Fast Radix-4 Retimed Division with Selection by Comparisons *

Elisardo Antelo[1], Tomás Lang[2], Paolo Montuschi[3] and Alberto Nannarelli[4]

[1]Dept. Electrónica e Computación.
Universidade de Santiago – Spain. elisardo@dec.usc.es.

[2]Dept. Electrical and Computer Engineering.
University of California at Irvine – USA. tlang@uci.edu.

[3]Dept. di Automatica e Informatica.
Politecnico di Torino – Italy. montuschi@polito.it

[4]Dipartimento di Ingegneria Elettronica.
Universitá di Roma, Tor Vergata – Italy. nannarelli@ing.uniroma2.it

Abstract

Since a large portion of the critical path in an implementation of radix-4 division corresponds to the delay of the quotient-digit selection module, it is of interest to reduce this delay. The proposal of this paper extends the approach presented recently of prestoring the selection constants corresponding to the actual value of the divisor and to perform the determination of the quotient digit by carry-free subtraction and sign detection. This extension consists in advancing the subtraction so that it is outside of the critical path. This advancement also provides the possibility of placing the registers so as to minimize the cycle time. We present the method and report results of synthesis using a family of standard cells. We conclude that the extension results in a speedup of 1.35 with respect to the basic implementation and of 1.3 with respect to the previously mentioned approach. We estimate that the areas of all three units are about the same.

1: Introduction

Digit-recurrence division is an algorithm in which the quotient is obtained one digit per iteration. This algorithm has been extensively studied (for additional references see for instance [1]) and provides good tradeoff among latency, area, and power, allows simple exact rounding, and does not introduce overhead on the floating-point multiplier. It has been implemented in many high-performance floating-point units for general-purpose and application-specific processors, such as processors for 3D graphic applications ; some recent examples are described in [2, 3, 4, 5].

Research of E. Antelo is partially supported by the Xunta de Galicia under project PGIDT99XI20601A and of T. Lang by UC MICRO Grant 01-047.

The determination of the quotient digit is performed by the quotient-digit selection function. The prevalent method used today for this quotient-digit selection is to use selection constants and implement it by a combinational network having as inputs a truncated redundant residual and a truncated divisor. Because of the delay/complexity of this network, this approach is practical for relatively small radices, mainly radix 4 and possibly up to radix 8. An alternative implementation is based on comparisons of the residual estimate with truncated multiples of the divisor; however, this implementation is rarely used in practice because it requires assimilation of the truncated redundant residual and comparison, so no advantage is obtained with respect to the implementation with selection constants.

Recently [6], a scheme was described that uses the selection function with selection constants but implements it with comparisons [1]. Specifically, since the divisor is fixed during all the iterations, the selection constants corresponding to the value of the truncated divisor are preloaded into registers and the quotient-digit selection is performed by comparing the truncated residual with the selection constants. To reduce the delay, the comparisons are implemented by a carry-free subtraction of the selection constants from the redundant residual and then performing a sign detection. From the synthesis we performed we estimate that for radix 4 this results in a delay reduction of about 5% with respect to the direct implementation. In this paper, we propose to utilize this decomposition of the comparison into redundant subtraction and sign detection and to retime the recurrence so that the subtraction is not part of the critical path. We have estimated the resulting delay and obtained an additional 30% reduction. Moreover, we estimate that the area of the three implementations is almost the same.

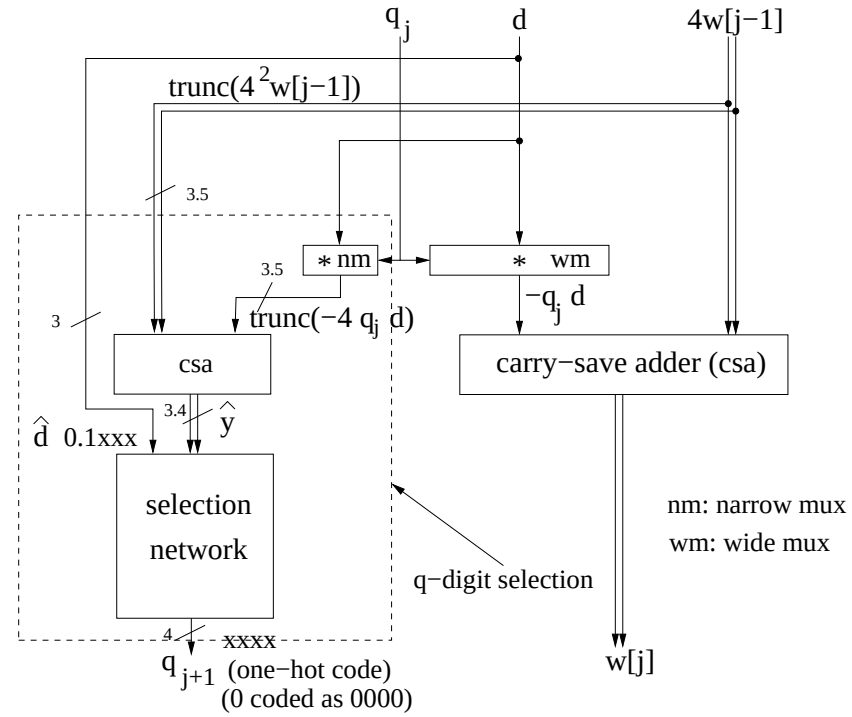
In the next sections we present the modified algorithm, discuss the timing alternatives for minimal delay, and do and estimation of the delay.

