of the speculation function is to perform divisions using a conventional algorithm. Then, for each pair of values $(\hat{w}^s, \hat{d}^s)$, select as q_{j+1}^s the value that occurs most frequently. However, this process is not accurate, because the residual values obtained when executing a conventional algorithm are not the same as those produced when speculation is used. To eliminate the corresponding error, we determine the speculation function as follows.

- Build a speculation table (Table-spec) that for each pair of values of $(\hat{w}^s, \hat{d}^s)$, has as an entry the speculated quotient digit. Initialize this table with any value, for example, all 0.

TABLE II
PROBABILITY OF SUCCESS OF QUOTIENT-DIGIT SPECULATION

radix	4	8	16
a	2	5	12
# bits $\hat{w}^s$ (s-c)	4-4	5-4	(6-5)
# bits $\hat{d}^s$	0	1	1
prob. of success	0.91	0.86	0.77

- Perform several iterations of the following process until a stable speculation table is obtained.
  a) Perform simulations of divisions in which the dividend and the divisor are selected at random.
  b) In these divisions, use as quotient digit the value obtained from Table-spec.
  c) Build a matrix with a row for each pair $(\hat{w}^s, \hat{d}^s)$ and a column for each value of q^s_{j+1}.
  d) For each residual and divisor obtained during the simulation, determine the possible correct values of the quotient digit and increment the corresponding entries in the matrix.
  e) At the end of the iteration, for each row of the matrix, select for quotient-digit speculation the value that has the maximum entry in the row. Update the speculation table.

For each simulation, 10^5 divisions were executed; for each division, the dividend and divisor were randomly generated, and a 54-bit quotient was calculated. Only few iterations (between 3 and 5) were required to obtain a stable speculation function. The technique of multiple independent repetitions was used to obtain results with a confidence level higher than 0.98.

Table II shows probabilities of success for the best function for several cases. Since $\hat{w}^s$ has a *carry-save* representation, different number of bits can be used from the *sum* and from the *carry*. When sum and carry bits are expressed in parenthesis, (s–c), the corresponding bits are first assimilated and then used by the speculation function. Since the probability of success is relatively high and the number of bits significantly smaller than those of Table I, the approach looks promising.

B. Reduction in Number of Outputs

The quotient-digit selection function can be simplified if not all output values are allowed. Moreover, if the values that are generated are selected in a suitable manner, it is possible to reduce the number of adders required to produce $q^s_{j+1}d$ and in this way reduce the iteration delay. Actually, this second reason motivated us to use this approach (combined with the other approaches). As an illustration, consider a radix-8 divider with $a = 7$. In a conventional algorithm, the generation of $q_{j+1}d$ requires two adders, because it is possible to decompose q_{j+1} as $q_{j+1} = q1 + q2$ with $q1 \in \{-8, -4, 0, 4, 8\}$ and $q2 \in \{-2, -1, 0, 1, 2\}$. To reduce the number of adders to one, it is possible, for example, to use $a = 5$ and generate only the values $q^s_{j+1} \in \{-4, -2, -1, 0, 1, 2, 4\}$. In this case, when the correct quotient-digit value has to be either ± 3 or ± 5, an incorrect value is speculated and a correction is performed.

C. Approximation of Arithmetic Function

Expression (3) gives an arithmetic function for the quotient-digit selection. As indicated in the previous section, this arithmetic structure allows the implementation for high radices. However, the arithmetic function still has a large delay (produced by the multiplication, the rounding, and the decoding of the quotient-digit values). Consequently, we can also use in this case the speculation approach to simplify the function and reduce the delay. The simplification can be applied to the various components of the quotient-digit selection, as well as to its integration into the whole recurrence. We show an example implementation in Section VI-B.

IV. Error Detection and Correction

A. Detection

Since the speculation is not always correct, it is necessary to detect the error. Two approaches for this seem possible.

1) Compute the correct value of the quotient digit and compare with the speculation. This scheme is not convenient, because for high radices, the exact quotient-digit selection is complex and slow, and because there are cases in which more than one value is correct.
2) Determine whether the next residual is within the allowed bounds.

We chose this second approach. We need to assure that the speculation is accepted only if

$$|w^s[j+1]| \leq \rho d. \tag{4}$$

To have a fast comparison, this determination has to be performed using a truncation of $w^s[j+1]$ and d. Let us call $\hat{w}^c$ and $\hat{d}^c$ these truncated values. We now determine the minimum number of bits that these estimates require. Let us call f the number of fractional bits of each estimate. Since a *carry-save* representation is used for w, $\hat{w}^c$ is calculated as the sum of the most significant bits of the two bit vectors representing w, and therefore we have

$$w \in [\hat{w}^c, \hat{w}^c + 2^{-f+1}).$$

On the other hand, d is nonredundant, so we have

$$d \in [\hat{d}^c, \hat{d}^c + 2^{-f}).$$

Consequently, relation (4) is satisfied if

$$-\rho\hat{d}^c \leq \hat{w}^c \leq \rho\hat{d}^c - 2^{-f+1}. \tag{5}$$

Therefore, these are the comparisons that have to be performed. Since w^c is truncated, only the truncated value of $\rho\hat{d}^c$ is required for the implementation of the comparisons.

