Digit-Recurrence Dividers with Reduced Logical Depth

Elisardo Antelo, *Member*, *IEEE*, Tomás Lang, *Member*, *IEEE Computer Society*, Paolo Montuschi, *Member*, *IEEE Computer Society*, and Alberto Nannarelli, *Member*, *IEEE Computer Society*

Abstract—In this paper, we propose a class of division algorithms with the aim of reducing the delay of the selection of the quotient digit by introducing more concurrency and flexibility in its computation. From the proposed class of algorithms, we select one that moves part of the selection function out of the critical path, with a corresponding reduction in the critical path compared with existing alternatives. We present the algorithm and describe the architectures for radix 4 and for radix 16. For radix 16, we use the scheme of overlapping two radix-4 stages. In both cases, radix 4 and radix 16, we show that our algorithms allow the design of units with well-balanced critical paths with consequent decreases of the cycle times. Moreover, in the radix-16 case, we include some additional speculation techniques. To estimate the speedup, we used a rough timing model based on logical effort. For both radices, we estimate a speedup of about 25 percent with respect to previous implementations. In the radix-4 case, this is achieved by using roughly the same area, while, in the radix-16 case, the area is increased by about 30 percent. We verified our estimations by performing a synthesis of the radix-4 units.

Index Terms—Digit-by-digit division, algorithms and architectures for computer arithmetic, division radix 4, division radix 16.

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 [2], chapter 5) and provides good trade-offs among latency, area, and power, allows simple exact rounding, and does not introduce overhead on the floating-point multiplier. Moreover, it fits the particular characteristics of integer division where the number of iterations is variable and, in many cases, a small number of iterations is needed. 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. It is also an SIP (Silicon Intellectual Property) core offered by ASIC design services. The current algorithms and implementations perform the determination of the quotient digits through the use of the quotient-digit selection function, implemented by the digit-selection module, whose inputs are the truncated divisor and the truncated residual (in carry-save or signed-digit representation [1], [2]). The quotient-digit selection algorithms used compare the truncated residual with truncated multiples of the divisor or with selection constants, this latter approach being more popular. In either case, the implementation can be table-based or comparison-based [3]. Because of the delay/complexity of this network, this method is practical for relatively small radices, i.e., up to radix 8. For radix 16, a successful scheme is to have two overlapped radix-4 stages, while, for higher radices, alternative techniques have been studied, such as multi-stage algorithms and/or prescaling [1] [2]. Several high-radix implementations are reported in [4]. Recent improvements and variations of the algorithm and its implementation have been proposed in [3], [5], [6], [7], [8].

In this paper, we propose a class of division algorithms with the aim of reducing the delay of the selection of the quotient digit by introducing more concurrency and flexibility in its computation. From the proposed class of algorithms, we select one that moves part of the selection function out of the critical path, with a corresponding reduction in the critical path compared with existing alternatives. We present the algorithm and describe the architectures for radix 4 and for radix 16. We have also done some designs and evaluations for the radix-8 case and have concluded that the increase in area is too high for the speedup obtained. For radix 16, we use the scheme of overlapping two radix-4 stages. In both cases, radix 4 and radix 16, we show that our algorithms allow the design of units with well-balanced critical paths with consequent decreases of the cycle times. Moreover, in the radix-16 case, we include some additional speculation techniques.

We concentrate on serial architectures, where the intermediate results are stored in registers and all the iterations are performed reusing the same hardware. This is the current practice for the division units in most of the systems. However, the same techniques can be used in an

- *E. Antelo is with the Departamento de Electrónica e Computación, Universidade de Santiago de Campostela, 15706 Santiago de Compostela, Spain. E-mail: elisardo@dec.usc.es.*
- *T. Lang is with the Department of Electrical Engineering and Computer Science, Engineering Tower, Room 602, University of California at Irvine, Irvine, CA 92697. E-mail: tlang@uci.edu.*
- *P. Montuschi is with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy. E-mail: paolo.montuschi@polito.it.*
- *A. Nannarelli is with the Department of Informatics and Mathematical Modeling, Technical University of Denmark, Richard Petersens Plads-Building 321, DK-2800 Kongens Lyngby, Denmark. E-mail: an@imm.dtu.dk.*

Manuscript received 16 Apr. 2004; revised 13 Dec. 2004; accepted 3 Feb. 2005; published online 16 May 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0131-0404.

unfolded and pipelined architecture (which may be of interest for high throughput applications such as high-performance 3D graphics).

For the exploration of the new algorithm alternatives (and the comparison with existing proposals), we needed a high-level delay model precise enough to lead to actual implementations with the expected speedup and a method to optimize the position and reduce the number of flip-flops of the architecture. For this purpose, we used a delay model based on logical effort [9]. This model was recently used with success to explore algorithmic high-level alternatives in the design of fast and low-power adders [10]. Since the delay model is rough and we are interested in the estimation of delay ratios, we do not take into account estimations of additional delay due to interconnections since the effect should be similar for all the architectures compared.

For minimal number and optimal placement of flip-flops, we use retiming and clock scheduling [11], [12] with the data provided by the rough delay model.[1] Retiming decreases the critical path by optimizing the position of registers so that the combinational paths are balanced. However, the resultant position might not be convenient in terms of number of flip-flops or the granularity of the modules.[2] Clock scheduling increases the slack along combinational critical paths by skewing the clock signal in the registers and allows the minimum number of flip-flops for a given granularity of the modules with the minimum delay.

For radix 4, we selected comparison with a standard implementation with digit set -2 to +2, as described, for instance, in [2] and [4], and with a recent implementation with digit set -3 to +3 [8]. For radix 16, we compare with the standard implementation using two overlapped radix-4 stages [2], [4], with the hybrid overlapped scheme, which was reported in [4] to be their fastest static implementation, and with the scheme proposed in [7].

The delay model allows us to estimate about a 25 percent reduction in execution time for both radix 4 and radix 16 for double-precision execution (for radix 16 and single-precision execution using the double-precision datapath, we achieve a 30 percent reduction) with respect to the fastest implementations with which we compare. To verify our results, we have synthesized all the compared implementations for radix 4 using standard cells and a synthesis tool and obtained very similar delay ratios as those resulting from our estimation. With respect to the hardware complexity, the radix-4 proposal has almost the same hardware complexity as the compared implementations, whereas, for radix 16, there is an increase of about 30 percent (with respect to the fastest compared radix-16 design). The area estimates have been obtained from the synthesis.

The paper is organized as follows: In Section 2, we briefly describe the basics of the "standard" digit-by-digit division algorithm. Section 3 introduces a new class of division algorithms for a generic radix. In Sections 4 and 5, we fully develop the design details for a radix-4 algorithm with minimally redundant digit set and a radix-16 algorithm with two overlapped radix-4 stages, respectively. Section 6 presents estimates of the cycle time and area of the proposed and of several previous implementations and reports on the validation by synthesis. Finally, in Section 7, we give our conclusions.

1. Since the delay model is rough, this register position and clock scheduling should be refined in actual implementations with detailed simulation.

2. The granularity at which the modules can be broken to introduce registers.

2 Basics of Digit-by-Digit Division

The "standard" digit-by-digit radix-r division algorithm computes the final result by performing a suitable number of iterations, each one consisting of the following phases (for details, see, for instance, [1]):

1. **Quotient-Digit Selection**: Based on the value of the current residual (denoted by $w[j]$) and of the divisor (denoted by d), one digit of the quotient is produced. If a redundant digit set is used for the quotient, estimates[3] of the shifted residual (denoted by $\widehat{rw}[j]$) and of the divisor ($\widehat{d}$)) can be used, resulting in

$$q_{j+1} = SEL(\widehat{rw}[j], \widehat{d}),$$

   where $q_{j+1} \in \{-a, \dots, 0, \dots, a\}$ with $a > (r-1)/2$, and SEL is the selection function.

   Any function SEL should assure convergence, that is, the next residual should be bounded by $|w[j+1]| \leq \rho d$, with $\rho = a/(r-1)$.
2. **Residual Updating and Quotient Formation**: Using the digit selected at Step 1, the next residual and quotient are produced, as follows:

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

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

2.1 Selection Function

The design of the selection function involves the following steps:

- Function specification: This corresponds to a high-level (arithmetic) description of the particular function chosen to satisfy the bound $|w[j+1]| \leq \rho d$.
- Algorithm for implementation: This is an algorithmic decomposition of the function specification that leads to a suitable implementation.
- Implementation at the logic level of the algorithmic decomposition: This implementation can have several levels, beginning with the interconnection of arithmetic modules (such as adders and comparators) and terminating at the logic gate level.

The following alternatives have been proposed for the **specification** of the function:

1. Use multiples[4] of $\widehat{d}$. That is,

$$q_{j+1} = k \quad \textbf{if} \quad M_k\widehat{d} \leq \widehat{rw}[j] < M_{k+1}\widehat{d}$$

   with $k = -a, \dots, 0, \dots, +a$.
2. Use selection constants. In this case, the range of d is divided into subranges and a set of comparison constants (called selection constants) is determined

3. In this section, estimates are generically designated by including a $\widehat{}$, that is, an estimate of x is designated by $\widehat{x}$. In later sections, we define specific estimates.

