Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

ALTERNATIVE IMPLEMENTATIONS OF A RADIX-4 DIVIDER WITH SCALING

Ramin Modiri
Tomas Lang

September 1988
CSD-880069

Alternative Implementations of a Radix-4 Divider with Scaling

Ramin Modiri and Tomas Lang

Department of Computer Science
School of Engineering and Applied Science
University of California, Los Angeles

Abstract

Several alternative designs for a radix-4 divider with scaling are presented and compared with a radix-2 divider and with a radix-4 divider without scaling. The designs use CMOS macrocells. The comparison shows that, for this technology and for a 32-bit quotient, the implementation with scaling produces a speed-up of 1.5 with respect to the radix-2 divider and 1.25 with respect to the radix-4 divider without scaling. This speedup is obtained at the expense of an increase in the number of cells by a factor of 2.5 and 1.2, respectively.

1. Introduction

In this report we consider several implementations for a radix-4 divider with scaling, described in [ERCE87a], using LSI Logic CMOS gate arrays, and compare the speed to a radix-2 divider and to a radix-4 divider without scaling. In the designs, areas of special interest include the implementation used in the adder and in the quotient-digit selection. The implementations are for 32-bit fractional and normalized operands. We use the Series 7000 of CMOS macrocells [LSI85].

We conclude that the fastest implementation for the scheme with scaling achieves a speed-up of 1.5 with respect to the radix-2 scheme at a cost of a factor of 2.5 in the number of gates. In contrast, the radix-4 implementation without scaling produces a speedup of 1.25 with a gate factor of 1.9.

2. The Basic Scheme

The basic algorithm and high-level implementation of the radix-4 division with scaling is described in [ERCE87a]. As shown in Figure 1, the operation consists of three steps: operand scaling, division recurrence, and quotient conversion. The scaling step finds a factor to scale the divisor into the range [63/64,9/8] and then scales both divisor and dividend by that factor. The scaling is done by addition of three multiples of the operand according to Table 1.

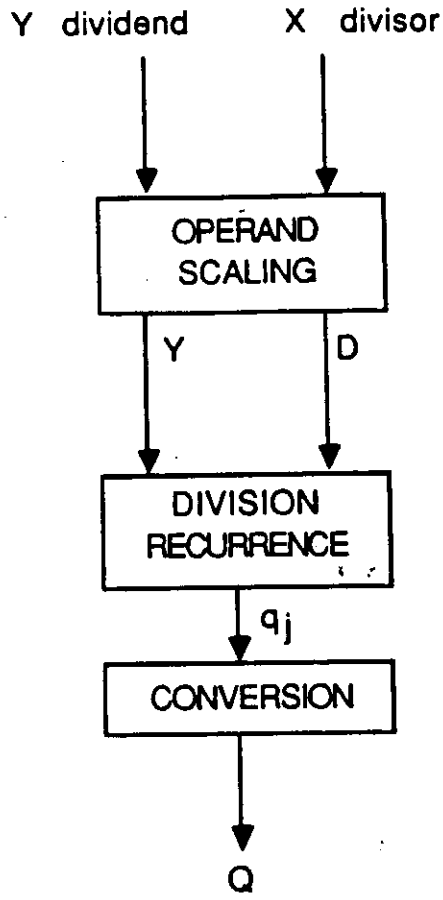

Figure 1. Division Scheme with Scaling

Table 1. Scaling Intervals

X	M	D
[14/16, 1)	9/8 = 1+1/8	[252/256, 288/256)
[13/16, 14/16)	10/8 = 1+1/4	[260/256, 280/256)
[12/16, 13/16)	11/8 = 1+1/4+1/8	[264/256, 286/256)
[11/16, 12/16)	12/8 = 1+1/2	[264/256, 288/256)
[10/16, 11/16)	13/8 = 1+1/2+1/8	[260/256, 286/256)
[9/16, 10/16)	14/8 = 1+1/2+1/4	[252/256, 280/256)
[8/16, 9/16)	16/8 = 1+1	[256/256, 288/256)

The division recurrence consists of two parts: the selection of the quotient digit and the computation of the next partial remainder, as shown in Figure 2.

That is,

$$q_j = F(4R[j-1], D)$$

$$R[j] = 4R[j-1] - q_j D$$

The main characteristics of the high-level implementation are

i) The recurrence uses a redundant adder for fast carry-free addition

ii) The quotient digit set is {-2,1,0,1,2} to have a simple computation of the divisor multiples.

iii) Because of the divisor scaling, the quotient-digit selection function is independent of the value of the divisor and can be obtained from a limited precision estimate of the partial remainder. This selection function depends on the type of representation of the partial remainder. For the two most typical redundant representations (carry-save and signed-digit) the functions are

a) for carry-save (2's complement) representation

$$q_j = \begin{cases} 2 & \textbf{if } w \geq 3/2 \\ 1 & \textbf{if } 11/8 \leq w \leq 1/2 \\ 0 & \textbf{if } 3/8 \leq w \leq -1/2 \\ -1 & \textbf{if } -5/8 \leq w \leq -13/8 \\ -2 & \textbf{if } -7/4 \leq w \end{cases}$$

where w is an estimate of $4R[j-1]$ with three fractional bits.