In addition, the continuity condition has to be preserved. That is, for any value of the residual inside the bounds, at least one choice of quotient-digit value should be valid. This requires that the difference between the upper and lower bound of the residual has to be at least equal to d. For our case,

$$\rho\hat{d}^c - 2^{-f+1} + \rho\hat{d}^c \geq d,$$

and since $d < \hat{d}^c + 2^{-f}$, it is sufficient that

$$2\rho\hat{d}^c - 2^{-f+1} \geq \hat{d}^c + 2^{-f},$$

TABLE III
NUMBER OF BITS FOR COMPARISON

r	a	ρ	f
4	2	2/3	5
4	3	1	3
8	5	5/7	4
16	10	2/3	5

e	i
0	1
1	2
2...3	3
4...7	4

and we get

$$(2\rho - 1)\hat{d}^c \geq 3 \times 2^{-f}.$$

Since $\hat{d}^c \geq 1/2$, it is sufficient that

$$2^{-f} \leq \frac{1}{3}\left(\rho - \frac{1}{2}\right).$$

We now determine the number of integer bits of the estimates. Since intermediate residuals produced by speculative division might have values larger than ρd, additional integer bits are required. Let us call e the maximum error produced by a speculation; that is, e is the maximum difference between a speculated value and a correct quotient digit. Moreover, call $\overline{M}$, and call $\underline{M}$ the highest and lowest values of $w^s[j+1]$ obtainable when the recurrence is performed with a speculation having the maximum error. Since a correct residual is $w^s[j+1] \leq \rho d$, then

$$\overline{M} \leq \rho d + ed,$$

and because $d < 1$,

$$\overline{M} < \rho + e$$

Similarly,

$$\underline{M} > -(\rho + e),$$

and thus the number of integer bits (i) required to represent wrongly speculated residuals is as follows:

$$i = \lceil \log_2(\rho + e) \rceil + 1.$$

For $\rho \leq 1$, this reduces to

$$i = \lceil \log_2(e + 1) \rceil + 1.$$

The value e is calculated as follows. Given a speculation function to obtain the quotient digit, for each rectangle R defined by $\hat{w}^s$ and $\hat{d}^s$ in the P-D diagram (see Fig. 8), we perform the following calculation:

$$e_R = \max_{q \in \{q_R\}} |q - \text{spec}(\hat{w}^s, \hat{d}^s)|,$$

where $\{q_R\}$ is the smallest set of quotient digits that cover the rectangle. Finally, $e = \max e_R$. Table III shows several examples of the minimum number of bits required.

In summary, to determine whether there is an error, it is necessary to have two comparators that perform the comparisons specified in (5). The estimate $\hat{w}^c$ has i integer bits and f fractional bits, whereas $\hat{d}^c$ has f fractional bits (the most significant always being 1).

Since the comparisons are done with truncated versions of $w^s[j+1]$ and d, there are cases in which the speculation is correct, but the comparison (to be conservative) fails. Consequently, the probabilities of successful speculation are somewhat smaller than those presented in Section III-A. Now these probabilities depend not only on the number of bits of $\hat{w}^s$ and $\hat{d}^s$ but also on f, and can be increased somewhat by using a larger value of f. For example, for $r = 16$ and $a = 12$, using $f = 4$ produces a probability of success of 0.73, whereas for $f = 8$, the probability is 0.77.

Error Detection for Partial Advance: As indicated before, a partial advance of $\log_2(p)$ bits is possible when the speculated residual satisfies

$$pw^s[j+1] \leq r\rho d,$$

because this is inside the bounds for the next iteration. Consequently, a partial advance can be performed if

$$\left|\frac{p}{r} w^s[j+1]\right| \leq \rho d.$$

These comparisons are done in a similar manner as those previously discussed, that is, using estimates of $w^s[j+1]$ and d.

B. Correction of Quotient Digit

Since, with the detection scheme presented in the previous section, when there is an error the correct digit is not known, it is necessary to perform an incremental correction and to check again whether a correct digit is obtained. For the scheme without partial advance, we have chosen to correct the quotient digit by $+1$ or -1, depending on the sign of the residual estimate, to assure that a sequence of correction cycles will reduce the residual until it is inside the bounds. This method requires that in some cases, more than one correction cycle be performed. However, this situation is found to be infrequent, so that the method is suitable.

On the other hand, for the case with partial advance, it is only necessary to assure that the bound will be reduced to $|w[j]| \leq (r/p)\rho d$. Consequently, it is possible to correct by a larger value. We have chosen the power of 2 that is immediately smaller than $(r/p)\rho$.

Fig. 9 depicts a block diagram of the circuit required for digit speculation, error detection, and digit correction. It consists of a multiplexor that selects between the speculated digit (q^s) and the correction (± 1 or q^c) according to the result of the comparison with the bounds. In the scheme with partial advance, two speculation tables (for q^s and q^p) and two comparisons with the bounds (for full and partial advance) are required.

V. EVALUATION

To evaluate the speed advantage of the scheme that we have described, we performed some implementations for 54-bit dividers. (This value affects only the area estimates.) Since there are many parameters that specify a particular implementation, we have not done a complete analysis of the solution space, but performed some reasonable designs to evaluate and compare.

All designs have used the same technology and design tools. In particular, we have used a 1-μm standard cell CMOS library [8] (size of a two-input NAND gate is 12.5×47.5 μm^2, delay of an inverter is 0.15 ns). Some simple modules have been designed by hand (multiplexors and CSA's), whereas *misII* [2] has been used for the synthesis of the quotient-digit selection