4. These multiples can be fractional, such as $(3/2)\widehat{d}$.

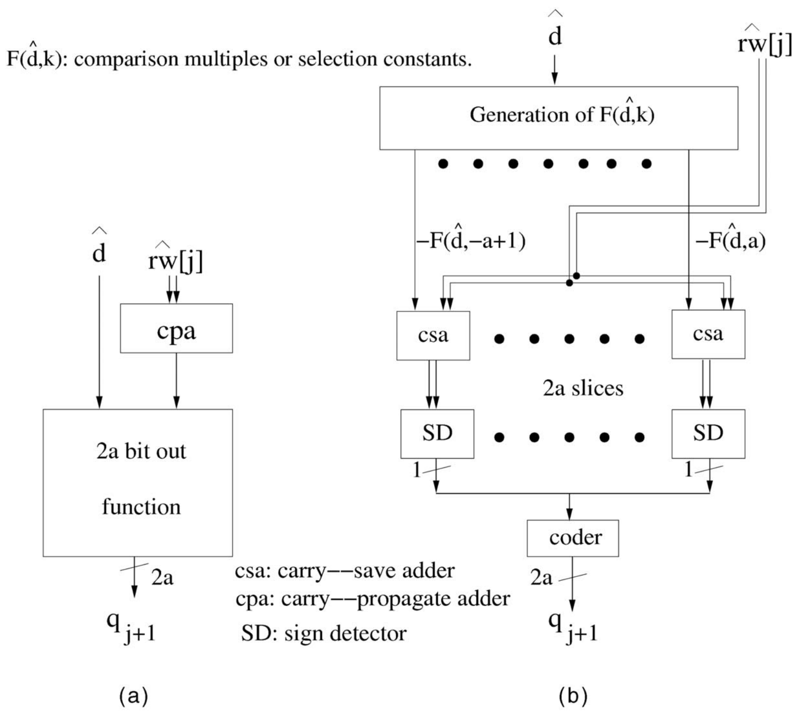


Fig. 1. Algorithms for the selection function. (a) Function-table-based. (b) Comparison-based.

for each subrange. That is, the selection function is specified as

$$q_{j+1} = k \quad \textbf{if}$$
$$\hat{d} \text{ is within subrange } i \textbf{ and } m_{i,k} \leq \widehat{rw}[j] < m_{i,k+1},$$

where $m_{i,k}$ is the selection constant at subrange i, with $k = -a, \ldots, 0, \ldots, +a$.

Alternatives 1 and 2 can be described by the generalized expression

$$q_{j+1} = k \quad \textbf{if} \quad F(\hat{d}, k) \leq \widehat{rw}[j] < F(\hat{d}, k+1),$$

where $F(\hat{d}, k)$ stands for $M_k\hat{d}$ or $m_{i,k}$.

3. Use the function

$$q_{j+1} = round(\widehat{rw}[j]) \times M,$$

where M is an approximation of the reciprocal of d. This function results from

$$q_{j+1} = integer(rw[j] \times (1/d));$$

the use of the estimate and of the approximation are possible because of the redundancy of the digit set for q_{j+1}.

4. Prescaling the divisor (and the dividend) so that the scaled divisor is close to 1 and the selection can be done by rounding, that is,

$$q_{j+1} = round(\widehat{rw}[j]).$$

Approaches 1 and 2 are 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. On the other hand, approach 3 has been used for very-high radices, where the multiplication is justified. Approach 4 has been used for radix 4, in which case, there is a trade-off between the overhead in prescaling and the reduction in the complexity of the selection function and, for very-high radices, where the standard algorithm is not practical. For intermediate radices, such as radix 16, multistage algorithms have been used.

For approaches 1 and 2, two **algorithms for implementation** of the function have been used:

1. Function-table-based. From the specification of the function, a description by a table is obtained which is then implemented using any technique for the realization of switching functions, such as networks of gates, networks of standard cells, PLAs, or ROMs. Although $\widehat{rw}[j]$ can be used directly in redundant form, in most implementations, it is first converted to a conventional two's complement representation (see Fig. 1a).
2. Comparison-based. The comparisons of the specification are performed by comparators and this is followed by a coder to obtain the quotient digit. When $\widehat{rw}[j]$ is in redundant form, the comparison can be implemented by a redundant subtraction followed by a sign detection (SD) and coder, as shown in Fig. 1b. Note that, in this implementation, the comparison constants (selection constants or divisor multiples) are computed outside of the iteration.

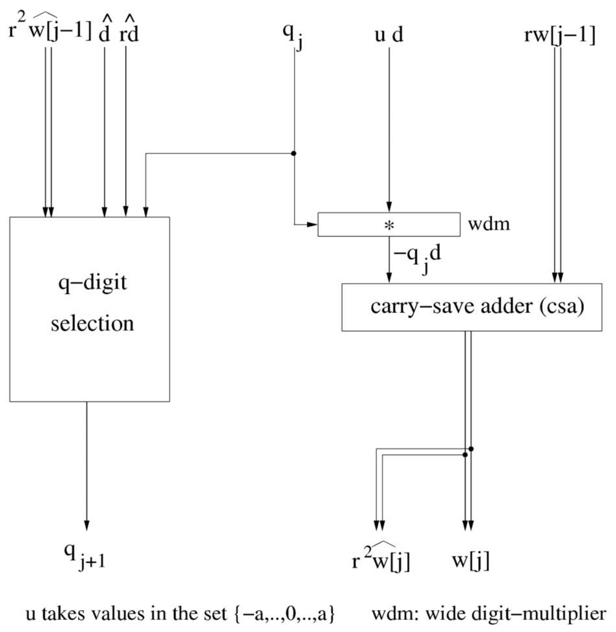

$r^2\widehat{w[j-1]}$ $\hat{d}$ $\widehat{rd}$
q_j
u d
rw[j−1]
q−digit selection
*
wdm
$-q_jd$
carry−save adder (csa)
q_{j+1}
$r^2\widehat{w[j]}$
w[j]
u takes values in the set {−a,..,0,..,a}
wdm: wide digit−multiplier

Fig. 2. An iteration of the proposed class of algorithms.

3 Proposed Algorithm

To reduce the cycle time of the division algorithm, we propose the following:

1. To perform the quotient-digit selection using an estimate of $r^2w[j-1]$ and q_j, that is, to compute the estimate of $rw[j]$ as part of the selection function. Then,

$$q_{j+1} = SEL(\widehat{r^2w}[j-1], q_j, \widehat{rd}, \widehat{d}).$$

   This selection function, although, in principle more complicated than the one discussed in the previous section, provides more flexibility to reduce the cycle time. Fig. 2 shows an iteration of the complete division algorithm, including the proposed selection function.
2. To use a comparison-based algorithm. The corresponding specification is

$$q_{j+1} = k \text{ if } F(\widehat{d}, k) \le \widehat{r^2w}[j-1] - q_j\widehat{rd} < F(\widehat{d}, k+1). \tag{1}$$

   The function $F(\widehat{d}, k)$ includes comparison with divisor multiples as well as with selection constants. This specification allows the partitioning of the selection function as a way of reducing the critical delay.
3. To perform transformations of the resulting algorithm to reduce the critical paths.
4. To obtain a small cycle time by suitable register placement and clock scheduling.

Items 1 to 3 above produce a class of algorithms and our intent is to develop one of these algorithms that produces an implementation with reduced cycle time and an acceptable area. Several of the algorithms are described at a high level in [15]; we have estimated the cycle time and area of these and have selected the one described in detail in the next section.

3.1 Detailed Description of the Selected Algorithm

We now develop the details of the selected algorithm. We use the comparison-based scheme with selection constants.[5] That is, (1) becomes

$$q_{j+1} = k \quad \text{if} \quad m_{i,k} \le \widehat{r^2w}[j-1] - q_j\widehat{rd} < m_{i,k+1}.$$

We use the set of constants of the standard algorithm since our transformations do not lead to a reduction in the number of constants or its number of bits.

As commonly done, the estimate $\widehat{d}$ corresponds to the divisor truncated to δ fractional bits and the estimate of $rw[j]$ results from truncating the carry-save representation to t fractional bits. As a consequence, the selection constants also have t fractional bits. For a specific value of $\widehat{d}$ (that is, a specific subrange), the corresponding set of selection constants is generated. For simplicity of notation, in the sequel, we call these constants m_k.

Specifically, in the comparison-based approach of Fig. 1, the quotient digit q_{j+1} is determined from the sign detection of the carry-save subtraction[6] $\lfloor rw[j] \rfloor_t - m_k$ for each value of k. Since

$$w[j] = rw[j-1] - q_jd,$$

we have

$$\lfloor rw[j] \rfloor_t - m_k = \lfloor r^2w[j-1] - rq_jd \rfloor_t - m_k.$$

When using carry-save representation, the previous computation is performed in two separate truncation steps with one additional fractional bit:

$$\lfloor rw[j] \rfloor_t - m_k = \Big(\big\lfloor \lfloor r^2w[j-1] \rfloor_{t+1} + \lfloor -rq_jd \rfloor_{t+1} \big\rfloor_t \Big) - m_k,$$

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

$$\lfloor rw[j] \rfloor_t - m_k = \Big(\big\lfloor \big[\lfloor r^2w[j-1] \rfloor_{t+1} - m_k \big] + \lfloor -rq_jd \rfloor_{t+1} \big\rfloor_t \Big). \tag{2}$$

In this way, the computation of $\lfloor r^2w[j-1] \rfloor_{t+1} - m_k$ is outside of the q_j to q_{j+1} path. Fig. 3 shows the detailed architecture of the proposed quotient-digit selection.