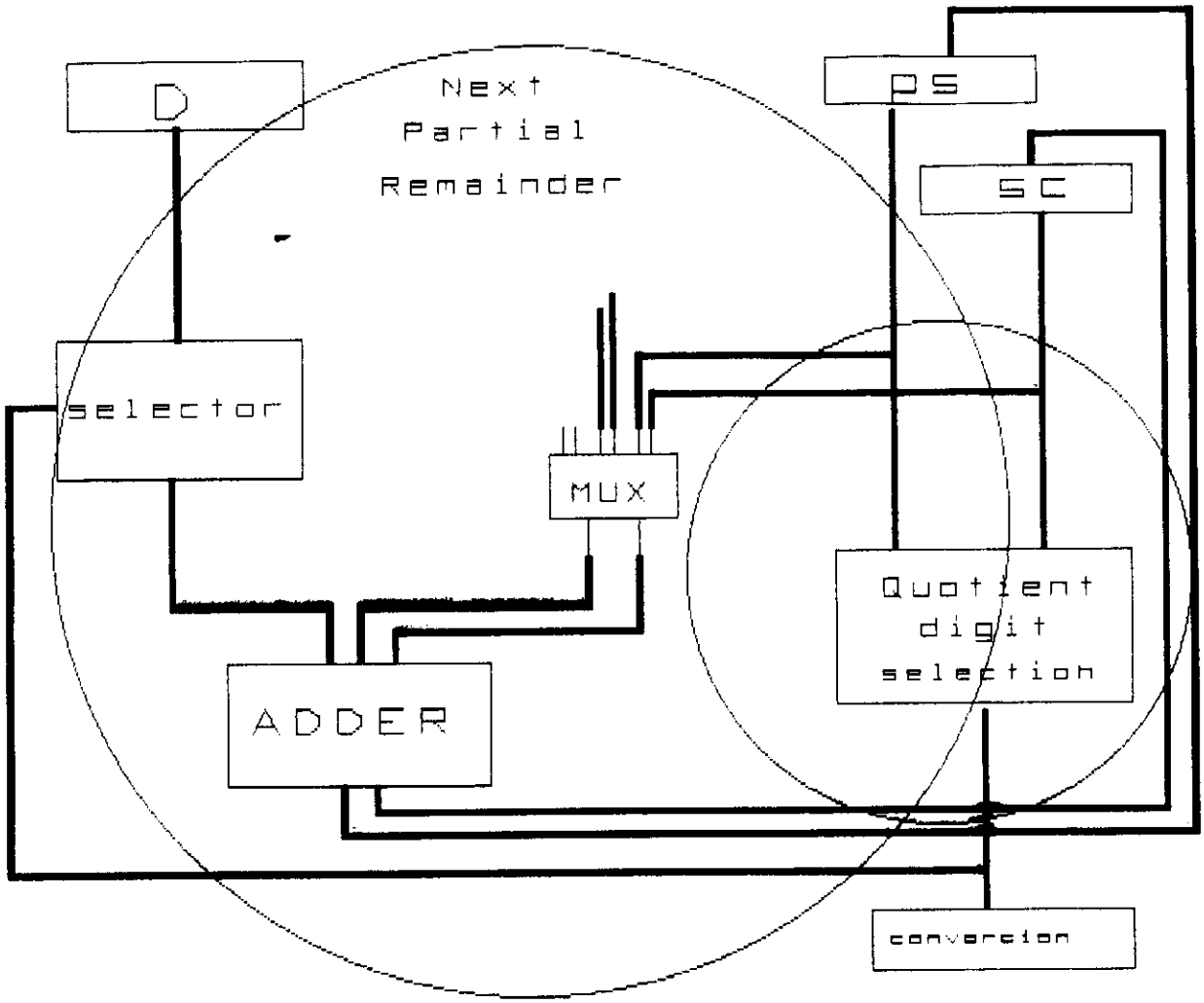

Figure 2. Main Division Cycle

b) for signed-digit representation

$$q_j = \begin{cases} 2 & \textbf{if } w \geq 13/8 \\ 1 & \textbf{if } 12/8 \leq w \leq 5/8 \ (or\ 1/2) \\ 0 & \textbf{if } 1/2\ (or\ 3/8) \leq w \leq -3/8\ (-1/2) \\ -1 & \textbf{if } -1/2\ (or\ 5/8) \leq w \leq -12/8 \\ -2 & \textbf{if } -13/8 \leq w \end{cases}$$

where again w is an estimate of $4R[j-1]$ with three fractional bits.

iv) The quotient is converted to conventional representation using an on-the-fly converter, as described in [ERCE87b]. This conversion does not add to the delay of the division.

Several designs will be considered in this report. These designs are determined by the type of redundant representation of the partial remainder. The alternative representations influence the design of the quotient selection function, of the adder, and of the scaling. The representations we consider are:

A) Radix-2 carry-save
B) Radix-4 carry-save
C) Radix-2 signed-digit
D) "0-1-2"
E) Radix-4 signed-digit

The two most promising schemes, radix 2 carry-save and radix 4 carry-save, are discussed in length. Toward the end of the report, we also briefly discuss the other alternatives.

3. Implementation using a (radix-2) carry-save adder.

The implementation, shown in Figure 3, consists of the following components

a) Register D to contain the divisor (unscaled), the dividend (unscaled) and the scaled divisor.

b) The register pair PS and SC to contain the partial remainder in carry-save form.

c) The quotient-digit selection function, which has as input the six most significant bits of $4PS$ and $4SC$. It has five outputs corresponding to the five values of the quotient digit. This quotient digit is used in two places: the selection of the divisor multiple in the recurrence and the on-the fly conversion.

d) A 3-2 carry-save adder consisting of 32 full adders.

e) Selectors at the carry-save adder inputs for the scaling and for the recurrence. These selectors are implemented using vectors of tri-state gates.

f) Logic to determine the scaling factor, using five bits of the unscaled divisor. The scaling factor is stored in register S.

g) A carry-propagate adder to assimilate the scaled divisor.

h) Two registers for the quotient conversion.

The timing of the operation is as follows:

Cycle 1. Assuming that the unscaled divisor is already stored in register D, the scaling factor is determined and stored in S. At the same time, the divisor is scaled and the scaled value in carry-save form is stored in registers PS and SC. The dividend is stored in register D.

Cycle 2. The scaled divisor is assimilated by the CPA (this takes two cycles)

Cycle 3. The scaled divisor is assimilated (second cycle) and is stored in D. At the same time, the dividend is scaled and stored in PS and SC.

Cycles 4 to 19. The recurrence is performed to obtain the 16 quotient digits.

In the implementation using CMOS LSI Logic macrocells, the most difficult component to design is the quotient-digit selection function. Two designs were considered: in the fastest, the quotient-digit values are obtained directly from the carry-save form of the partial remainder (six pairs of bits), while in the second, the six pairs of bits are first assimilated using a 6-bit carry-propagate adder and the six output bits of the adder used to obtain the quotient-digit value.

For the first alternative the following design is obtained. First, the corresponding bits of both vectors are decoded into the values 0,1, and 2. Then, those values are used in a NAND-NAND implementation to obtain the quotient-digit value. Using the minimization program espresso, we arrive at 78 product terms with a maximum of 18 product terms for any given quotient-digit value.