TABLE IV
CHARACTERISTICS OF DESIGNS

	conventional					speculative				
						no part. adv.		part. adv.		
radix	2	4	8	4 × 4	512	16	512	32	64	512
a	1	2	7	2 × 2	511	12	320	22	37	320
# of CSAs	1	1	2	2	12	2	4	2	2	4
# bits $\hat{w}^s$	(4-4)	(7-7)	(8-8)	(7-7)	12-12	(6-5)	12-12	(7-7)	(7-7)	12-12
# bits $\hat{d}^s$	0	3	3	3	15	1	6	2	3	6
cmp: (i,f)	-	-	-	-	-	3,5	6,4	4,5	4,6	6,4
part. adv.	-	-	-	-	-	-	-	3	4	6
cycles/digit	1	1	1	1	2.2	1.3	1.8	1.14	1.21	1.17
cycle delay	20.4	24.6	30.4	31.0	40.0	28.8	43.6	34.2	36.0	43.8
delay/bit	20.4	12.4	10.2	7.8	9.8	9.4	8.8	7.8	7.2	5.6
cell area	3100	3400	4500	4900	13600	4900	8400	6800	8900	10500
speedup	1.0	1.6	2.0	2.6	2.1	2.2	2.3	2.6	2.8	3.6
area factor	1.0	1.1	1.5	1.6	4.4	1.6	2.7	2.2	2.9	3.4

functions and the comparators. *MisII* has always been guided to optimize delay at the expense of increasing the area. Fan-in and fan-out capacitances (but no routing) have been considered for delay calculations. In Table IV, we report the final results. The main measures we use for comparison are the speedup and the area factor with respect to the conventional radix-2 case. These measures are summarized in Fig. 16. We now briefly comment on these designs; more details can be found in [7] for the conventional designs, and in [4] for the speculative cases.

A. Conventional Designs

The designs for radix 2, 4, and 8 use a table specification of the quotient-digit selection and a multilevel implementation with *MisII*. We were surprised by the good performance obtained for the radix-8 case. This is achieved by guiding *MisII* to provide a different delay for each of the two components forming the radix-8 digit. Going to higher radices with the table approach does not improve the speed, because of the increase in complexity of the quotient-digit function. The radix-16 design uses two overlapped radix-4 stages (called 4 × 4 in Table IV).

On the other hand, the radix-512 design uses an arithmetic function for the quotient-digit selection, based on expression (3). As can be seen, this does not produce a significant improvement in speed; probably an even higher radix would result in a faster implementation, but at the expense of an even higher area factor.

B. Speculation Without Partial Advance

The average execution time per quotient bit is given by

$$T = \frac{C_d}{\log_2 r} \times t_c,$$

where C_d is the average number of cycles per quotient digit (this includes the speculation cycles and the correction cycles), and t_c is the delay of one cycle.

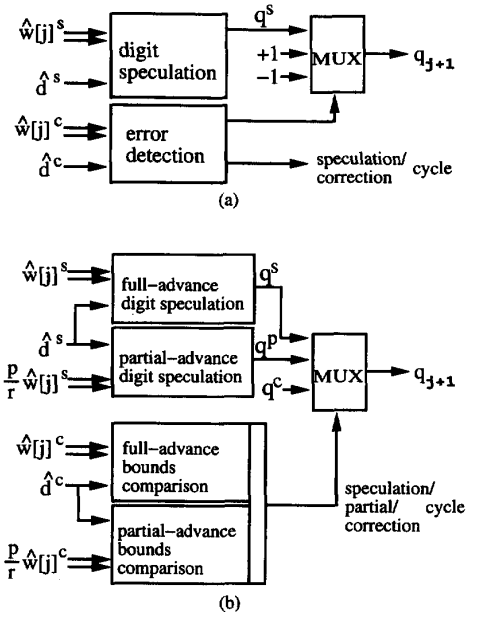


Fig. 9. Units for digit speculation. (a) Basic scheme. (b) With partial advance. (q_{j+1} denotes q^s_{j+1}, q^p_{j+1}, or q^c_{j+1} depending on the type of cycle.)

The results shown are cases in which increasing the radix produces a faster execution. The radix-16 design uses a speculation table approach, whereas the radix-512 one uses an arithmetic approach.

We conclude that without partial advance, no improvement is obtained with respect to the fastest conventional design (radix-16 with two overlapped radix-4 stages).

C. Speculation with Partial Advance

To calculate the average number of cycles per quotient digit, we perform simulations that are similar to those discussed in

TABLE V
SPECULATIVE DIVISION WITH PARTIAL ADVANCE

Radix	32			64			512		
$\log_2 p$	4	3	2	5	4	3	7	6	5
p_f	0.55	0.64	0.58	0.50	0.50	0.50	0.39	0.44	0.47
p_p	0.33	0.36	0.42	0.31	0.49	0.50	0.50	0.54	0.52
p_n	0.12	0.00	0.00	0.19	0.01	0.00	0.11	0.02	0.01
C_d	1.20	1.14	1.30	1.31	1.21	1.29	1.22	1.17	1.27

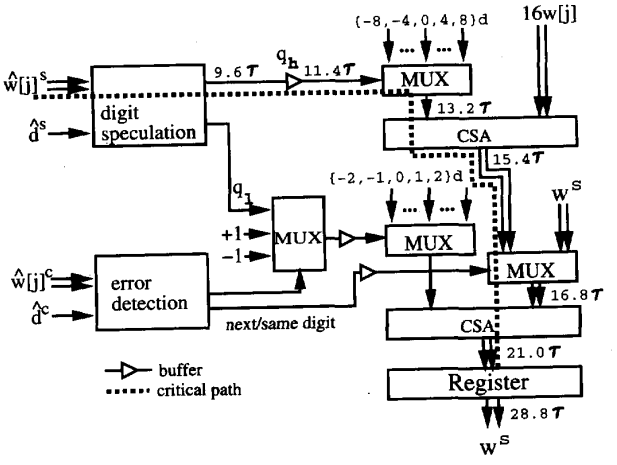


Fig. 10. Block diagram for a speculative radix-16 divider.