The number of fractional bits of the various signals is as described by (2). We now consider the number of integer bits, which corresponds to the maximum value of $|\lfloor rw[j] \rfloor_t - m_k|$. Since

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

from (2), these maximum values are

$$\begin{aligned} &\max(|\lfloor rw[j] \rfloor_t - m_k|) = \\ &\max(\lfloor r\rho \rfloor_t + |m_-|, |\lfloor -r\rho - 2^{-t} \rfloor_t - m_+|). \end{aligned} \tag{3}$$

This results in $\log_2(r) + 2$ or $\log_2(r) + 3$ integer bits, depending on the values of r, ρ, and t.

5. The use of selection constants provides more flexibility than the case of multiples of the divisor.
6. $\lfloor x \rfloor_t$ means truncation of x to t fractional bits.

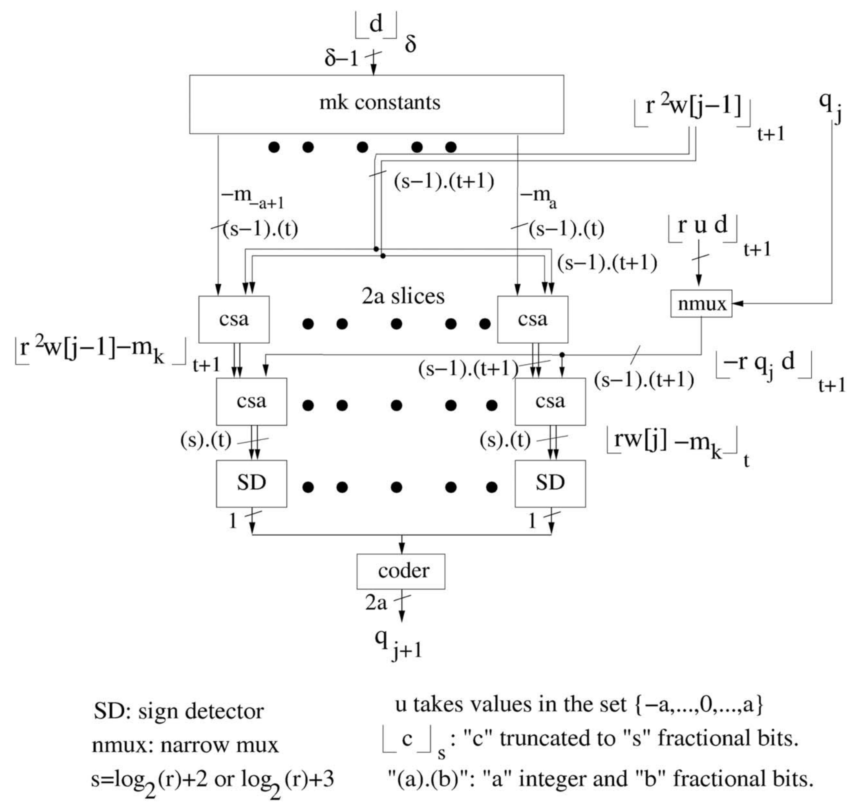


Fig. 3. Architecture of the proposed digit selection.

In the next section, we study the architecture for the radix-4 case with a minimally redundant digit set and, in Section 5, we present the details for the radix-16 architecture using two overlapped radix-4 stages.

4 Architecture for Radix 4 with Minimally Redundant Digit Set

We now consider the case for radix 4 and $a = 2$. This is a widely implemented algorithm since the multiples of the divisor are easily generated. However, the same techniques could be applied to other radix-4 cases (for instance, $a = 3$).

As mentioned in the detailed analysis above, since we obtain the same $\lfloor rw[j] \rfloor_t$ as in the standard algorithm, we can use the selection constants described in [1]. In this case, $\delta = 4$ and $t = 3$. Table 1 shows the ranges for the constants. The detailed view of this algorithm corresponds to Fig. 3 with $r = 4$, $a = 2$, $\delta = 4$, and $t = 3$. However, for radix 4, an optimization in the number of bits is possible, as described now.

4.1 Reduction in the Number of Bits of $\lfloor rw[j] \rfloor_t - m_k$

From (3), for $r = 4$, $\rho = 2/3$, and $t = 4$, we get

$$\begin{aligned} -43/16 - 11/8 = -65/16 \leq \lfloor rw[j] \rfloor_t - m_k \leq 42/16 + 11/8 \\ = 64/16. \end{aligned}$$

Consequently, the number of bits of $\lfloor rw[j] \rfloor_t - m_k$ 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 2, for some ranges of values of $\lfloor rw[j] \rfloor_t$, some of the outputs of the SD modules are not used in the determination of the quotient digit. In the table, we denote by $[c, d]$ the interval of values

TABLE 1
Ranges of m_k (Values Scaled by 16) for Radix 4 ($a = 2$) [1]

i	0	1	2	3	4	5	6	7
m_{-1}	-13	-15	-16	[-18, -17]	[-20, -19]	[-21, -20]	[-23, -21]	[-25, -23]
m_0	[-5, -4]	[-6, -5]	[-6, -5]	[-7, -5]	[-8, -6]	[-8, -6]	[-9, -6]	[-10, -7]
m_1	[3, 4]	[4, 5]	[4, 5]	[4, 6]	[4, 7]	[5, 7]	[5, 8]	[6, 9]
m_2	12	14	15	[16, 17]	[18, 19]	[19, 20]	[20, 22]	[22, 24]

TABLE 2
Reduction of Number of Bits of $\lfloor rw[j]\rfloor_t - m_k$

$\lfloor rw[j]\rfloor_t$	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 $(SD_0, SD_{-1}) = (-,-)$
$\lfloor rw[j]\rfloor_t - m_k$	$[m_1 - m_2, H - m_2]$	$[L - m_1, H - m_1]$	$[L - m_0, H - m_0]$	$[L - m_{-1}, m_0 - m_{-1}]$	
$\max(\lvert\lfloor rw[j]\rfloor_t - m_k\rvert)$	1.25	3.25	3.125	1.312	
Integer bits	2	3	3	2	
Total no. bits	2+4 =6	3+3=6	3+3=6	2+4=6	

of $\lfloor rw[j]\rfloor_t$, by L and H its minimum and maximum values, and, by x, the don't cares. Note that the values of c, d, and m_k depend on $\widehat{d}$.

2. Considering the most positive and most negative values (not don't cares), we determine the maximum magnitudes of $\lfloor rw[j]\rfloor_t - m_k$. The corresponding values are given in the table, together with the required number of integer bits.

With respect to the number of fractional bits, we observe the following:

- For $\lfloor rw[j]\rfloor_t - m_2$, four bits are required because m_2 includes the value 15/16 (see Table 1).
- For $\lfloor rw[j]\rfloor_t - m_1$, it might be possible to use three bits because all m_1 can be selected so that they are multiples of 1/8. However, using three bits requires that $t = 3$, so it is necessary to determine whether constants that are multiples of 1/8 are still possible for $t = 3$. We verified that this is not possible only for $\widehat{d} = 8/16$ $(i = 0)$; 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 $\lfloor rw[j]\rfloor_t - m_0$, also, three fractional bits can be used since all m_0 can be selected so that they are multiples of 1/8. Again, this requires the addition of 1/16 to the constant m_0 for $\widehat{d} = 8/16$.
- For $\lfloor rw[j]\rfloor_t - m_{-1}$, four fractional bits are needed because m_{-1} can take the value -15/16.

In summary, the last row of Table 2 gives the required number of integer and fractional bits of $\lfloor rw[j]\rfloor_t - m_k$. The corresponding quotient-digit selection implementation is shown in Fig. 4.

5 Architecture for Radix 16 with Two Overlapped Radix-4 Stages

The architecture design technique described in Section 4 can be implemented directly for radices higher than 4, say 8 or 16. However, these implementations have the following drawbacks:

1. The number of replicated slices in the selection function and the number of constants to be "preloaded" increases. For instance, for radix 8, this number would be between eight and 14, depending on the quotient-digit set, and, for radix 16, between 16 and 30. This significantly increases the area of the selection function.
2. The increase in the number of replicated slices increases the load of $\widehat{q_j d}$, increasing the delay of the path q_j to q_{j+1}.
3. To avoid the need for precomputation of multiples of the divisor, the quotient digit is decomposed into (two) components, each of which is restricted to the range -2 to 2 (this results in $a \leq 10$ for radix-16). This increases the delay of the addition in the recurrence.

We have done some designs and evaluations for the radix-8 case and have concluded that the increase in area is too high for the speedup obtained. Consequently, we have explored the radix 16 with two overlapped radix-4 stages [1], [2]. The corresponding radix-16 iteration is of the form:

$$w[j+2] = 4(4w[j] - q_{j+1}d) - q_{j+2}d,$$

where q_{j+1} and q_{j+2} are two radix-4 digits with $a = 2$.