The time for decoding is 2.8 nsec. After decoding, the path is composed of two "conceptual" NAND levels, as follows. The first level generates the individual product terms with the use of an 8-input NAND gate with FO(fanout)=1. The second level is an equivalent 18-input NAND gate and is realized using three levels: 6-input NAND (FO=4), inverter/buffer (FO=32), and 3-input NAND gate (FO=2).

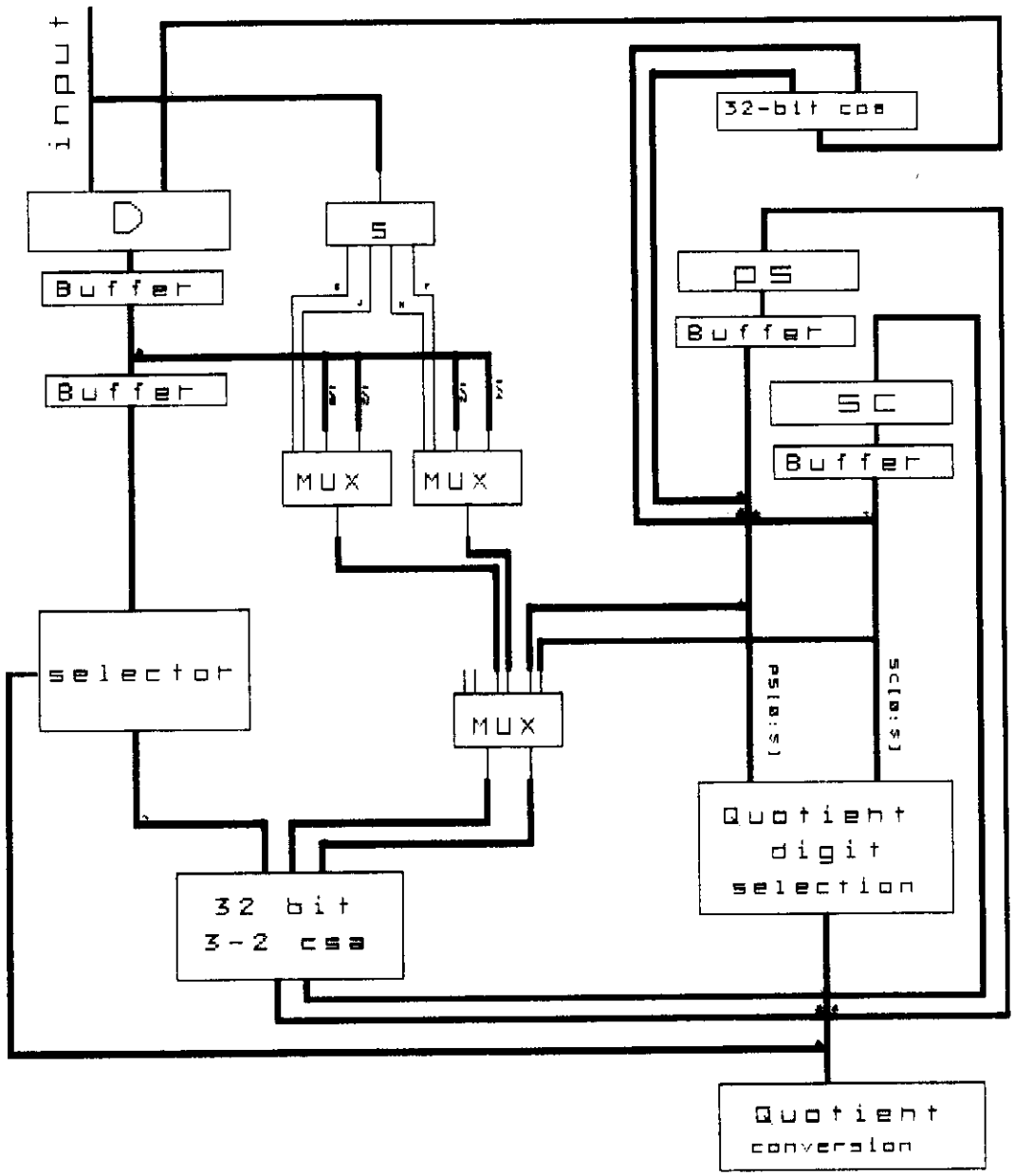

Figure 3. Implementation using radix-2 CSA

Note that the quotient-digit values must have an equivalent fanout of 64 to drive 32 tri-state gates with a load factor of 2 each. In designing the 2nd NAND level, we took advantage of the fact that the inverter is much less sensitive to load (i.e. delay does not change significantly as its fanout increases) than the 3-input NAND gate. Also, since the gate count for each 3-input NAND is small(2 gates), we chose to meet the 64 fanout criterion by connecting the inverter to 32 3-input NAND gates, each with a fanout of 2.

Furthermore, since after the decoding level the number of bits to the selection function is only 6, each product term has a maximum of 6 literals. Consequently, the first-level 8-input NAND gates have 2 extra inputs. These are used to inhibit (set to logic "0") the selection signals during the scaling. Selecting the Q = -1 signal is no problem since the last NAND gate can change from 3-input to 4-input, with no increase in delay (because the fanout is 2) nor in gate count. Remember that during the scaling cycles the "QD" multiplexer must pass 1*D, which means that the quotient digit must be set equal to "-1".

The delay for this quotient-digit selection function is

3.7 nsec + (3.6 nsec + 2.3 nsec + 2.1 nsec) = 14.4 nsec

The second alternative for the quotient-digit selection assimilates 6 bits of the partial remainder using a 6-bit adder. This results in a quotient-selection time of 17.9 ns.

The quotient conversion is implemented as described in [ERCE87b]. Two 33-bit shift registers are used, each capable of 2-bit shift and parallel loading. The control signals for these registers are produced using one level of NAND gates, which use the quotient digit values as their input. The implementation of this module is fairly simple, and requires only 450 gates.

The critical path is given in Table 2a and the number of gates in Table 2b.