TABLE VI
AREA AND DELAY FOR THE SPECULATIVE RADIX-16 DIVIDER

Module	Area	Delay (τ)
digit speculation	270	9.6* (for q_h) 12.6 (for q_l)
error detection	210	10.6
MUX for $q_i d$	2×310	1.4*
MUX for $w[j]$	90	1.8*
MUX for q_l	4	1.8
CSA (56 bits)	2×380	4.2* (from residual) 2.2* (from $q_i d$)
Buffer	9×3	1.8*
Registers	166×6	7.8*
Conversion & Rounding (see [7])	1900	
Total	4900	28.8

(* indicate delays in the critical path)

Section III-A. Now three types of cycles exist, namely, full cycles (with probability p_f), partial cycles (with probability p_p), and no-advance cycles (with probability p_n). The average number of cycles per digit is

$$C_d = \frac{\log_2 r}{p_f \log_2 r + p_p \log_2 p}. \quad (6)$$

Expression (6) holds when the number of quotient digits is very large. However, for the case of n-bit dividers (being n small), the value of C_d varies slightly because more than n bits may be generated when different types of cycles are executed during the division. Table V shows the number of cycles for several radices, several values of p, and $n = 54$.

We have performed several designs, selecting from Table V the values of p that produce the lowest average number of cycles per digit. Table IV shows the characteristics for implementations that are faster than those described in Section V-B.

We observe a progressive speedup when the radix is increased. The radix-512 design achieves a speedup of 1.4 with respect to the fastest conventional.

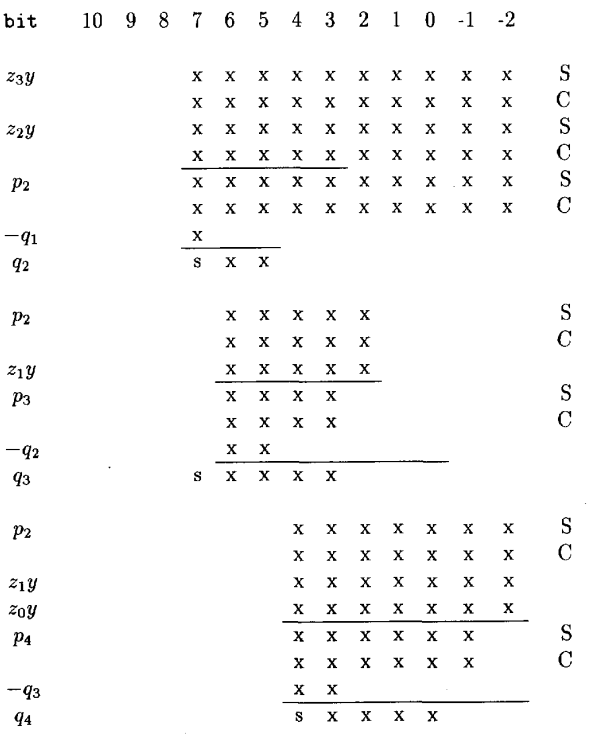


Fig. 11. Multiplication of zy and generation of q.

VI. DESIGN EXAMPLES

We now present the implementation details of two designs. The first is a radix-16 without partial advance and is presented because of its simplicity. The second is a radix-512 with partial advance and corresponds to the fastest of the designs we performed.

A. Radix-16 Without Partial Advance

The block diagram is shown in Fig. 10, and the characteristics are shown in Table IV. The quotient-digit generated by the speculation function is $q_h + q_l$, where $q_h \in \{-8, -4, 0, 4, 8\}$ and $q_l \in \{-2, -1, 0, 1, 2\}$. Therefore, the values $\{-12, -11, 11, 12\}$ are always obtained after corrections of the initially speculated digit. Although this increases somewhat the number of correction cycles, the use of $a = 12$ results in a larger overlap than $a = 10$, which makes the implementation of the speculation function simpler and faster. Moreover, the limited precision of the comparisons (error detection) reduces the range of "accepted" residuals and consequently diminishes the probability of requiring quotient digits near $\pm a$. Exploring several designs, we found $a = 12$ to be the best trade-off for radix-16.

Table VI reports area and delay characteristics of this design. Because q_h is the highest-weight component of the quotient digit, the speculation function is simpler and faster than for q_l. When synthesizing the speculation function, *misII* has been guided to reducing the delay of q_h, which, in this case, is in the critical path.

The CSA has been designed as a radix-2 full adder. Its delay (4.4τ) is determined by two cascaded XOR gates (2.2τ each). However, the outputs of the $q_i d$ multiplexors have been connected to the last gate in order to reduce the critical path. (The same optimization has been used for the conventional designs.) This approach cannot be used with the residual, because the redundant representation requires two signals.

B. Radix-512 Divider with Partial Advance

This unit uses as speculation function an approximation of expression (3). This approximation is obtained by the following method.

1) Reducing the accuracy of the approximation $z \approx 1/d$. This reduces the delay of the calculation of this approximation, so that no additional cycle is required for it, and it reduces the number of bits of z so that the multiplication $z\hat{w}$ is simplified. The resulting z has four components:

$$z = z_3 + z_2 2^{-2} + z_1 2^{-4} + z_0 2^{-6},$$

where $z_3 \in \{1, 2\}$, and the others have values $\{-1, 0, 1, 2\}$.

2) Only four components of the radix-512 digit are generated, so that $q^s = q_1 + q_2 + q_3 + q_4$, with values $q_1 \in \{-256, -128, 0, 128, 256\}$, $q_2 \in \{-64, -32, 0, 32, 64\}$, $q_3 \in \{-64, -32, -16, 8, 0, 8, 16, 32, 64\}$, and $q_4 \in \{-8, -4, -2, -1, 0, 1, 2, 4, 8\}$. This reduces the complexity of the generation of the speculated digit and reduces the delay of the iteration, because only four CSA are used (see Fig. 12) instead of the six that would be required in a standard radix-4 recoding of the quotient digit. This decomposition of the radix-512 digit does not permit the generation of all values; consequently, several values are speculated incorrectly, and a partial advance cycle has to be used. Note that the components overlap to allow the generation of a larger subset of values.