For the radix-4 selection function, we use the algorithm proposed in Section 4. However, in this case, we have to obtain two quotient digits. Specifically, the selection functions for each digit are of the following type:

$$q_{j+1} = SEL(\widehat{4^2 w}[j-1], q_j, \widehat{4d}, \widehat{d})$$

and

$$q_{j+2} = SEL(\widehat{4^3 w}[j-1], q_j, q_{j+1}, \widehat{4^2 d}, \widehat{4d}, \widehat{d}).$$

For the computation of q_{j+1}, we proceed as in the radix-4 algorithm by computing the sign of the carry-save representation of

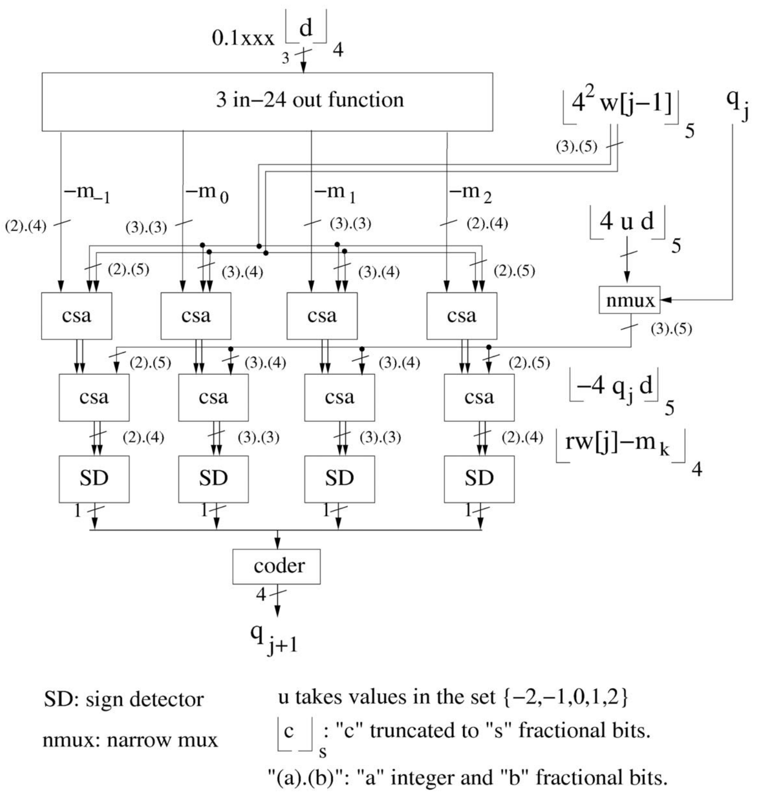


Fig. 4. Proposed radix-4 quotient-digit selection implementation.

$$(\lfloor 4w[j]\rfloor_t - m_k) = \left\lfloor \left(\left\lfloor 4^2 w[j-1] - m_k \right\rfloor_{t+1} - \left\lfloor 4q_j d \right\rfloor_{t+1} \right) \right\rfloor_t .$$

To determine q_{j+2}, we compute the sign of the carry-save representation of $(\lfloor 4w[j+1]\rfloor_t - m_k)$. A direct implementation to obtain the carry-save representation is:

$$(\lfloor 4w[j+1]\rfloor_t - m_k) = \left\lfloor \left(\left\lfloor 4^3 w[j-1] - m_k \right\rfloor_{t+2} - \left\lfloor 4^2 q_j d \right\rfloor_{t+2} \right) - \left\lfloor 4q_{j+1} d \right\rfloor_{t+1} \right\rfloor_t .$$

To have overlapping with the selection of q_{j+1}, the above computation (as well as the sign detections) is performed speculatively for all values of q_{j+1}. Finally, the correct value is selected using a multiplexer. That is,

$$(\lfloor 4w[j+1]\rfloor_t - m_k)_u = \left\lfloor \left(\left\lfloor 4^3 w[j-1] - m_k \right\rfloor_{t+2} - \left\lfloor 4^2 q_j d \right\rfloor_{t+2} \right) - \lfloor 4ud \rfloor_{t+1} \right\rfloor_t ,$$

for $u = \{-2, -1, 0, 1, 2\}$ and $k = \{-1, 0, 1, 2\}$.

Fig. 5 shows the architecture. An analysis of the components in the different paths shows that this implementation would result in a large cycle time, not producing the desired speed-up with respect to the radix-4 case. We now explore ways of reducing this cycle time.

5.1 Architecture for Reduced Cycle Time

To reduce and balance the critical paths, we perform the following transformations:

1. For the path going from $\lfloor 4^2 w[j-1]\rfloor$ to $\lfloor 4^2 w[j+1]\rfloor$ and $\lfloor 4^3 w[j+1]\rfloor$ (see Fig. 6):
   a. Instead of using the wide multiplexer to produce the most significant part of $-q_{j+1}d$, we introduce an additional narrow multiplexer for $\lfloor 4^3 q_{j+1} d\rfloor$. This reduces the load on q_{j+1} in this critical path.
   b. We eliminate the half-adder delay to produce $\lfloor 4^2 w[j+1]\rfloor$ and $\lfloor 4^3 w[j+1]\rfloor$ by introducing a narrow carry-propagate adder (cpa) to assimilate the carry-save representation of $\lfloor 4^4 w[j]\rfloor$.
   c. We eliminate the carry-save adder to produce $\lfloor 4^2 w[j-1]\rfloor - m_k$ by generating $\lfloor 4ud - m_k\rfloor$ (instead of $-m_k$), with $u = \{-2, -1, 0, 1, 2\}$, and selecting the correct value with q_j. This increases the complexity of the generation of the constants, but this is outside of the critical path.
2. For the path going from $\lfloor 4^3 w[j-1]\rfloor$ to q_{j+2}, we eliminate the first carry-save adder by generating $\lfloor 4^2 ud - m_k\rfloor$, with $u = \{-2, -1, 0, 1, 2\}$ and selecting the correct value with q_j (see Fig. 7). Note that, because of the term with 4^2, these constants are different than those of item 1.c.
3. For the path from q_j to q_{j+2}, the addition of $\lfloor u(4d)\rfloor$, with $u = \{-2, -1, 0, 1, 2\}$ is advanced and performed directly with $\lfloor 4^3 w[j-1]\rfloor$ (see Fig. 7).

The complete resulting radix-16 architecture is shown in Fig. 8. As seen in the figure, these transformations increase the area required for the q-selection implementation. We also include in the figure the critical path and the positioning of registers; these aspects are discussed in Section 6.

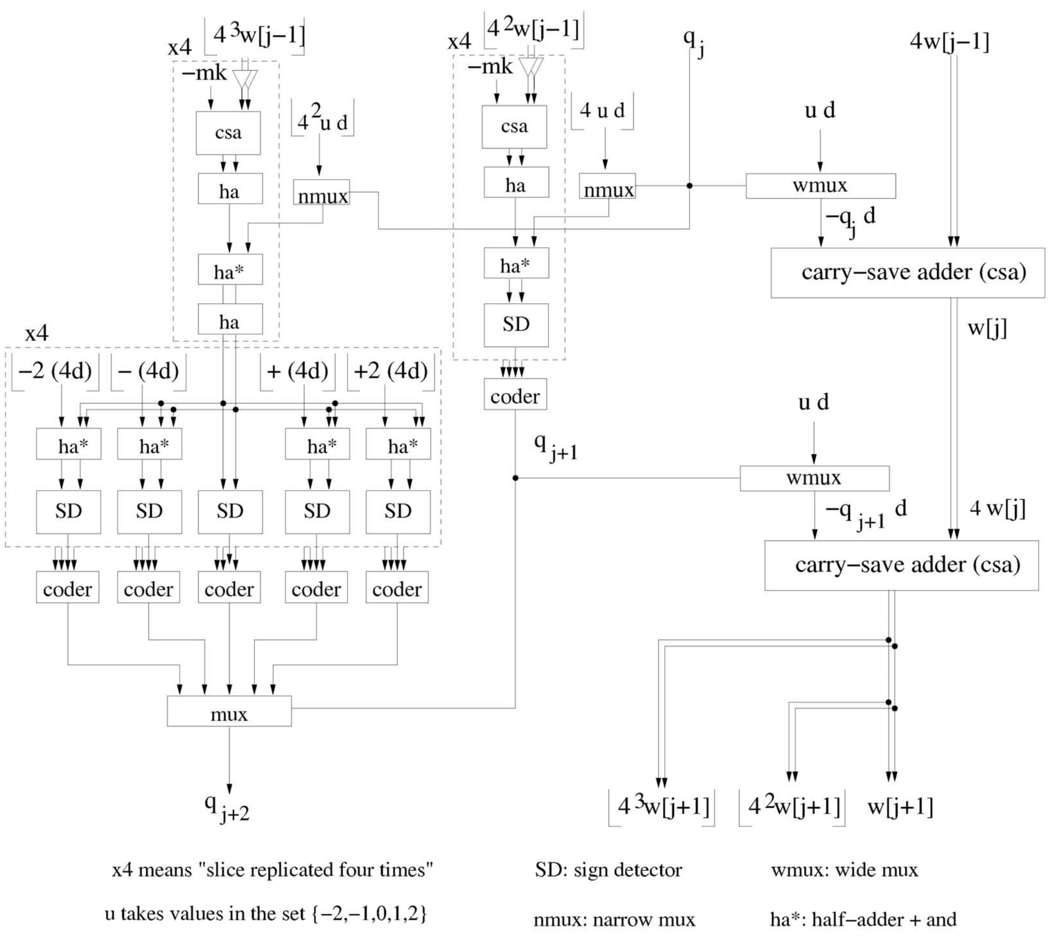


Fig. 5. Radix-16 iteration implementation.