2: Proposed selection function implementation

Digit-by-digit radix-r division algorithms compute the final result by performing a suitable number of iterations, each one consisting of the following phases:

1. **Quotient-Digit Selection**: based on the value of the current residual and of the divisor one digit of the quotient is produced. If a redundant digit set is used for the quotient, estimates of the shifted residual (denoted by $\widehat{y}$) and of the divisor ($\widehat{d}$) can be used, resulting in
$$q_{j+1} = sel(\widehat{y}, \widehat{d})$$
The prevalent method to perform the quotient-digit selection uses selection constants $m_{i,k}$ such that
$$q_{j+1} = k \text{ if } \hat{d} = 2^{-1} + i \times 2^{-\delta} \text{ and } m_{i,k} \leq \widehat{y} < m_{i,k+1} \text{ for } 0 \leq i \leq 2^{\delta-1} - 1$$
where $-a \leq k \leq a$, $\hat{d}$ is the truncated divisor with δ fractional bits, and $\widehat{y}$ is the truncated shifted residual ($rw[j]$) with t fractional bits.
2. **Residual Updating and Quotient Formation**: using the digit selected at step 1, the next residual and quotient are produced, as follows:
$$w[j+1] = rw[j] - q_{j+1}d$$

[1] This scheme is similar to the use of truncated multiples of the divisor, but the use of selection constants as comparison points gives more flexibility, which is reflected in a somewhat simpler implementation.



a.b means a integer and b fractional bits.

Figure 1. Retimed radix-4 iteration.

$$Q[j+1] = Q[j] + q_{j+1} r^{-(j+1)}$$

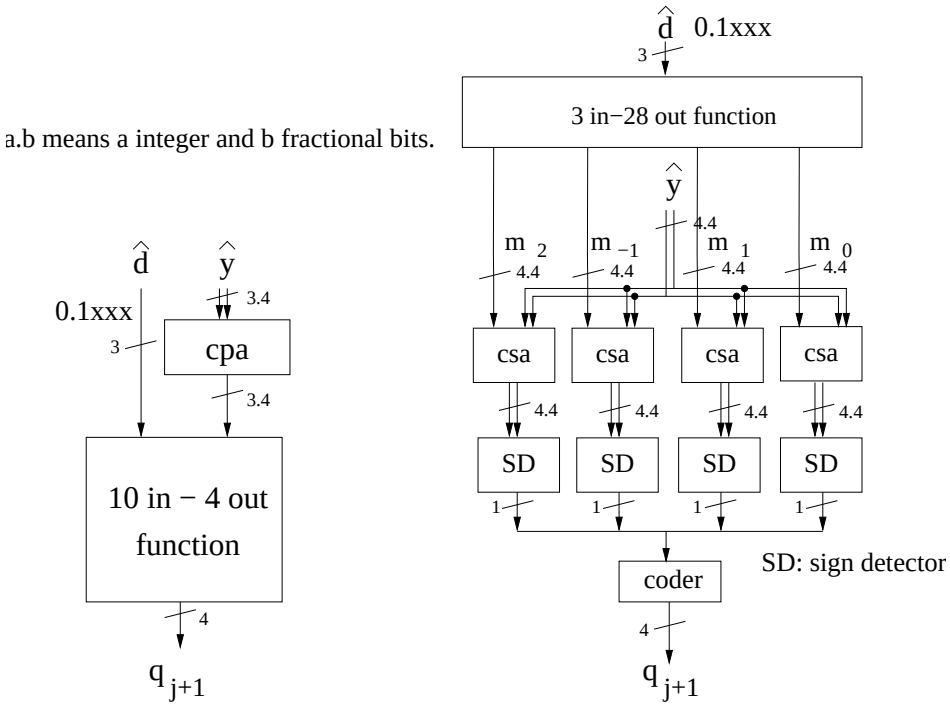
To reduce the delay of an iteration a redundant adder (carry-save or signed digit) is used for this residual updating. Moreover, the conversion of the quotient to conventional representation can be done on-the-fly [1].