Table 2a. Critical path for the radix-2 carry-save alternative

	No assimilation	With assimilation
Component	Delay [ns]	Delay [ns]
Flip-flop (propagation delay)	2.1	2.1
Inverter	1.5	1.5
Quotient-digit selection	14.4	17.9
3-state gates	2.6	2.6
carry-save adder	4.2	4.2
Flip-flop setup time	1.5	1.5
TOTAL	26.3	29.8

Table 2b. Number of gates for the radix-2 carry-save alternative
(in thousands)

Component	A	B	C	D	E	F	G	TOTAL
No assimilation	1.4	0.5	0.4	0.7	0.4	0.5	0.8	4.7
With Assimilation	1.4	0.5	0.4	0.1	0.4	0.5	0.5	3.8

A: quotient conversion, registers, inv-buffer after each F-F.
B: adder for assim. of divisor
C: main adder
D: selection function
E: scaling adder and multiplexers
F: 3-state gates for passing multiple of divisor.
G: extra buffers

In summary the radix-2 carry-save alternative has the following characteristics:

	No assimilation	With assimilation
Cycle time (ns.)	26.3	29.8
Total division time (ns.)	26.3×19 = 500	29.8×19 = 570
Gates (thousands)	4.7	3.8

Two adder scheme

Since in the design without assimilation (the fastest) the most time-consuming and gate-expensive part is the quotient-digit selection function, we consider another design in which the quotient-digit selection is divided into two parts: sign and magnitude. The sign is simple to compute and is used to generate two conditional partial remainders. The correct partial remainder is selected when the magnitude of the quotient-digit is known. This design offers a faster division and simpler selection at the cost of an extra 3-2 carry-save adder.

This results in a cycle time of 24.7 nsec at 4626 gates. The total division takes 470 nsec. But, as we shall see, the radix-4 carry-save adder approach produces approximately the same cycle time with fewer gates.

4. Implementation using a radix-4 carry-save adder

One of the reasons for the cost and delay of the quotient-digit selection (without assimilation) is that the number of variables of the function is large (12). To reduce this number we now represent the partial remainder in a radix-4 carry-save form. That is, the partial remainder is

$$\sum_{i=0}^{15} x_i 4^{-i} \quad where \; x_i = PS_i + SC_i \quad with \quad PS_i = \{0, 1, 2, 3\} \quad \text{and} \quad SC_i = \{0, 1\}$$

In this form, each of the radix-4 carry-save digits is represented by three bits -- two for PS and one for SC, which is the incoming carry from the previous CSA.

Several modifications with respect to the radix-2 carry-save adder are needed. We now describe the adder, the scaling, and the quotient-digit selection.

The adder

The adder is now composed of 2-bit binary adders (with carry-in and carry-out), as shown in Figure 4a. The partial remainder represents one of the inputs plus the carry in, while the divisor multiple is the other. We can take advantage of the early availability of the partial remainder and the timing characteristics of the tri-state gates to perform some time minimization. A 2-bit slice of the adder in which the inputs labeled "a" are from the quotient-digit multiplexer is shown in Figure 4b. The critical path for the adder is

$$\text{XOR}(6) + \text{XOR}(4) = 2.6 + 2.5 = 5.1 \text{ nsec}$$

Scaling

The scaling requires the addition of three multiples of the divisor or dividend. Since, in contrast with the radix-2 carry save adder, the adder now has only two inputs: one binary and the other radix-4 carry-save, it is necessary to pre-add two of the multiples to produce a radix-4 carry-save representation. To do this, the two lines from the multiplexer of Figure 1 are passed through 16 2-bit adders, as shown in Figure 5.

Quotient-digit selection

The quotient-digit selection function uses 8 bits of the radix-4 carry-save partial remainder, as shown in Figure 6. In terms of these 8 bits we define

$$z = [32a + 16(b_0 + b_1) + 8c + 4(d_0 + d_1) + 2e + f] \; mod \; 64$$

Since $z/8$ is the 2's complement representation of the estimate $\hat{w}$, that is

$$\hat{w} = \begin{cases} z/8 & \text{if } z < 32 \\ (z-64)/8 & \text{if } z \geq 32 \end{cases}$$

the quotient-digit selection function of Section 2 becomes

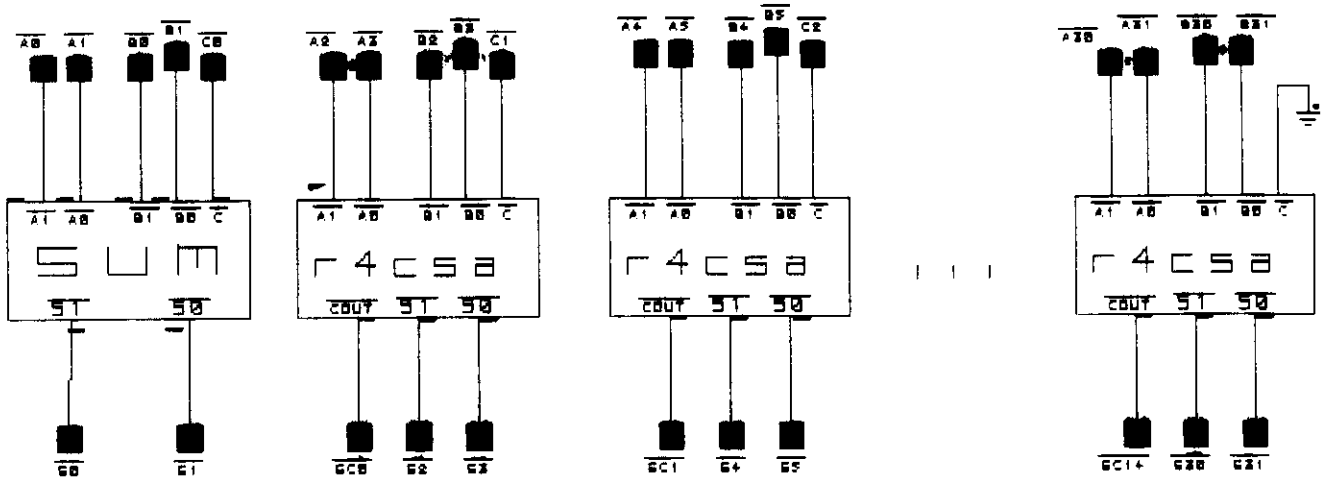

Figure 4A. 32 bit radix-4 CSA

~S0 ($1N172)