5.2 Implementation of the Initial Constants

As is shown in Figs. 6 and 7, the algorithm relies on the computation of the constants $\lfloor 4ud - m_k \rfloor_5$ and $\lfloor 4^2ud - m_k \rfloor_5$ in the initialization cycle. We consider now alternative implementations. The most direct computation method seems to be the implementation of a combinational network (look-up table) having, as input, the required bits of d and, as output, the constants. The area of this solution might be quite large; based on our estimations, using a synthesis tool it would correspond to more than 50 percent of the total area.

An alternative is to compute the subtraction using carry-propagate adders. That is, a small network determines the m_k and then 32 adders are used to produce the output (for $u = 0$, no adders are needed). In our synthesis with standard cells, the area contribution is now about 25 percent. To reduce the area further, it is possible to use the symmetry among pairs of m_k, in particular, the fact that $m_2 \approx -m_{-1}$ and $m_1 \approx -m_0$. In general, consider the computation of $G = -4ld - m_g$ and $H = -4(-l)d - m_h$ for $l > 0$ and $m_g = -m_h$. Then,

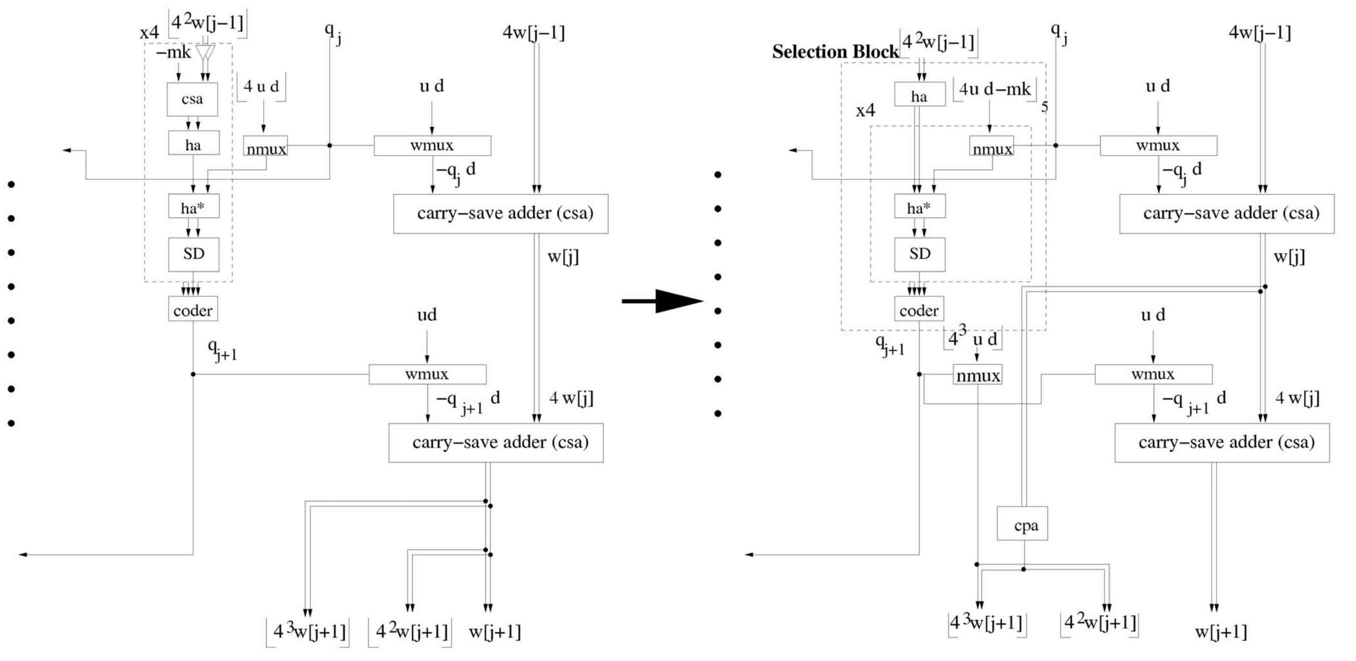
$$H = -4(-l)d - m_h = (4ld - m_h) = 4ld + m_g = -G \text{ for } l > 0.$$

Consequently, it is possible to compute either H or G and obtain the other by a change of sign operation. This change of sign is performed by a bit-invert operation followed by the addition of one unit in the last position (ulp). However, since the value is truncated, the ulp is neglected without additional error and the change of sign is done by a bit-invert operation. This can be done for both sets of constants and replaces half of the adders by bit-invert operations.

Unfortunately, the corresponding m_k pairs are not exactly symmetrical. For instance, for m_2 and m_{-1}, we see from the allowed intervals that it is possible to make $m_{-1} = -(m_2 + 1/16)$ (see Table 1). Consequently, we get

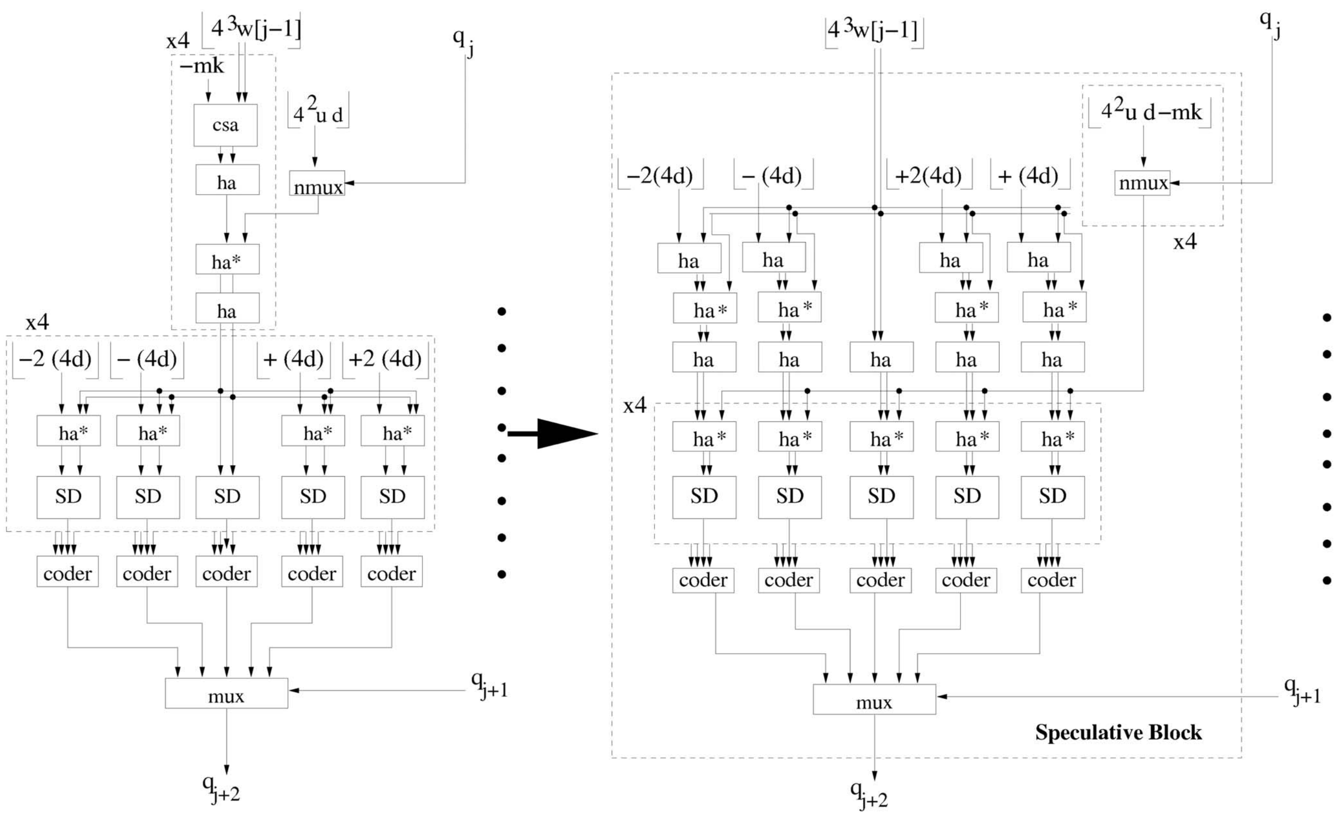
$$-4ld - m_{-1} = -(4ld - (m_2 + 1/16)) = -(4ld - m_2) + 1/16.$$