The implementation of the recurrence for radix-4 is shown in Figure 1. Note that we present a retimed version in which the quotient-digit selection is placed at the end of the iteration since this constrains the critical path to the most-significant slice, which can be designed for smaller delay. This retiming was used in [7], where it is called model division with prediction, and in [8], to reduce the energy dissipation. The retimed version allows us to speed up the scheme proposed in [6].

Although it is possible to define correct digit selection rules for any radix, the approach is practical only for relatively low radices, such as 2, 4, and 8, because for higher radices the resulting implementation has a large area and delay. For this reason, for higher radices alternative techniques have been studied, such as multistage algorithms and/or prescaling.

The prevalent implementation of the selection function above consists of a combinational network (multilevel network, PLA or ROM) having as inputs the estimate of the residual and of the divisor and producing the quotient digit. For instance, for radix 4 with a quotient digit-set $\{-2, -1, 0, 1, 2\}$, 7 bits of $\hat{y}$ and 3 bits of $\hat{d}$ are sufficient (see Figure 1). Typically, the selection network is implemented as shown in Figure 2a) and its delay corresponds to about 50% of the iteration delay.

Recently [6] a scheme was described to implement the quotient-digit selection by preloading the selection constants corresponding to $\hat{d}$ (which does not change for the whole oper-



(a) Conventional implementation. (b) A variation of [6].

Figure 2. Previous implementations of selection network.

ation) and performing the selection using comparisons[2] (see Figure 2b). Specifically, the algorithm consists in

1. Preloading the selection constants. Let us call the preloaded constants[3] m_k.
2. Perform the comparison in two steps:
   (a) produce $z_k = \widehat{y} - m_k$. This subtraction is done with redundant adders.
   (b) perform the comparison by a sign detection of z_k.

The implementation in [6] includes a "compression" module; this is due to the use of signed-digit representation for the residual and is not needed for carry-save representation.

Based on our synthesis results (see Section 4) this scheme does not produce a significant speed up as compared to the basic algorithm.

Proposed implementation

To reduce the delay of the scheme proposed in [6] we retime the subtraction of m_k so that it is outside of the critical path. Note that this is not possible in a standard non retimed recurrence as used originally in [6].

Specifically, since

$$w[j] = rw[j-1] - q_j d$$

[2] Actually the scheme proposed in [6] was implemented in a standard non retimed recurrence.

[3] That is, the set of constants m_k are obtained from the set $m_{i,k}$ for the value of i corresponding to the actual divisor.

we have[4]

$$\widehat{y} - m_k \quad = \quad \lfloor rw[j] \rfloor_t - m_k = \left\lfloor r^2 w[j-1] - r q_j d \right\rfloor_t - m_k$$

Since we use carry-save representation, the previous computation is performed in two separate truncation steps with one additional fractional bit

$$\widehat{y} - m_k = \left\lfloor \lfloor r^2 w[j-1] \rfloor_{t+1} + \lfloor -r q_j d \rfloor_{t+1} \right\rfloor_t - m_k$$

that is, using the truncated $r^2 w[j-1]$ and the truncated $-r q_j d$. Moreover, since m_k has t fractional bits we may perform the computation as follows

$$\widehat{y} - m_k = \left\lfloor \left[\lfloor r^2 w[j-1] \rfloor_{t+1} - m_k \right] + \lfloor -r q_j d \rfloor_{t+1} \right\rfloor_t \tag{1}$$

In this way the computation of $\lfloor r^2 w[j-1] \rfloor_{t+1} - m_k$ is outside of the critical path.

2.1: Number of bits

The number of fractional bits of the various signals is as described by expression (1). We now consider the number of integer bits, which corresponds to the maximum value of $|\widehat{y} - m_k|$. Since,

$$m_- = m_{\max(i),-(a-1)} \leq m_{i,k} \leq m_{\max(i),a} = m_+$$

from [1], these maximum values are

$$\max(|\widehat{y} - m_k|) = \max(\lfloor r\rho \rfloor_t + |m_-|, \ |\lfloor -r\rho - 2^{-t} \rfloor_t - m_+|)$$

Radix-4 minimally redundant digit set

Since we obtain the same $\widehat{y}$ as in the standard implementation, we can use the selection function described in [1]. For $r = 4$, $\rho = a/(r-1) = 2/3$ and $t = 4$ we get

$$-43/16 - 11/8 = -65/16 \leq \widehat{y} - m_k \leq 42/16 + 11/8 = 64/16$$

This means that 4 integer bits are necessary for performing a correct sign comparison, i.e. one more than the number of integer bits required to represent $\widehat{y}$ in the "classical" algorithm.