3) Use a left-to-right multiplier [6] to compute $z\hat{w}$. Moreover, the components of the speculated quotient digit are extracted at different levels of the multiplier. As shown in Fig. 11, this corresponds to the following expressions:

$$\begin{aligned}
\mathtt{p2} &= \mathtt{z3\ \ y} + \mathtt{z2\ \ y} \\
\mathtt{q2} &= \mathtt{sel(trunc(p2 - q1))} \\
\mathtt{p3} &= \mathtt{p2} + \mathtt{z1\ \ y} \\
\mathtt{q3} &= \mathtt{sel(trunc(p3 - (q1 + q2)))} \\
\mathtt{p4} &= \mathtt{p2} + \mathtt{z1\ \ y} + \mathtt{z0\ \ y} \\
\mathtt{q4} &= \mathtt{sel(trunc(p4 - (q1 + q2 + q3)))},
\end{aligned}$$

where $y = r\hat{w}$, the `trunc` function truncates to a suitable number of bits and the `sel` function encodes the value on the corresponding digit set. This extraction at different levels matches the delays in the residual generation part, and reduces the overall delay. However, this might not produce the correct quotient-digit components, because carries from the least-significant portion of the multiplication are not included. These errors are partially compensated by actually computing the remainder obtained by subtracting the more significant components of the quotient digit.

The corresponding digit-speculation implementation is shown in Fig. 13. Note that to adapt to the delays of the z's, some precomputation on y is performed before the MUX, either by calculating the p and g of each bit to reduce the delay of the 4-2 CSA, or by assimilating the required number of bits, to reduce the number of summands.

The delay of a cycle is obtained from the delays shown in Figs. 12 and 13. The resulting critical path is q_4-DRIVER-MUX-CSA-REG and results in 43.8τ.

The number of cycles per digit is obtained by simulating the actual implementation. The value $p = 6$ in the partial advance is selected to achieve the minimum number of cycles.

The area is computed for the components of Figs. 12 and 13 plus the comparators and the quotient conversion and rounding. Note that two quotient-digit speculation modules are required: one for full advance and one for partial advance. They are identical, but use different bits of $w[j]$. The total area is equivalent to about 10 500 two-input NAND gates.

VII. Extension to Square Root

The scheme we have described can be used also for a digit-recurrence square-root algorithm [3], [7], [12]. The recurrence is

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

where $S[j]$ is the result after the jth iteration, and s_{j+1} is the $(j+1)$th result digit.

The result-digit selection function,

$$s_{j+1} = \mathrm{sel}\big(\hat{w}[j], \hat{S}[j]\big),$$

is determined so that the residual is bounded by

$$-2\rho S[j] + \rho^2 r^{-j} < w[j] < 2\rho S[j] + \rho^2 r^{-j}.$$

The same two approaches used for the result-digit selection are also possible, namely a table that has as inputs an estimate of the residual and of the result (instead of the divisor), and an arithmetic procedure that computes the result digit by multiplying an estimate of the residual by an approximation of the reciprocal of the result. With respect to division, the following complications arise.

1) For the implementation of the residual, the additional term $s_{j+1}^2 r^{-(j+1)}$ has to be generated and subtracted. This increases the delay, because this term requires the value of s_{j+1}.
2) $S[j]$, which is generated in signed-digit form, has to be converted to a form suitable for the recurrence. This conversion can be done by the on-the-fly algorithm as described in [5].

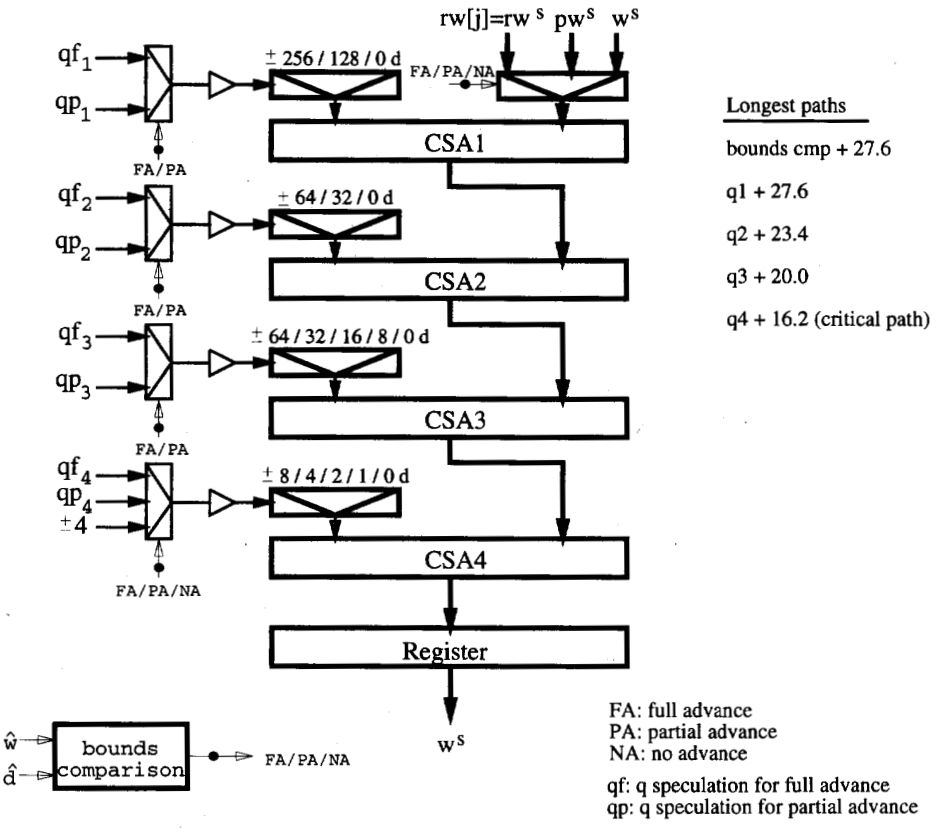


Fig. 12. Block diagram of the CSA's used for radix-512 division.