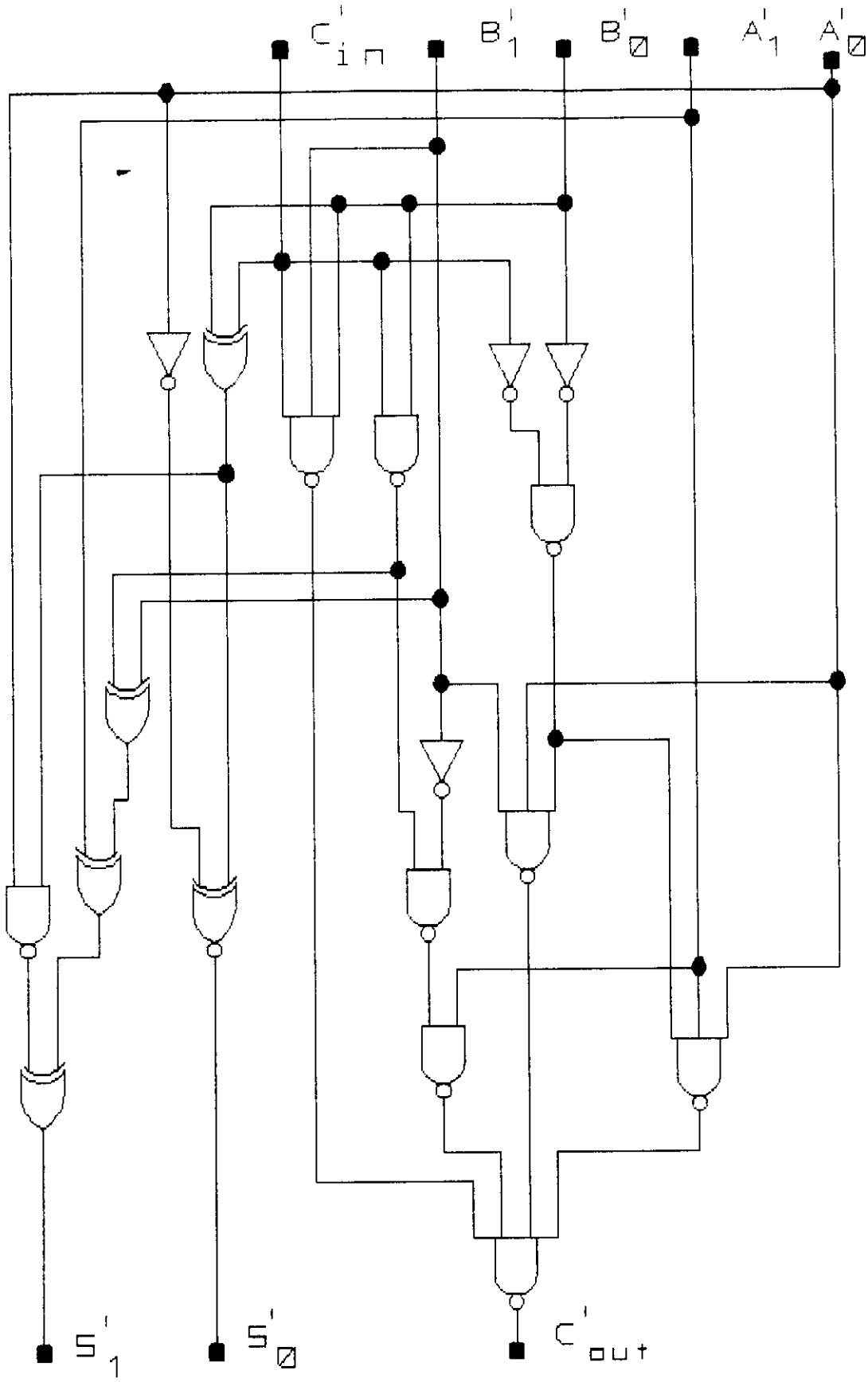

Figure 48. Bit slice of radix-4 CSA

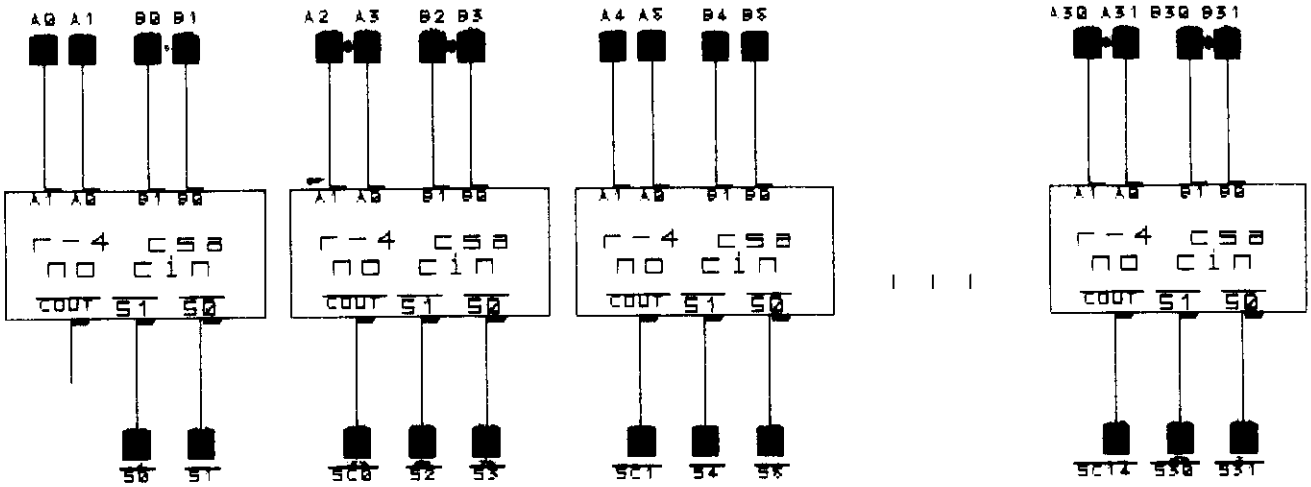

Figure 5. Scaling Adder for radix-4 CSA Scheme

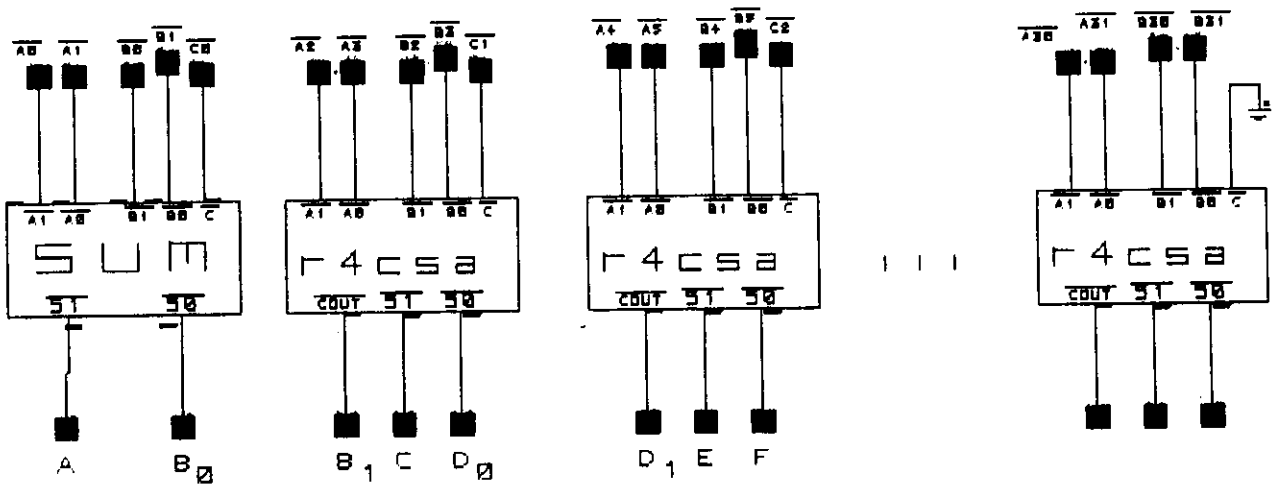

Figure 6. Inputs for Quotient Selection Func.

$$q_j = \begin{cases} 2 \text{ if } 12 \le z \le 24 \\ 1 \text{ if } 4 \le z \le 11 \\ 0 \text{ if } 0 \le z \le 3 \text{ and } 60 \le z \le 63 \\ -1 \text{ if } 51 \le z \le 59 \\ -2 \text{ if } 40 \le z \le 50 \end{cases}$$

with $25 \le z \le 39$ as don't cares, since the maximum value of the shifted partial remainder is 3.

This results in the following switching expressions:

$$(q_j{=}2) = a'b_0'b_1'c(d_0 + d_1) + ab_0b_1c(d_0 + d_1) + a'b_0'b_1 + a'b_0b_1'$$

$$(q_j{=}1) = a'b_0'b_1'(cd_0'd_1' + c'd_0 + c'd_1) + ab_0b_1(cd_0'd_1' + c'd_0 + c'd_1)$$

$$(q_j{=}0) = b_0'b_1'c'd_0'd_1' + ab_0b_1c'd_0'd_1' + a(b_0'b_1 + b_0b_1')(cd_0 + cd_1)$$

$$(q_j{=}{-}1) = a(b_0'b_1 + b_0b_1')(cd_0'd_1' + c'd_0 + c'd_1 + c'ef) + a'b_0b_1cd_0d_1ef$$

$$(q_j{=}{-}2) = a(b_0b_1' + b_0'b_1)(c'd_0'd_1')(e' + f') + (ab_0'b_1' + a'b_0b_1)(cd_0d_1ef)'$$

The implementation for the quotient values $(q_j = 2)$, $(q_j = 1)$, and $(q_j = 0)$ is straightforward and can be done using two levels of NAND gates. The implementation for the other two quotient-digit values is slightly more complicated and is as follows:

$$(q_j = -1) = (b_0 \oplus b_1)acd_0'd_1' + (b_0 \oplus b_1)ac'd_0 + (b_0 \oplus b_1)ac'd_1 + (b_0 \oplus b_1)ac'ef + a'b_0b_1cd_0d_1ef + S$$

$$(q_j = -2) = (ef)'ab_0b_1'c'd_0'd_1'S' + (ef)'ab_0'b_1c'd_0'd_1'S' + ab_0'b_1'c'S' + a'b_0b_1c'S' + (d_0d_1ef)'ab_0'b_1'S' + (d_0d_1ef)'a'b_0b_1S'$$

where $S{=}1$ during the scaling cycles.