Reduction in the number of bits of $\widehat{y} - m_k$

According to the discussion above, the number of bits of $\widehat{y} - m_k$ is of four integer bits and four fractional bits, for a total of eight bits. We have performed a more detailed analysis to reduce this number to six. This reduction is based on the following considerations:

1. As shown in Table 1, for some ranges of values of $\widehat{y}$ some of the outputs of the SD modules are not used in the determination of the quotient digit. In the Table we denote by $[L, \ H]$ the interval of values of $\widehat{y}$ and by x the dont cares. Note that the values of H, L, and m_k depend on $\widehat{d}$.

[4] $\lfloor a \rfloor_t$ means truncation of a to t fractional bits.

Table 1. Reduction of number of bits.

$\widehat{y}$	SD_2	SD_1	SD_0	SD_{-1}	
$[m_2, H]$	+	+	+	x	$q=2$ if $(SD_2, SD_1) = (+,+)$
$[m_1, m_2)$	−	+	+	x	$q=1$ if $(SD_2, SD_1) = (-,+)$
$[m_0, m_1)$	x	−	+	x	$q=0$ if $(SD_1, SD_0) = (-,+)$
$[m_{-1}, m_0)$	x	−	−	+	$q=-1$ if $(SD_0, SD_{-1}) = (-,+)$
$[L, m-1)$	x	−	−	−	$q=-2$ if $(SD_0, SD_{-1}) = (-,-)$
$\widehat{y} - m_k$	$[m_1 - m_2, H - m_2]$	$[L - m_1, H - m_1]$	$[L - m_0, H - m_0]$	$[L - m_{-1}, m_0 - m_{-1}]$	
$\max(\lvert\widehat{y} - m_k\rvert)$	1.25	3.25	3.125	1.312	
Integer bits	2	3	3	2	

Table 2. Reduced number of bits.

k	-1	0	1	2
Integer + fractional= total	$2+4=6$	$3+3=6$	$3+3=6$	$2+4=6$

2. Considering the most positive and most negative values (not dont cares) we determine the maximum magnitudes of $\widehat{y} - m_k$. The corresponding values are given in the table (see details in [9]) together with the required number of integer bits.

With respect ot the number of fractional bits:

- For $\widehat{y} - m_2$, four bits are required because m_2 can take the value 15/16.
- For $\widehat{y} - m_1$, it might be possible to use three bits, because all m_1 are multiples of 1/8. However, this requires the use of $t = 3$ so it is necessary to determine whether constants which are multiples of 1/8 are still possible for $t = 3$. As described in [9], only for $\widehat{d} = 8/16$ this is not possible; however this can be solved by adding 1/16 to the corresponding $rw[j]$ and this addition can be achieved by augmenting the constant m_1 by 1/16 for that value of $\widehat{d}$.
- For $\widehat{y} - m_0$ also 3 fractional bits can be used since all m_0 are multiples of 1/8. Again this requires the addition of 1/16 to the constant m_0 for $\widehat{d} = 8/16$.
- For $\widehat{y} - m_{-1}$ four fractional bits are needed because m_{-1} can take the value $-15/16$.

In summary, Table 2 gives the required number of integer and fractional bits of $\widehat{y} - m_k$.

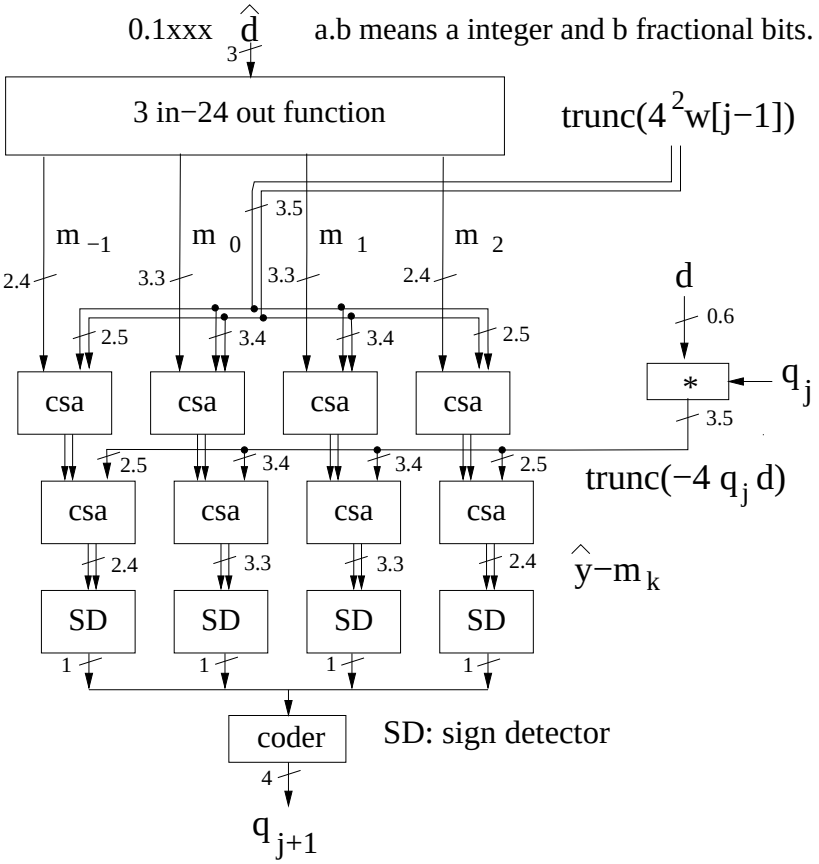
2.2: Implementation of the quotient-digit selection