That is, bit invert and add 1/16. Since the sign-detector (SD) modules corresponding to the m_{-1} slice have four fractional bits (see Section 4.1), the 1/16 can be added as a carry-in to the sign-detector (SD) modules corresponding to m_{-1}. For the pair m_1 and m_0, it is possible to obtain symmetrical values (see Table 1). However, this would require that these constants have four fractional bits (see Table 2), instead of the minimum three bits. Since these constants with three fractional bits were used to achieve the width of six bits for the SD modules, it might be preferable to use the same approach as for the (m_2, m_{-1}) pair, that is, add a carry-in to the SD modules corresponding to m_0 (this results in adding 1/8 to m_1 to produce $= -m_0$). Our synthesis results with


x4
$\lfloor 4^2 w[j-1] \rfloor$
$-mk$
csa
ha
$\lfloor 4\,u\,d \rfloor$
nmux
q_j
$4w[j-1]$
u d
wmux
$-q_j\,d$
carry−save adder (csa)
ha*
SD
coder
$w[j]$
q_{j+1}
ud
wmux
$-q_{j+1}\,d$
$4\,w[j]$
carry−save adder (csa)
$\lfloor 4^3 w[j+1] \rfloor$
$\lfloor 4^2 w[j+1] \rfloor$
$w[j+1]$
Selection Block
$\lfloor 4u\,d-mk \rfloor_5$
$\lfloor 4^3\,u\,d \rfloor$
nmux
cpa

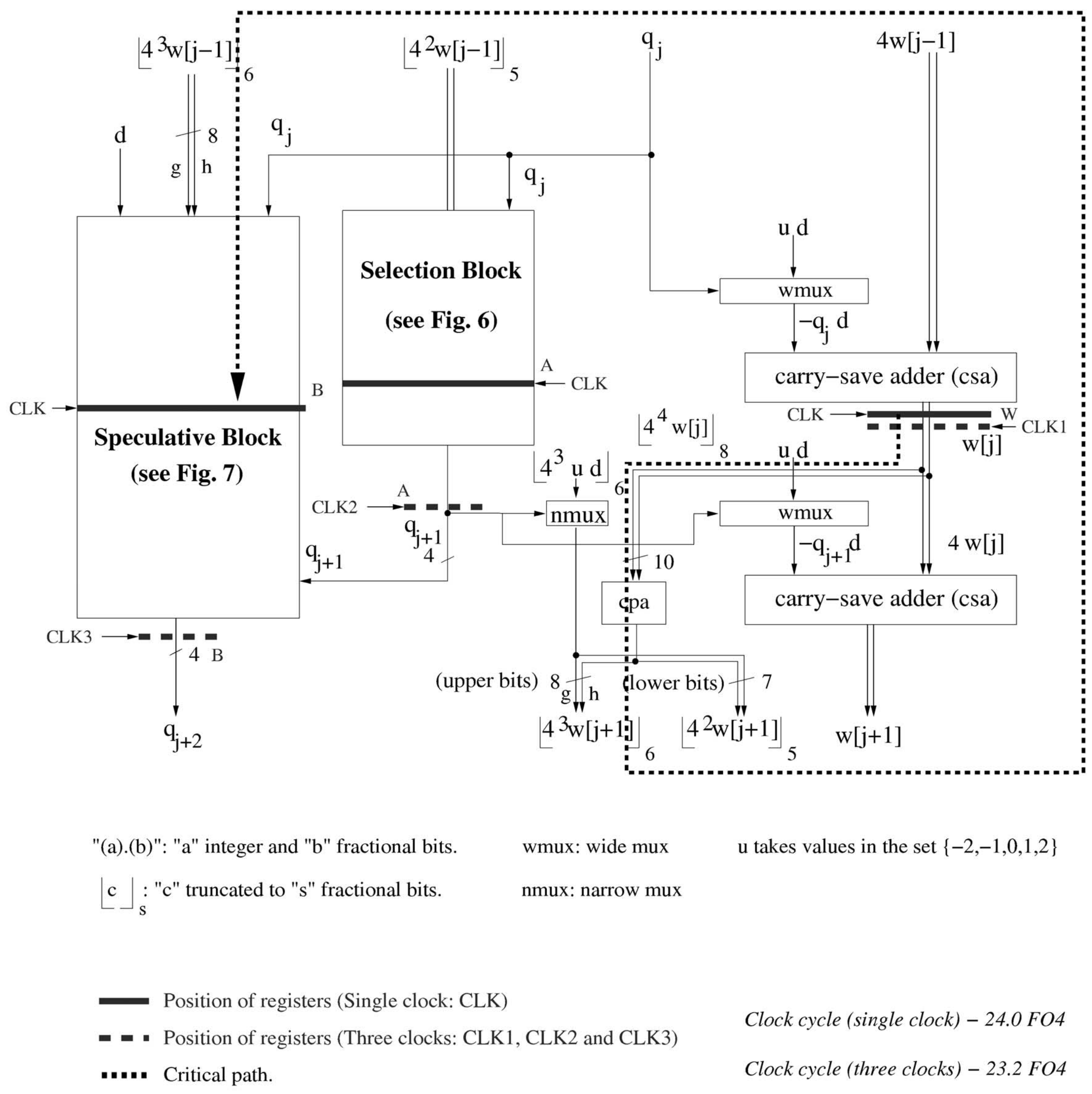
u takes values in the set {−2,−1,0,1,2}
x4 means "slice replicated four times"
SD: sign detector
nmux: narrow mux
wmux: wide mux
ha*: half−adder + and

Fig. 6. Detail of the radix-16 iteration after improvement 1.


x4
$\lfloor 4^3 w[j-1] \rfloor$
$-mk$
csa
ha
$\lfloor 4^2 u\,d \rfloor$
nmux
q_j
ha*
ha
$\lfloor -2\,(4d) \rfloor$
$\lfloor -\,(4d) \rfloor$
$\lfloor +\,(4d) \rfloor$
$\lfloor +2\,(4d) \rfloor$
SD
coder
mux
q_{j+1}
q_{j+2}
$\lfloor 4^2 u\,d-mk \rfloor$
Speculative Block

x4 means "slice replicated four times"
u takes values in the set {−2,−1,0,1,2}
SD: sign detector
nmux: narrow mux
wmux: wide mux
ha*: half−adder + and

Fig. 7. Detail of the radix-16 iteration after the improvements 2 and 3.



$\lfloor 4^3 w[j-1] \rfloor_6$
$\lfloor 4^2 w[j-1] \rfloor_5$
q_j
$4w[j-1]$
d
8
g
h
q_j
q_j
Selection Block
(see Fig. 6)
u d
wmux
$-q_j\, d$
carry−save adder (csa)
A
CLK
B
CLK
Speculative Block
(see Fig. 7)
CLK
W
CLK1
$w[j]$
$\lfloor 4^4 w[j] \rfloor_8$
$\lfloor 4^3\, u\, d \rfloor_6$
u d
CLK2
A
nmux
wmux
q_{j+1}
4
q_{j+1}
$-q_{j+1} d$
4 w[j]
10
cpa
carry−save adder (csa)
CLK3
4
B
(upper bits)
8
g
h
(lower bits)
7
q_{j+2}
$\lfloor 4^3 w[j+1] \rfloor_6$
$\lfloor 4^2 w[j+1] \rfloor_5$
w[j+1]
"(a).(b)": "a" integer and "b" fractional bits.
wmux: wide mux
u takes values in the set {−2,−1,0,1,2}
$\lfloor c \rfloor_s$: "c" truncated to "s" fractional bits.
nmux: narrow mux
Position of registers (Single clock: CLK)
Position of registers (Three clocks: CLK1, CLK2 and CLK3)
Critical path.
Clock cycle (single clock) – 24.0 FO4
Clock cycle (three clocks) – 23.2 FO4

Fig. 8. Proposed radix-16 architecture.

standard cells show that the area contribution is now about 15 precent.

6 Estimation of Cycle Time and Comparisons

This section presents the cycle time estimation of the proposed architectures and compares them with previous implementations. As mentioned in the introduction to make the delay estimations, we used a timing model based on logical effort [9]. We provide the delays[7] relative to the delay of a minimum-sized inverter with a fanout of four (FO4 delay). This is a common measure of delay that remains almost constant over a wide variety of process technologies, temperatures, and voltages. This delay model guided us in the algorithmic design of the proposed schemes. Moreover, it provides more insight at the architecture level than the results obtained from a synthesis tool. Later in this paper, we validate our evaluations, based on this rough model, by performing an actual synthesis.

To estimate the cycle time, it is necessary to place the registers and define the clock scheduling to achieve the minimum critical path. For delay calculations, we analyzed combinational loops; for cycle time estimation, we added a constant register delay of 4.0 FO4.

We proceeded as follows: 1) We determined the placement of registers to minimize cycle time with a single clock. This solution might not be convenient in terms of the partitioning of blocks and in terms of the number of flip-flops. 2) We proposed a clock scheduling with two or three clocks to simultaneously achieve the minimum cycle time and number of flip-flops. Here, we summarize the main results obtained. A very detailed analysis is reported in [15].