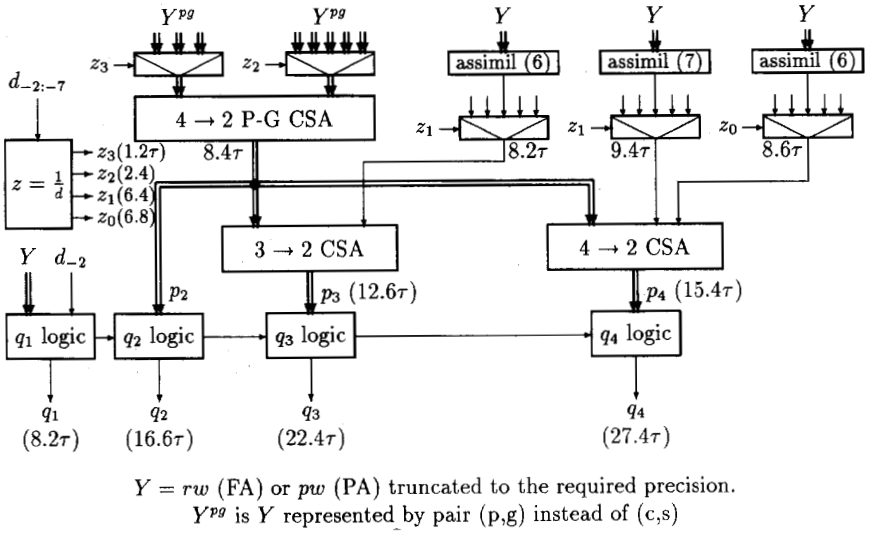


Fig. 13. Block diagram of the digit-speculation circuit for radix-512 division.

3) The bound is now a function of j, so that the result-digit selection might be different in different iterations. Because the term depending on j is $\rho^2 r^{-j}$, this has an effect only in the first iterations. In a conventional algorithm, this is solved by obtaining the initial bits of the result directly from the argument. In the case with speculation, it would be possible to use the same result-digit selection for all iterations and handle the anomaly of the initial ones by correction. Because this produces too many correction cycles, in the implementation we

$$
\begin{array}{lccccc}
(p_3)\{ & s_6 & s_5 & s_4 & s_3 & s_2 \\
 & c_6 & c_5 & c_4 & c_3 & c_2 \\
 & & & & & +1 \quad \text{(rounding)} \\
\hline
 & t_6 & t_5 & t_4 & t_3 & t_2 \\
(-q_2) & x_6 & x_5 & & & \\
\hline
 & u_6 & u_5 & u_4 & u_3 & u_2
\end{array}
$$

$u_{6:2}$	q_3	$u_{6:2}$	q_3
0000-	0	100--	-64
0001-	8	10100	-64
0010-	16	10101	-32
00110	16	1011-	-32
00111	32	1100-	-32
010--	32	11010	-32
01100	32	11011	-16
01101	64	1110-	-16
0111-	64	1111-	-8

Fig. 14. Implementation of q_3 logic block.

performed, we used a hybrid approach in which one digit of the result is obtained directly from the operand, and for the rest a single selection function is used. Moreover, to simplify the bound comparisons (to avoid the multiplication of $S[j]$ by ρ and the addition of $\rho^2 r^{-j}$), we compare with $kS[j]$, where k is a constant that makes the product easy to compute and the bound satisfied for any j (has to absorb the effect of $\rho^2 r^{-j}$ for the negative bound).

A. Combined Division and Square Root

It is interesting to combine division and square-root in a single unit. Such units have been reported in [9]. The result-digit selection function is the same for both operations, with the exception of the beginning; this is due to the effect of the term of the bound of square-root dependent on j. The solution for this is discussed above.

We have developed such a combined implementation for the scheme with speculation. This implementation has two objectives:

1) To share as much as possible the hardware among both operations, and
2) Not to increase the execution time for division, which we assume is much more frequent than square-root.

Division is done using the recurrence and quotient-digit selection discussed before. For square-root, as is usually done to combine it with division, a new residual $v[j]$ is defined, such that

$$v[j] = 2^{-1} w[j]$$

Moreover, to account for the full advance, partial advance, and no advance, we define a sliding 1 by

$$R[j+1] = m^{-1} R[j],$$

with the initial condition $R[0] = 1$, where $m = r$ for full advance, $m = p$ for partial advance, and $m = 1$ for no advance. The square-root recurrence becomes

$$v[j+1] = mv[j] - S[j]s_{j+1} - 2^{-1} s_{j+1}^2 R[j] m^{-1}$$
$$S[j+1] = S[j] + m^{-1} R[j] s_{j+1},$$

and the bound

$$-\rho S[j] + 2^{-1}\rho^2 R[j] < v[j] < \rho S[j] + 2^{-1}\rho^2 R[j].$$

As indicated, we want to maintain the delay of a cycle the same as that obtained for division alone. To achieve this, we retime the square-root recurrence by postponing the subtraction of $2^{-1} s_{j+1}^2 R[j] m^{-1}$ until the next cycle. That is,

$$w[j+1] = m(w[j] - 2^{-1} s_j^2 R[j]) - S[j] s_{j+1}, \qquad (7)$$

(calling this residual w to conform with the division case), which has the same delay as for division, because the term $2^{-1} s_j^2 R[j]$ does not depend on s_{j+1}. This implementation is shown in Fig. 15, together with the controls for division and square-root.