The quotient-digit selection of this implementation (for radix-4) is shown in Figure 3. Note that the first subtraction does not depend on q_j, which is advantageous for the timing of the recurrence. As discussed in Section 3, the advancement of the subtraction is combined with an optimal placement of the registers to achieve the minimum cycle time.

3: Timing and Complete Algorithm

3.1: Timing

The iterations described in the previous section are executed either by a combinational network (if all iterations are unfolded) or by a sequential system, in which a part is executed



Sign extensions and truncations at the input of some modules.

Figure 3. Proposed q-selection implementation.

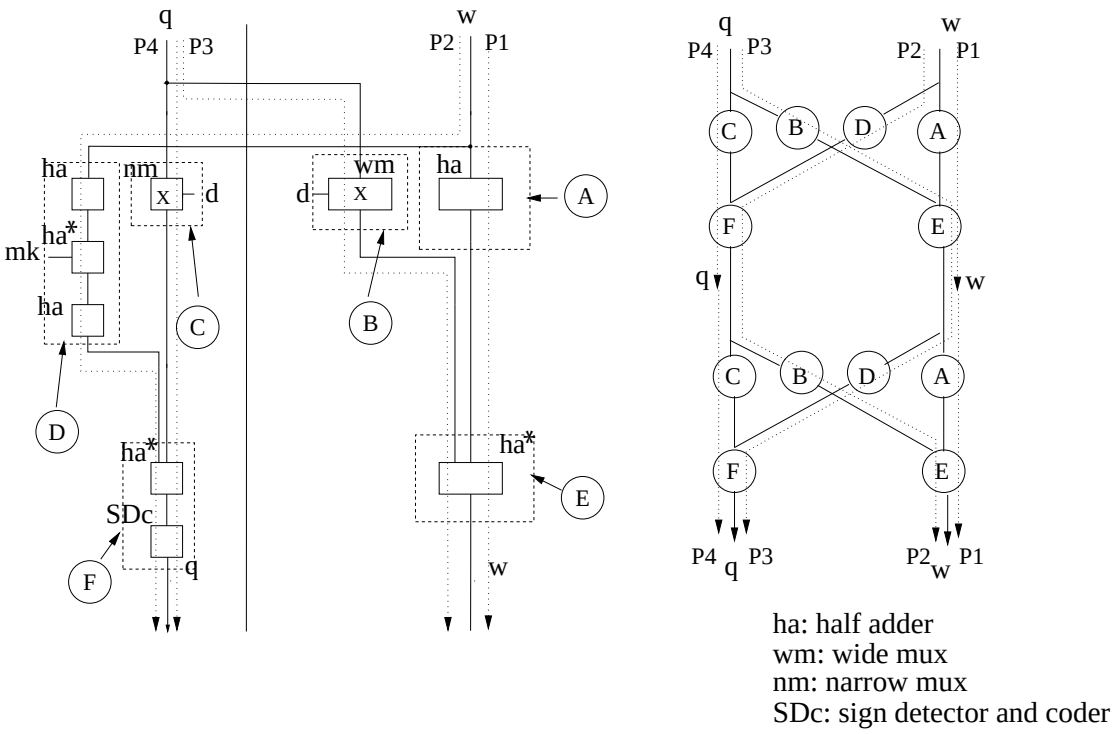
each cycle. We consider this second approach, for the case where one iteration is performed per cycle.

The corresponding sequential system requires the introduction of registers that store the state between cycles. The placement of these registers has an effect on the minimum cycle time and on the area and energy of the implementation. We describe now a method to analyze the cycle time for the particular algorithm of the previous section.

The process begins with a dependency graph of the algorithm. In Figure 4(a) we show this graph for two iterations[5], so that we can place the registers that define a complete cycle. We place registers in both iterations, but the corresponding registers of both iterations are implemented as a single register in the sequential system. To simplify the description of the analysis, we label the nodes A to F, resulting in the graph of Figure 4(b).