Each of the values $(q_j = 2)$, $(q_j = 1)$, and $(q_j = 0)$ is implemented by two levels of NAND gates. The first level consists of six 8-input NANDs each with a fanout of eight; the second level consists of eight 6-input NAND gates with fanout of eight (to drive the tristate gates). The value $(q_j = -1)$ has a first level of one EXOR gate with fanout 4, a second level of 6-input NAND gates with fanout 8 and a third level with one 8-input NAND gate. Finally, the value $(q_j = -2)$ uses three levels of NAND gates: a first level of 4-input gates, a second level of 6-input gates, and a third level of 6-input gates.

The critical path of the quotient-selection part is of 11.1 ns. and corresponds to the $(q_j = -1)$ value.

The critical path components are shown in Table 3a, resulting in an overall cycle of 23.3 ns. Since 19 cycles are required (3 for the scaling and 16 for the recurrence) the total division time is $19 \times 23.3 = 443ns$. The gate counts are given in Table 3b.

Table 3a. Critical path for the radix-4 carry-save alternative

Component	Delay (ns)
Propagation delay of FF	2.1
Inverter and Buffer	0.9
Quotient-digit selection	11.1
Multiple selection	2.6
Radix-4 carry-save adder	5.1
Set-up delay of FF	1.5
TOTAL	23.3

Table 3b. Number of gates for the radix-4 carry-save alternative (in thousands)

Component	A	B	C	D	E	F	G	Total
Gates	1.4	0.4	0.5	0.5	0.9	0.4	0.2	4.3

In summary, the radix-4 carry-save alternative has the following characteristics:

Cycle time	23.3 ns.
Total division time	443 ns.
Number of gates	4.3 thousands

5. Other three cases

The other three alternatives mentioned in Section 2 produce implementations that are slower than the previous one. Consequently, we discuss them only briefly.