We want to perform the result-digit speculation by

$$s_{j+1}^s = \text{Spec}(\hat{w}[j], \hat{S}[j]),$$

using the same function as for division. Moreover, the bound comparisons should also use $\hat{w}^c$. However, in addition to the problem at the beginning discussed before, another complication arises because of the retiming, since the value of $w[j]$ is not the correct residual of square-root, because the term $2^{-1} s_{j+1}^2 R[j] m^{-1}$ has not been subtracted. On the other hand, this term has an effect on $\hat{w}$ only in the first iterations.

The effect on the speculation function can be taken care of by the nature of the speculation process that allows errors in the speculation and corrects them by partial advance and/or by correction cycles. However, if the speculation produces large errors, this might increase by too much the number of cycles.

On the other hand, the error in the comparisons is more severe, because these comparisons determine the correctness of the algorithm. Consequently, it is necessary to have a sufficiently accurate $\hat{w}^c$ to assure correct comparisons. For this, in the first iterations, we perform one cycle to update the value of w before doing the comparison (and the result-digit selection). That is, for the first few iterations we perform an iteration in two cycles, as indicated below.

Cycle 1:

$$v[j] = w[j] - 2^{-1} s_j^2 R[j].$$

Cycle 2:

$$s_{j+1} = \text{Spec}(\hat{v}[j], \hat{S}[j])$$

and comparison

$$w[j+1] = mv[j] - S[j] s_{j+1}.$$

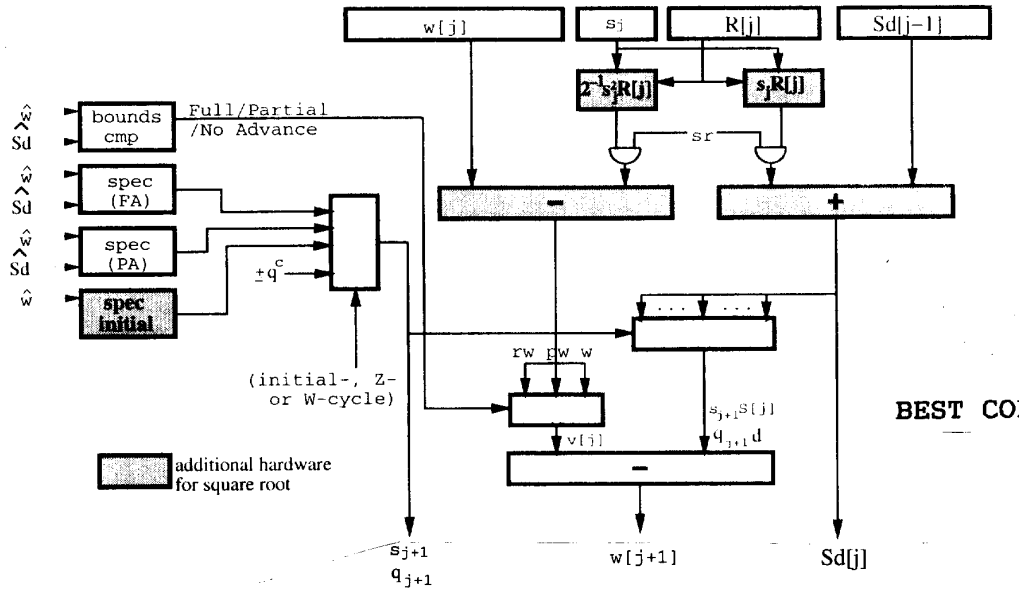


Fig. 15. Unit for division and square-root.

Consequently, in Cycle 1, no shift is performed (this connection is required anyhow for the correction cycles), and a result-digit equal to zero is used. In Cycle 2, the result digit is speculated (and the comparison performed), the subtraction of the term s_j^2 is inhibited, and the shift is performed.

The number of double-cycle iterations required has to be sufficient to make the term on s_j^2 not affect $\hat{w}^c$. Since $s^2 \leq a^2$, if we call b_w the number of fractional bits of $\hat{w}^c$, double-cycle iterations have to be performed while the following condition exists:

$$R[j] > \frac{2^{-b_w}}{a^2}.$$

After these iterations, the retimed iteration can be used. The algorithm for square-root then consists of the following steps:

1) An initial iteration that obtains the first digit of the result directly from the operand,
2) Several double-iterations as discussed above, and
3) Retimed iterations given by expression (7).

B. Radix-512 div/sqrt Unit

We made an implementation for radix-512 using the design for division with speculation and partial advance described in Section VI-B. The resulting circuit is shown in Fig. 15.

Signal `sr/div` controls whether a division or square-root has to be executed. Register `Sd` stores either the divisor (for division) or the result (for square-root). The following are the additional hardware introduced for square-root.

- *A speculation circuit for the initial iteration.* For the radix-512 implementation, 8 bits of the operand are taken to produce an 8-bit estimation of the result. This circuit is not in the critical path.
- *A unit to subtract $2^{-1}s^2R[j]$ from the shifted residual.* This unit is disabled when division is executed.
- *A unit to calculate the result for iteration j by adding $s_jR[j]$ to $S[j-1]$.* This unit is also disabled for division in such a way that register `Sd` always contains de divisor.
- *As indicated before, the comparisons are done using $kS[j]$, where k is a simple constant.* As in our case $\rho = 320/511$, we can use $k = 0.625$, because we have the following condition:

$$2^{-10} < \rho - 0.625 < 2^{-9},$$

which is sufficient to absorb the term $\rho^2 r^{-1}$, which appears in the negative bound.

In this case, double-cycle iterations must be executed until 9 bits of the result are obtained. Because of the possibility of partial advance, the number of cycles required for this phase of the algorithm varies.