In the graph we identify four paths labeled P1 to P4. To produce a sequential system we have to introduce registers to cut these paths, once per iteration. Consequently, a maximum of four registers are needed. However, we see that the paths are not disjoint, so that one register can be used to cut more than one path. In this particular algorithm, paths P1 and P3 intersect at node E and paths P2 and P4 at node F. Therefore, there are the following four possibilities in the placement of registers:

[5]The full adders have been divided into two half adders so as to have a finer granularity in the timing analysis.



(a) Dependency graph. (b) High level view of dependency graph (two iterations

Figure 4. Dependency graph.

1. Two registers, one on node E and one on node F.
2. Three registers, one on node E, one on node C, and one on node D.
3. Three registers, one on node F, one on node A, and one on node B.
4. Four registers, one on each of the nodes A,B,C and D.

Now we consider the cycle time[6]. The minimum value is obtained by the maximum delay of all the paths between registers of the two iterations. Consequently, the minimum cycle time is obtained by minimizing this maximum delay.

A lower bound on the cycle time is given by the total length of the paths divided by two (since there are two iterations). We obtain

$$t_{cycle} \geq \max(A+E, 1/2(D+F+B+E), 1/2(B+E+D+F), C+F)$$

Note that the second and third components of this expression have the same value.

However, this bound cannot be always achieved, because of the restrictions on the placement of registers.

For reduced hardware complexity we restrict the discussion to the two-registers case. We now perform an analysis to obtain the conditions for minimum cycle time. We consider the possibility of partitioning the nodes and placing the register between these partitions.

[6]We do not include the register delay or set–up times at this point since these do not affect the placement of registers.

However due to the characteristics of each of the nodes, the partition is not convenient for all of them. The following considerations should be taken into account

- The nodes that can be considered for partitioning should have enough logical depth. In this sense the half adders (nodes A and E) and the decoded (buffered) multiplexers (nodes B and C) seem not suitable for partitioning. Then we only consider the potential partitioning of nodes D (partitions D1 and D2 so that D=D1+D2) and F (F=F1+F2).
- For two registers, since E is not partitioned, the register should be placed after this node (if it is placed before a total of three registers are necessary). Moreover, also for this case, it is not necessary to consider the partition of D, since the registers are placed in nodes E and F.

With these considerations we have the following paths between registers:

1. $R1 \rightarrow R1$: $F2 + C + F1 = F + C = t_{ha+}t_{SDc} + t_{nm}$.
2. $R1 \rightarrow R2$: $F2 + B + E = F2 + t_{wm} + t_{ha}$.
3. $R2 \rightarrow R1$: $D + F1 = 3t_{ha} + F1 = 3t_{ha} + F - F2 = 4t_{ha} + t_{SDc} - F2$.
4. $R2 \rightarrow R2$: $A + E = 2t_{ha}$.

At this point we can discard Path $R2 \rightarrow R2$ as critical, since it is composed only of two half adders. Therefore the problem consists in finding $F2$ (and then $F1$) so that the maximum of Path $R1 \rightarrow R1$, Path $R1 \rightarrow R2$ and Path $R2 \rightarrow R1$ is minimized. Since Path $R1 \rightarrow R1$ does not depend on $F1$ nor $F2$, we try first to minimize the maximum of Paths $R1 \rightarrow R2$ and $R2 \rightarrow R1$.

Path $R1 \rightarrow R2$ is a increasing function of $F2$ and Path $R2 \rightarrow R1$ is a decreasing function with the same variable. Therefore the minimum of the maximum of Paths $R1 \rightarrow R2$ and $R2 \rightarrow R1$ is obtained when Path $R1 \rightarrow R2$=Path $R2 \rightarrow R1$. Using this condition we obtain

$$F2 = \frac{3t_{ha} + t_{SDc} - t_{wm}}{2}$$

and then

$$\text{Path } R1 \rightarrow R2 = \text{Path } R2 \rightarrow R1 = \frac{5t_{ha} + t_{SDc} + t_{wm}}{2}$$

Note that F is composed of a half adder and the sign-detector-and-coder and that it is convenient to place the flip-flops so that the half adder is not partitioned. If $F2 > t_{SDc}$ it might be necessary to place the flip-flops inside the half adder to optimize the path. However this condition is satisfied only if $t_{SDc} + t_{wm} < 3t_{ha}$ which seems not to be true for practical implementations.