A) 0-1-2 adder

In the radix-2 carry-save adder case, the first step in the quotient-digit selection is to decode the pairs of bits of the partial remainder into the three values 0, 1, and 2. In this alternative, we include this decoding as part of the adder. That is, the partial remainder is stored in the 0, 1, 2 code and the input and output of the adder is in this representation.

This variation reduces the quotient-digit selection function by 1.6 nsec. (since the decoding is replaced by an inverter). However, the adder is more complex and its delay increases from 4.2 ns. to 8.5 ns. Consequently, the cycle time increases from 26.3 to 29.1.

B) Radix-2 signed-digit adder.

In this alternative we use a radix-2 signed-digit adder. A design for this adder is described in [KUNN87]. The needed modules were slightly modified for the buffering requirement of our design and the gate delay characteristics of LSI LOGIC. The resulting adder delay is 5.2 ns.

As indicated in Section 2, the quotient-digit selection function in this case uses 5 signed digits, that is, 10 bits. The implementation is made up of 14 product terms, with 4 product terms involving 5 variables, 8 product terms involving 3 vars., and 2 product terms involving 3 vars. The difference in the number of variables permits the use of various gate sizes for the second level, allowing us to reduce the time from a potential of 11.7 nsec to 10.5 nsec. However, this reduction in time causes an irregular gate layout. The critical path is shown in Table 4.

Table 4. Critical path for the radix-2 signed-digit alternative

Component	Delay (ns.)
Flip-flop propagation	2.3
ND2(3) + B2A(inverter)(FO = 8)	3.4
Quotient-digit selection	10.5
3-state gate(3)	2.2
Signed-digit adder	5.2
Flip-flop setup time	1.5
TOTAL	25.1

C) Radix-4 signed-digit adder

In this case the representation of the partial remainder is in radix-4 signed-digit. This simplifies the quotient-digit selection implementation, since fewer bits are required. However, the adder is significantly more complex and outweighs the savings. The quotient-digit selection takes 10.9 ns. Since this is larger than the quotient-digit selection for the radix-4 carry-save case and the adder is also more complicated, this alternative is worse than the radix-4 carry-save one.

The design of the quotient-digit selection follows. Assume that each digit of the partial remainder is coded with three bits (s_3,s_2,s_1)in a sign-and-magnitude code. The quotient-digit selection uses three digits a, b, and c. The corresponding estimate of the partial remainder $\hat{w}$ is computed as

$$\hat{w} = (8-16a_3)(2a_2+a_1) + (2-4b_3)(2b_2+b_1) + (1-2c_3)c_2$$

The quotient-digit selection function is

$q=2$ for $\hat{w} \geq 13$

$q=1$ for $12 \geq x \geq 4$

$q=0$ for $3 \geq x \geq -3$

$q=-1$ for $-4 \geq x \geq -12$

$q=2$ for $-13 \geq x$

The resulting switching expressions are

$$(q=1) = a_3'a_2a_1'b_3b_2(b_1+c_3+c_2') + a_3'a_2'a_1b_3'b_1'(c_3+c_2') + a_3'a_2'a_1b_3'b_2' + a_2'a_1'b_3'b_2b_1 + a_2'a_1'b_3'b_2(c_3 + c_2')$$

$$(q=2) = a_3'a_2a_1 + a_3'a_2b_3' + a_3'a_2b_3b_2' + a_3'a_2b_3b_1'c_3'c_2$$

Note that $(q=-2)$ and $(q=-1)$ are symmetric to $(q=2)$ and $(q=1)$. Moreover, $(q=0)$ does not need to be implemented.

Notice that the $(q=1)$ expression has 9 product terms in it and can not be implemented in two levels. The implementation will be similar to that of the r-2 CSA and the 0-1-2 approach. The 1st nand level is an 8-input NAND gate with FO=1. The second level is an equivalent 9-input NAND gate and is realized using 3 levels, as follows:

3-input NAND (FO=4), inverter/buffer (FO=32), and 3-input NAND (FO=2)

The delay of quotient-digit selection is

3.7 + (2.8 + 2.3 + 2.1) = 10.9 nsec

Table 5 shows the critical path.

Table 5. Critical path for the radix-4 signed-digit alternative

Component	Delay (ns.)
Flip-flop propagation	2.1
B2A(inverter)(FO = 16)	1.5
Quotient-digit selection	10.9
3-state gate(4)	2.6
Signed-digit adder	?
Flip-flop setup time	1.5
TOTAL	18.1 + ?

Since the adder is more complex than the radix-2 signed-digit adder, this alternative is not attractive.

6. Comparisons and Conclusions

Table 6 gives the summary of the schemes presented and compares them with the radix-2 division and with the radix-4 without scaling (for the implementation see Appendix). We conclude that the fastest scheme (with radix-4 carry-save adder) produces a speed up of 1.5 with respect to the radix-2 divider with an increase in the number of gates by a factor of 2.5. In contrast, the radix-4 divider produces a speed-up of 1.25 with a gate factor of 1.9. Consequently, the scaling approach provides a significant speed-up.

Table 6. Comparison of schemes

Scheme	Division time (μs.)	No. of gates (thousands)	Speedup	Gate factor 1
Radix-2 divider				
Signed-digit adder	.67	1.7	1	1
Radix-4 dividers				
No scaling	.53	3.2	1.25	1.9
With Scaling				
r-2 carry-save	.5	4.7	1.3	2.7
r-2 carry-save(2 adder)	.47	4.6	1.4	2.7
0-1-2	.56	4.2	1.2	2.5
r-2 signed-digit	.48	4.4	1.4	2.6
r-4 carry-save	.44	4.3	1.5	2.5
r-4 carry-save(assim.)	.54	3.0	1.25	1.8

If the number of gates for the fastest scheme is too high, an alternative is to use a quotient-digit selection implementation that first assimilates the 6 bits of the partial remainder. This results in a speedup of 1.25 with a gate factor of 1.8.