7. The delay equations of the modules used can be found in [15].

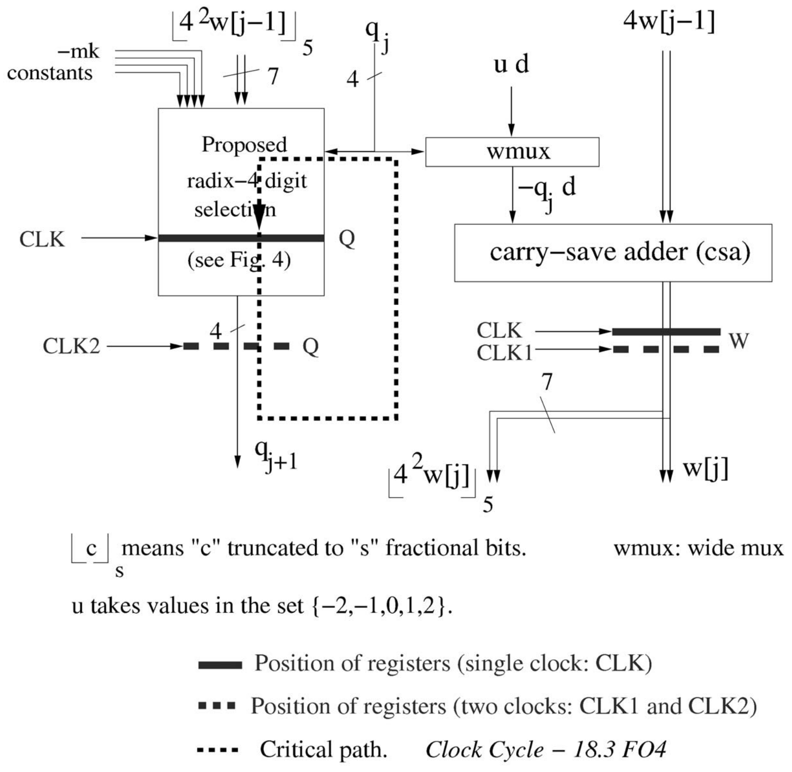

$\lfloor 4^2 w[j-1] \rfloor_5$
q_j
4w[j−1]
−mk constants
7
4
u d
Proposed radix-4 digit selection (see Fig. 4)
wmux
$-q_j\,d$
CLK
Q
carry−save adder (csa)
CLK2
4
Q
CLK
CLK1
W
7
q_{j+1}
$\lfloor 4^2 w[j] \rfloor_5$
w[j]
$\lfloor c \rfloor_s$ means "c" truncated to "s" fractional bits.
wmux: wide mux
u takes values in the set {−2,−1,0,1,2}.
Position of registers (single clock: CLK)
Position of registers (two clocks: CLK1 and CLK2)
Critical path. Clock Cycle − 18.3 FO4

Fig. 9. Proposed radix-4 architecture and timing.

6.1 Cycle Time Estimation of Proposed Architectures

Fig. 9 shows the proposed radix-4 implementation indicating the register placement and the critical path. From the details given in [15], the estimated cycle time is

$$t_{prop-4} = t_{sd}(7.8) + t_{buf}(0.9) + t_{mux4}(2.4) + t_{buf}(1.1) + t_{ha}(2.1) + t_{reg}(4) = 18.3.$$

This register placement partitions the sign detector (see Fig. 4). The figure also shows the register position that minimizes both cycle time and number of flip-flops, using two clocks (CLK1 and CLK2). The corresponding clock scheduling is described in [15].

Fig. 8 shows the proposed radix-16 unit, including the critical path and the position of registers (single clock and three clocks). As described in [15], the cycle time for single clock is

$$t_{prop-16} = t_{buf}(0.7) + t_{cpa}(7.3) + t_{buf}(1.7) + t_{fa}(4.8) + t_{ha}(2.3) + t_{buf}(1.2) + t_{ha}(2.1) + t_{reg}(4.0) = 24.0.$$

In [15], we also show that, using three clocks, the cycle time is reduced to 23.2 FO4.

6.2 Cycle Time Comparison with Previous Proposals

We now compare with previous proposals.

For radix 4, we chose to compare with a standard implementation with digit set -2 to +2, as described, for instance, in [2] and [4], and with a recent implementation with digit set -3 to +3 [8]. For radix 16, we compare with the standard implementation using two overlapped radix-4 stages [2], [4], with the hybrid overlapped scheme, which was reported in [4] to be their fastest static implementation, and with the scheme proposed in [7].

Fig. 10 shows two implementations (function-table-based and comparison-based) of the standard radix-4 architecture with $a = 2$. It indicates the register placement and the critical path. This register placement achieves both the minimum cycle time and number of flip-flops with a single clock. Following [4], [14], we used a narrow path with lower buffering for the part of the residual that inputs the selection module. The delay of the corresponding selection functions (t_{sel}) is 14.4 and 13.6, respectively, and the cycle time is

$$t_{r4a2} = t_{sel} + t_{buf}(0.9) + t_{mux4}(2.5) + t_{ha}(2.3) + t_{reg}(4.0),$$

resulting in cycle times of 24.1 FO4 (function-table-based) and 23.3 FO4 (comparison- based).

A recent radix-4 implementation uses the rarely utilized quotient-digit set from -3 to +3 [8]. This reduces the delay of the quotient-digit selection, but increases the delay of the addition since it implements the multiples $\pm 3d$ as $\pm 2d \pm d$. As shown in [15], the cycle time is

$$t_{r4a3} = t_{sel}(11.3) + t_{buf}(0.9) + t_{mux2}(2.0) + t_{fa}(4.8) + t_{reg}(4.0) = 22.7.$$

The register placement achieves both the minimum cycle time and number of flip-flops with a single clock.

Fig. 11 shows the standard radix-16 architecture with critical path and register placement. The register position achieves the minimum cycle time as well as the minimum number of flip-flops with a single clock. As described in [15], the cycle time is

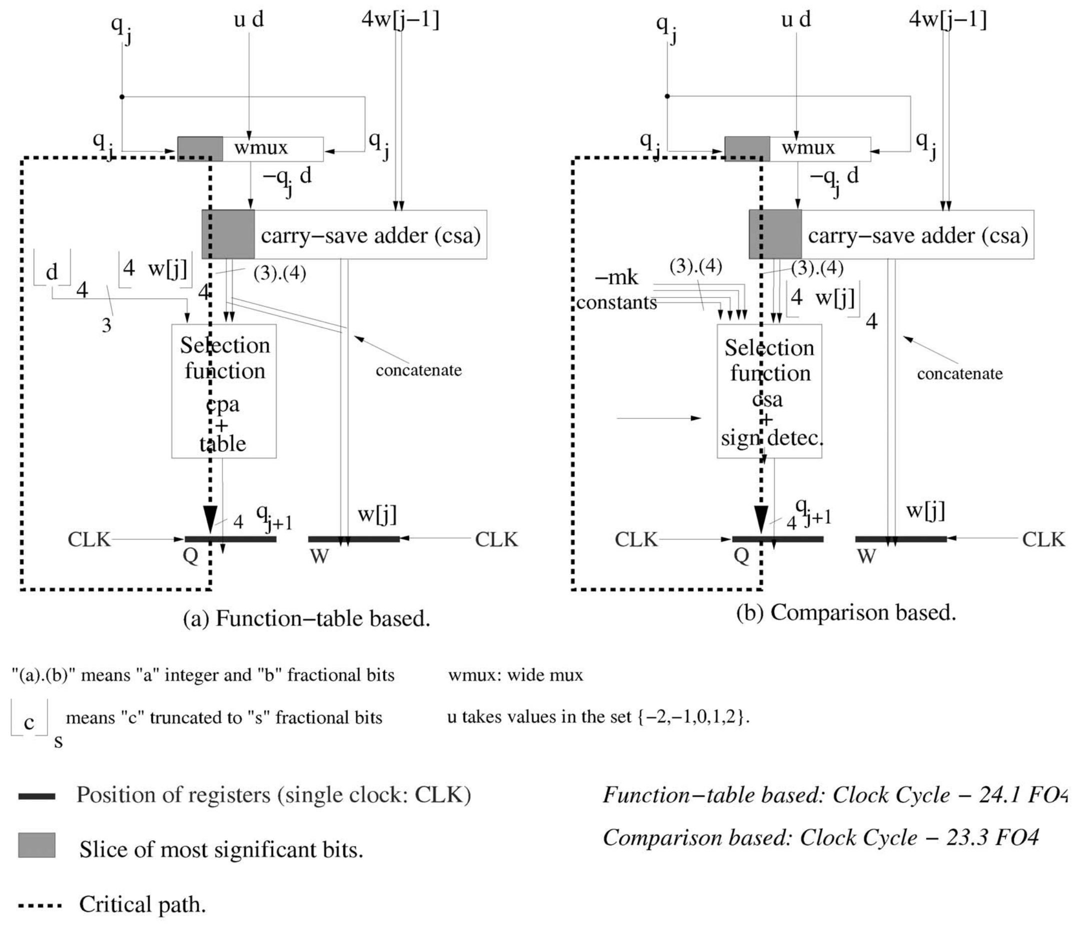

q_j
u d
4w[j−1]
wmux
−q_j d
carry−save adder (csa)
(3).(4)
d 4 3
4 w[j] 4
Selection function cpa + table
concatenate
q_j+1
w[j]
CLK
Q
W
(a) Function−table based.
−mk constants
Selection function csa + sign detec.
(b) Comparison based.
"(a).(b)" means "a" integer and "b" fractional bits
wmux: wide mux
c s means "c" truncated to "s" fractional bits
u takes values in the set {−2,−1,0,1,2}.
Position of registers (single clock: CLK)
Slice of most significant bits.
Critical path.
Function−table based: Clock Cycle − 24.1 FO4
Comparison based: Clock Cycle − 23.3 FO4

Fig. 10. Standard radix-4 architecture and timing.

$$\begin{aligned} t_{r16s} &= t_{buf}(1.1) + t_{mux4}(2.5) + t_{ha}(2.3) + t_{buf}(1.5) + t_{fa}(5.1) \\ &\quad + t_{sel-4}(13.3) + t_{mux5}(2.8) + t_{reg}(4.0) = 32.6. \end{aligned}$$

The radix 16 with hybrid overlapped stages performs a speculative calculation of both the next digit and the most significant bits of the next residual [4]. As is shown in [15], this implementation achieves the following cycle time:

$$\begin{aligned} t_{r16hyb} &= t_{buf}(1.4) + t_{mux5}(2.8) + t_{buf}(1.4) + t_{fa}(5.1) \\ &\quad + t_{sel-4}(13.3) + t_{mux5}(2.8) + t_{reg}(4.0) = 30.8. \end{aligned}$$

In this case, two clocks are necessary to minimize the number of flip-flops.