The critical path is $\max(\text{Path } R1 \rightarrow R1, \text{Path } R1 \rightarrow R2 = \text{Path } R2 \rightarrow R1)$, that is

$$tcycle = \max\left(t_{ha} + t_{SDc} + t_{nm}, \frac{5t_{ha} + t_{SDc} + t_{wm}}{2}\right)$$

Note that in case the maximum corresponds to Path $R1 \rightarrow R1$, the placement of registers does not affect the cycle time. In that case, the placement of registers might be guided by the reduction in the number of flip-flops, with the condition $\max(\text{Path } R1 \rightarrow R2, \text{Path } R2 \rightarrow R1) \leq \text{Path } R1 \rightarrow R1$ so that the cycle time remains the same.

3.2: Complete algorithm

We now describe the complete algorithm for radix 4 and quotient-digit set $\{-2, -1, 0, 1, 2\}$.

3.2.1: Initialization

The standard initial values are $w[0] = 2^{-2}x$ (to assure $w[0] \leq (2/3)d$) and $q[0] = 0$. Since this produces $2^{-2}q$, an additional iteration is required. The actual initialization depends on the location of registers (since these have to be initialized). For the two-register case, one possibility is

- $w[-1] = 2^{-4}x$
- $q_0 = 0$ (this requires that the register, which does not contain necessarily the q_j value[7], should be initialized so that $q_0 = 0$ is produced).

After this initialization, the first iteration produces $w[0] = 2^{-2}x$ and q_1 (or the value associated with q_1 in the corresponding register).

3.2.2: Iterations and Termination

To obtain the quotient of n radix-4 digits (including the rounding bit), it is necessary to compute $n+1$ digits (since, because of the initialization, the algorithm produces $2^{-2}q$). At the end of $n+1$ iterations, one register contains $w[n]$ and the other register a value "associated" with q_{n+1}. After that it is necessary to

- Obtain q_{n+1} and the last residual $w[n+1]$ (one cycle)
- Detect sign of residual, correct quotient, normalize, and round (one cycle).

The total number of cycles is then $n+3$. This number of cycles is the same as for the other retimed schemes.

4: Estimation and Comparison

In this section we estimate the execution time to obtain a normalized rounded quotient of 53 bits (double precision) using the proposed implementation of the selection function and compare it with previous radix-4 implementations. We also estimate and compare the cell areas.

For this estimate we have used a 0.35 μm standard-cells library with power supply of 3.3 V and Synopsys analysis and synthesis tools (Design Analyzer). We performed the estimation pre-layout, therefore, we do not take into account the additional delay due to the actual parasitic capacitance of interconnections. To make the estimation as independent as possible from the technology we also give the delay using as unit the delay of an inverter loaded with four inverters; we call this unit t_{std} and it is 0.15 ns for the library used.

As a first step we perform the timing of the recurrence. To do this, we use the delays of the modules[8] indicated in Figure 4 and given in Table 3. Using the procedure described in the previous section[9], we determine that the lower-bound of the cycle time (including register setup and propagation delay) is 2.40 ns and the registers should be placed as shown in the diagram of Figure 5.

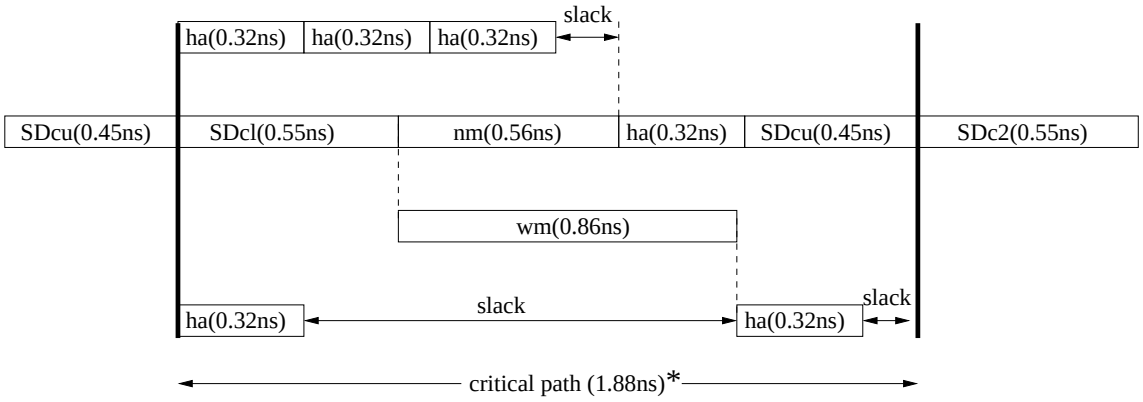
[7] Because of the placement of registers, as discussed in Section 3.1.

[8] These delays have been obtained by loading each module output with four inverters.

[9] We consider the two-register case since this achieves the bound of the critical path.