Acknowledgment. We thank Viewlogic Corporation for providing us with their gate-array design tools.

References

[ERCE87a] M.D. Ercegovac and T. Lang, "Simple Radix-4 Division with Divisor Scaling", UCLA CSD Internal Report, March 1987.

[ERCE87b] M.D. Ercegovac and T. Lang, "On-the-Fly Conversion of Redundant into Conventional Representation", IEEE Trans. on Computers, July 1987, pp. 895-897.

[ERCE88] M.D. Ercegovac and T. Lang, Notes for CS 252A: Arithmetic Processors, Computer Science Dept., UCLA, 1988.

[KUNN87] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi and N. Takagi, "Design of High Speed MOS Multiplier and Divider Using Redundant Representation", IEEE Proceedings 8th Symposium on Computer Arithmetic, 1987, pp. 80-86.

[LSI85] LSI Logic Corporation, CMOS Macrocell Manual, 1985.

Appendix: Radix-2 divider and Radix-4 divider without scaling

A) Radix-2 divider

We now implement a radix-2 divider as a reference for the comparisons. This divider is a standard SRT divsion (with redundant adder). We perform several designs (for different representations for the redundant partial remainder) and select the fastest for the comparisons.

i) With 0-1-2 representation

The adder is the same as the 0-1-2 adder in the radix-4 divider. The quotient-digit selection uses 3 digits of the partial remainder. The minimization using espresso results in 9 product terms. We implement the function using two levels of NAND gates (4-input NAND and then 6-input NAND) and then use an inverter/buffer to drive the 32 3-state gates from "QD". The delay is (1.7 + 3.5 + 2.3) = 7.5 ns.

The critical path is

Component	Delay (ns.)
f/f propagation	3.1
B2A(inverter)(FO = 4)	1.0
Quotient-digit selection	7.5
Tri-state(3)	2.2
0-1-2 Adder	8.5
f/f setup time	1.5
TOTAL	23.8

The resulting division time is 762 ns. and the number of gates 1.7 thousands.

ii) Carry-Save Representation

The adder is the same as the r-2 carry-save adder used in the radix-4 division. For the quotient-digit selection, as done in the r-4 division, we implement a pairing level, and then use the same selection as used for the 0-1-2 case. The delay is 7.5 + 2.8 = 10.3 ns.

The critical path is

Component	Delay (ns.)
f/f propagation	3.1
B2A(inverter)(FO = 8)	1.2
Quotient-digit selection	10.3
Tri-state(3)	2.2
Adder	4.2
f/f setup time	1.5
TOTAL	22.5

The division time is 720 ns. and the number of gates 1.8 thousands.

iii) Signed-Digit Representation

The adder is the same as the one used in the signed-digit representation for the radix-4 division. The quotient-digit selection uses two levels of 2-input NAND gates (with loads 1 and 3), a buffer with load 32, and a buffer with load 4. The delay is 1.7 + 2.4 + 1.0 + 2.3

The critical path is

Component	Delay (ns.)
f/f propagation	2.1
B2A(inverter)(FO = 8)	1.2
Quotient-digit selection	7.4
Tri-state(3)	2.2
Signed-digit Adder	6.2
f/f setup time	1.5
TOTAL	20.6

The division time is 670 ns. and the number of gates 1.7 thousands.

B) Radix-4 divider without scaling

We now discuss the implementation of a radix-4 divider without scaling, for comparison.

We select an implementation using a radix-4 carry-save adder, since this design simplifies the quotient-digit selection function.

The quotient-selection function is described by the following Table [ERCE88]

$[d_i, d_{i+1})$	8/16 9/16	9/16 10/16	10/16 11/16	11/16 12/16	12/16 13/16	13/16 14/16	14/16 15/16	15/16 16/16
$L_2(d_{i+1}),\hat{U}_1(d_i)$	3/4, 37/48	5/6, 7/8	11/12, 47/48	1, 13/12	13/12, 57/48	7/6, 31/24	15/12, 67/48	4/3, 72/48
$c_i(2)$	3/4	7/8	15/16	1	9/8	19/16	5/4	6/4
$L_1(d_{i+1}),\hat{U}_0(d_i)$	3/16, 13/48	5/24, 15/48	11/48, 17/48	1/4, 19/48	13/48, 21/48	7/24, 23/48	5/16, 25/48	1/3, 27/48
$c_i(1)$	1/4	1/4	1/4	1/4	3/8	3/8	1/2	1/2
$L_0(d_i),\hat{U}_{-1}(d_{i+1})$	-1/3, -1/4	-3/8, -13/48	-5/12, -7/24	-11/24, -5/16	-1/2, -1/3	-13/24, -17/48	-7/12, -3/8	-5/8, -19/48
$c_i(0)$	-1/4	-3/8	-3/8	-3/8	-1/2	-1/2	-1/2	-1/2
$L_{-1}(d_i),\hat{U}_{-2}(d_{i+1})$	-5/6, -13/16	-15/16, -43/48	-25/24, -47/48	-55/48, -17/16	-15/12, -55/48	-65/48, -59/48	-35/24, -21/16	-75/48, -67/48
$c_i(-1)$	-13/16	-15/16	-1	-9/8	-5/4	-5/4	-11/8	-6/4

The table shows the quotient selection function as a function of the three most significant bits of the divisor and the seven most-significant bits of the assimilated partial remainder. The quotient-digit selection is divided into three logical units

1) A 3-to-8 decoder is used to determine the correct divisor interval. This decoder is not in the critical path.

2) Production of the seven bit estimate of the partial remainder. The delay of this is 8.9 ns. This delay is relatively small because of the radix-4 carry-save representation.

3) Implementation of each of the eight ranges described in the table and anding the product term with the corresponding range of the divisor. Using espresso we came up with a maximum of 24 product terms per quotient-digit value and eight literals per product term. The delay of this is 11.2 nsec.

The critical path is

Component	Delay (ns)
Propagation delay of FF	2.1
Buffer	1.5
Quotient-digit selection	20.1
Multiple selection	2.6
Radix-4 carry-save adder	5.1
Set-up delay of FF	1.5
TOTAL	33.1