It is important to notice that the critical path for the division is not lengthened by the additional units introduced in the circuit. The calculation of $2^{-1}s_j^2R[j]$ and $s_jR[j]$ and the subtraction and addition with $w[j]$ and $S[j-1]$, respectively, are executed in parallel with the speculation of the result digit. Furthermore, these additional units are not in the critical path of the circuit. Therefore, division and square-root can be executed with the same clock cycle without increasing the original delay required for division only. This is possible because of the retiming performed by (7).

Simulations performed with this div/sqrt unit show that the average number of cycles/digit for square-root is 1.6 (division is executed with 1.17 cycles/digit). If a square-root unit would have been designed without the retimed recurrence, and therefore with a significantly longer cycle time, the cycle count would have decreased to 1.3 cycles/digit. However, this would increase the delay for division. Consequently, for the case in which frequency of division is much larger than that

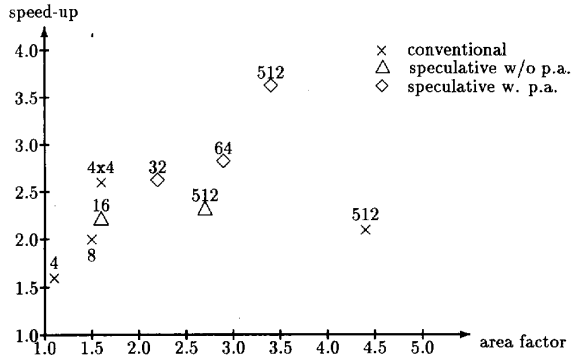


Fig. 16. Summary of implementations for different dividers.

of square-root, the approach based on retiming the square-root recurrence is superior.

VIII. Summary and Conclusion

The division and square-root method that we have presented is based on the speculation of the result digit and on a rollback when the speculation is incorrect. Because of the reduction in complexity of the result-digit selection, this can result in faster implementations with higher radices than for the conventional approach. Moreover, a reduction in the number of adders required is possible by speculating only a reduced set of result-digit values.

We have discussed approaches to determine a suitable speculation function. Moreover, we detect whether the speculation is correct by determining whether the next residual is inside the required bound.

The approach is extended to the partial-advance case. In this scheme, when an error occurs in the speculation, it is not always necessary to correct the previous digit; but it is possible, if the error is small enough, to advance a number of bits that is smaller than a radix-r digit. In this way, the penalization of errors is significantly reduced. We determined the condition required for partial advance and described an implementation that replicates the result-digit selection.

We extended the scheme to square-root and described a combined division/square-root implementation. We have developed this design so that the more frequent division operation is not slowed down by the added complications of the implementation of square-root.

We performed several designs using the same technology and determined the relative speed and area. The results for division are summarized in Fig. 16. Without partial advance, no improvement is obtained with respect to the fastest conventional (radix-16 composed of two overlapped radix-4 stages). On the other hand, the scheme with partial advance produces a speedup of up to 1.4 (again with respect to the fastest conventional).

We performed a design of a radix-512 combined division/square-root unit. This produces no speed degradation for division. The square-root operation would be about 1.4 times slower than division, because of the double-cycles and a somewhat higher error rate.

Acknowledgment

We thank P. Montuschi for his valuable discussions and comments.

References

[1] D. E. Atkins, "Higher-radix division using estimates of the divisor and partial remainder," *IEEE Trans. Comput.*, vol. C-17, pp. 925–934, Oct. 1968.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Comput.-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[3] L. Ciminiera and P. Montuschi, "Higher radix square-rooting," *IEEE Trans. Comput.*, vol. 39, pp. 1220–1231, Oct. 1990.

[4] J. Cortadella and T. Lang, "Division with speculation of quotient digits," UPC/DAC Tech. Rep., 1993.

[5] M. Ercegovac and T. Lang "On-the-fly conversion of redundant into conventional representations," *IEEE Trans. Comput.*, vol. 36, pp. 895–897, July 1987.

[6] ——, "Fast multiplication without carry propagation," *IEEE Trans. Comput.*, vol. 39, pp. 1385–1390, Nov. 1990.

[7] M. Ercegovac and T. Lang, *Division and Square-Root: Digit-Recurrence Algorithms and Implementations.* Boston: Kluwer Academic, 1994.

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

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

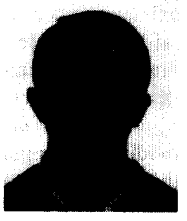
[10] T. Lang and P. Montuschi, "Higher radix square-root with prescaling," *IEEE Trans. Comput.*, vol. 41, pp. 996–1009, Aug. 1992.

[11] D. W. Matula, "Design of a highly parallel floating point arithmetic unit," *Symp. Combinatorial Optimization Sci. and Technol. (COST)*, Apr. 1991.

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

[13] G. S. Taylor, "Radix-16 SRT dividers with overlapped quotient selection stages," *Proc. 7th IEEE Symp. Comput. Arithmetic*, 1985, pp. 64–71.

[14] T. E. Williams and M. A. Horowitz, "A 160 ns 54 bit CMOS division implementation using self-timing and symmetrically overlapped SRT stages," *Proc. 10th Symp. Comput. Arithmetic*, 1991, pp. 210–217.



J. Cortadella received a degree in computer science from the Polytechnic University of Catalonia, Barcelona, Spain, in 1985, and the Ph.D. degree from the same university in 1987.

He is a Professor at the Department of Computer Architecture, Polytechnic University of Catalonia. In 1988, he was a Visiting Scholar at the University of California, Berkeley. His research interests include computer arithmetic, computer-aided design of very large scale integration (VLSI) systems, with emphasis on synthesis and verification of asynchronous circuits, and parallel architectures.

Dr. Cortadella has coauthored more than 30 research papers in technical journals and conferences.

T. Lang for photograph and biography please see p. 918 of this TRANSACTIONS.