Table 3. Delay of basic blocks in the $0.35\ \mu m$ standard-cells library.

unit/gate	delay [ns]
half adder	0.32
sign detection and coding (6-bit)	1.00
wide mux (56-bit)	0.86
narrow mux (10-bit)	0.56
register (prop. delay)	0.42
register (set-up)	0.10



* register delay and set−up not included

ha: half−adder − SDc: Sign Detection and coder − wm: wide mux − nm: narrow mux

Figure 5. Placement of registers for optimal cycle time.

To have a more reliable estimate we have done a global synthesis of the whole unit. In the synthesis of the part which computes the quotient digit, we lumped the short mux, the two levels of carry-save adders and the upper part of the sign-detector-and-coder (see Figure 3), into one block to optimize logic synthesis across the original blocks boundaries. The resulting cycle time is now 2.42 ns. (see Table 4). The slight increase with respect to the cycle time obtained using the delays of Table 3 is due to the actual loads. In this global implementation we obtained an area of about 4780 nand2 equivalent gates.

Since the proposed algorithm requires (54/2)+3=30 cycles for double precision, the execution time is 73 ns. For the comparison, we synthesized the complete unit using the standard quotient-digit selection function and a variation of [6] using carry-save adders (Figures 2a and 2b). In these cases, the registers store directly $w[j]$ and q_{j+1} since there is no advantage in other placements. Again for best use of the synthesis tool we lumped together the modules that compose the quotient-digit selection.

We obtained the cycle times reported in Table 4. From the table we conclude that the proposed implementation produces a speedup of about 1.35 and that the implementation of Figure 2b (variant of [6] using carry-save adders and retimed recurrence) produces a small speedup with respect to the basic implementation. The areas are roughly the same in all schemes.

Table 4. Summary of results for the implemented units.

scheme	critical path		T_c [ns]	# t_{std}	speed-up	AREA
basic (Fig. 2a)	$t_{REG} + t_{nm} + t_{HA} + t_{CPA} + t_{SEL} + t_{su}$	=	3.25	22	1.00	4660
variant of [6] (Fig. 2b)	$t_{REG} + t_{nm} + t_{HA} + t_{CSA} + t_{SDc} + t_{su}$	=	3.10	21	1.05	4800
proposed (Fig. 3)	$t_{REG} + t_{SDc_l} + t_{nm} + t_{HA} + t_{SDc_u} + t_{su}$	=	2.42	16	1.34	4780

5: Conclusions

We have developed a scheme to reduce the cycle time of digit-recurrence division by retiming part of the quotient-digit selection so that it is not in the critical path. In addition, this modification allows the placement of the registers in a way to minimize the delay. Although the scheme has been developed for a general radix, all illustrations and evaluations have been done for radix 4. For this radix, we have estimated a speedup of about 35% with respect to the basic implementation and about a 30% with respect to the variation described in [6]. A natural extension would be to develop the details for higher radices.

We have only considered division but, as for the standard implementation, it can be combined with a similar scheme for square root.

References

[1] M. Ercegovac and T. Lang, "Division and Square Root: Digit-Recurrence Algorithms and Implementations", Kluwer Academic Publishers, 1994.

[2] T. Fu et al., "R18000: The latest SGI superscalar microprocessor", in Slides of presentations of Hot Chips 13, August 19-21, 2001.

[3] J. Clouser et al., "A 600 MHz superscalar floating-point processor", IEEE Journal of Solid–State Circuits, Vol. 32, no 7, pp. 1026-1029, 1999.

[4] M. Suzuoki, et al., "A microprocessor with a 128–bit CPU, ten floating–point MAC's, four floating–point dividers, and a MPEG2 decoder", IEEE Journal of Solid–State Circuits, Vol. 34, no. 11, pp. 1608–1618, 1999.

[5] G. Hinton, et al. "A 0.18 um CMOS IA–32 Processor with a 4–GHz Integer execution Unit", IEEE Journal of Solid–State Circuits, Vol. 36, no. 11, pp. 1617-1627, Nov. 2001.

[6] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit", Proceedings of the 35th Asilomar Conference on Signals, Systems and Computers, Nov. 2001, pp. 1646-1650.

[7] J.E. Robertson, "A new class of digital division methods", IRE Transactions on Electronic Computers, vol. 7, no. 3, Sep. 1958, pp. 88-92.

[8] A. Nannarelli and T. Lang, "Low-Power Divider", IEEE Transactions on Computers, vol. 48, no. 1, Jan. 1999, pp. 2-14.

[9] E. Antelo, T. Lang, M. Montuschi and A. Nannarelli, "Fast Radix–4 Retimed Division with Selection by Comparisons", Internal Report, Dept. Electrical and Computer Eng., University of California at Irvine, USA. (Available at http://www.eng.uci.edu/numlab/archive).