Finally, the implementation proposed in [7] corresponds to two radix-4 stages (without overlap) in which prescaling is performed so that the quotient-digit selection can be done by rounding the estimate of the residual [16]. The degree of prescaling is more than required for the selection by rounding, so that it is possible to delay the computation of part of the recurrence to the next cycle. As described in [15], the cycle time is

$$t_{r16sca} = t_{fa}(4.9) + t_{fa}(4.8) + t_{mux4}(2.7) + t_{reg}(4.0) = 17.0.$$

Note that, in this scheme, there is an overhead for the prescaling (of 11 cycles).

Table 3 and Table 4 give a summary of the cycle times. These tables also provide the total latency for double-precision execution and the speedup with respect to the corresponding standard implementation.

For single-precision execution using the same double-precision datapath, the speedups for radix 4 are the same as in the double-precision execution. For the radix-16 designs, the only variation corresponds to the design reported in [7], which achieves a speedup of about 0.8 (slower than the reference design) due the large overhead of the prescaling.

6.3 Synthesis Validation and Delay-Area Graph

To validate the model used for the estimations, we report in this section on a synthesis of the radix-4 division units described in the previous sections. From this synthesis, we also obtain an estimate of the area of the blocks to provide a delay-area graph. For this synthesis, we have used the $0.35\ \mu m$ STM standard-cells library with power supply of $3.3\ V$ and Synopsys analysis and synthesis tools (Design Analyzer). The prelayout estimation of delay and area was carried out on the synthesized circuit (or gate-level netlist), including an estimate of the interconnect capacitance. To have results as independent as possible from the technology, we report the delay using as unit the FO4 delay, which is $0.15\ ns$ at power supply of $3.3\ V$ for the library used. Table 5 shows the results of the synthesis.

We conclude that our delay estimations conform well with the results of the synthesis, especially when considering the speedup. The difference in the absolute values in FO4 units results from optimizations performed by the synthesis tool, especially when combining several blocks.

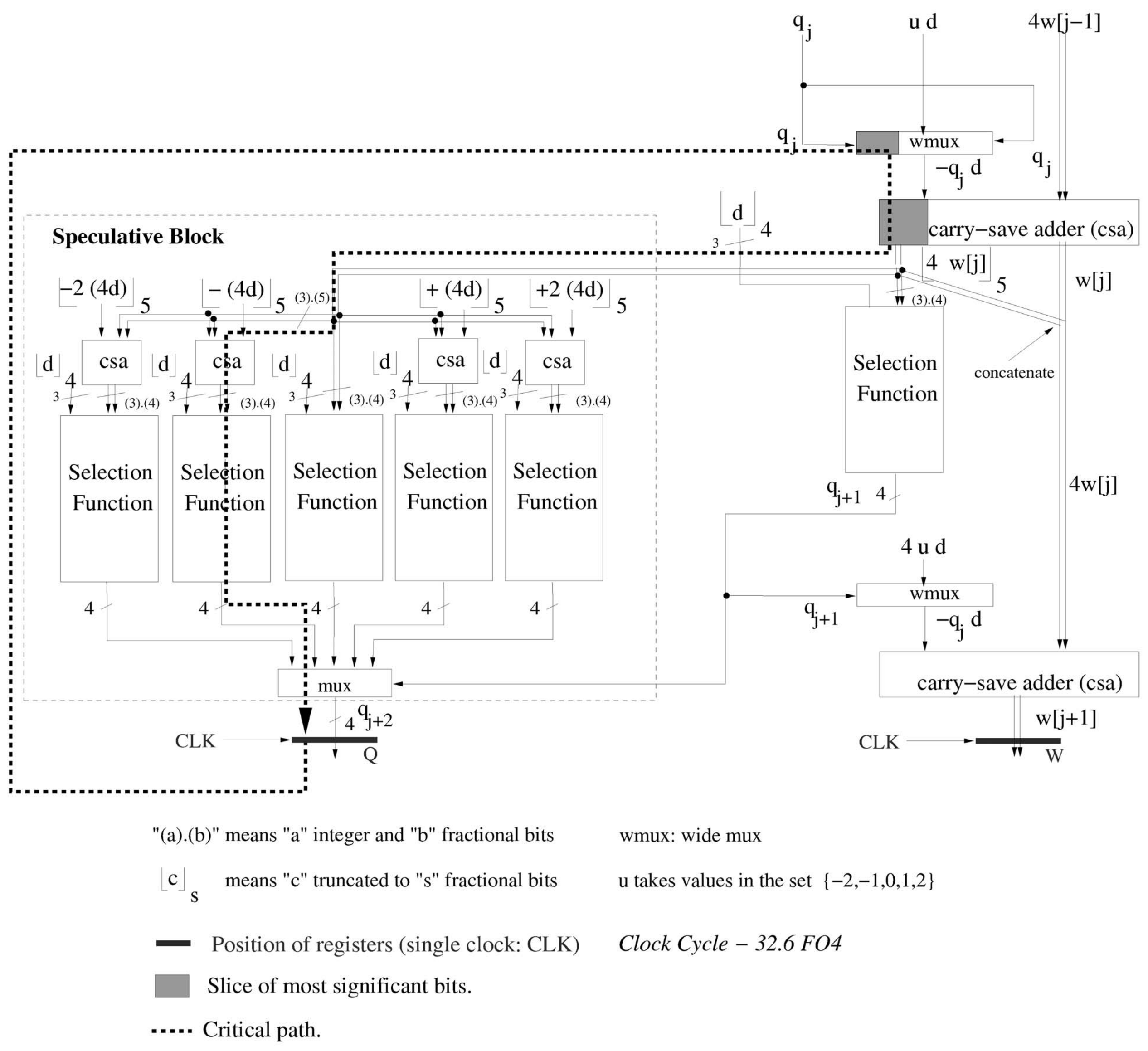


Fig. 11. Standard radix-16 architecture and timing.

TABLE 3
Estimated Speedup (Double Precision) for Radix-4 Implementations

Design	Standard (function-table based)	Standard (comparison based)	$a = 3$	**This work**
cycle time (#FO4)	24.1	23.3	23.0	18.3
number of cycles	30	30	30	30
total delay (#FO4)	723	700	690	550
speedup	1.0	1.03	1.05	1.30

Table 6 gives the area and execution time values for the units compared. We show all cases presented in the previous section, using, for delay, the values reported in Tables 3 and 4 and, for area, values computed from the synthesized blocks. The execution time and hardware complexity of the function-table based radix-4 implementation is taken as the reference. Fig. 12 presents the execution time-area graph, further illustrating the values provided in the table.

7 Conclusions

As we have illustrated in previous sections, the cycle time of existing low-radix digit-recurrence division, such as radix 4,

TABLE 4
Estimated Speedup (Double Precision) for Radix-16 Implementations

Design	Standard	Hybrid	reported in [7]	**This work**
cycle time (#FO4)	32.6	30.8	17.0	24.0 (23.2)*
number of cycles	16	16	27	16
total delay	522	493	459	384 (371)*
speedup	1.0	1.06	1.14#	1.35 (1.40)*

* with three clocks
# Because of the large overhead in this implementation, the speedup is smaller for smaller precision. For instance, for single precision it is (32.6x8)/(17.0x19)=0.81.

TABLE 5
Summary of Results for the Synthesized Radix-4 Units

scheme	T_c [ns] (#FO4)	speedup	Area*	Area ratio
Standard function table–based (Fig. 10(a))	3.25 (22)	1.00	4700	1.00
Standard comparison based (Fig. 10(b))	3.10 (21)	1.05	4800	1.02
Digit set -3 to 3 ([8])	3.12 (21)	1.04	5700	1.20
This work (Fig. 9)	2.42 (16)	1.34	4800	1.02

is determined by the path q_j to q_{j+1}, which includes the digit-selection function. As a way of reducing this cycle time, we propose a class of algorithms for the selection function that allows transformations to take out parts of this function from the q_j to q_{j+1} path. Once this path is not critical, it is possible to balance the paths by the placement of the registers and/or by scheduling the clocks to the various registers.

After describing the above-mentioned class of algorithms, we have examined specific instances and selected one that, for the radix-4 case, produces about a 25 percent reduction in the cycle time without increasing the area. The delay estimates have been done using a model based on the logical-effort technique. To validate the estimate, we have done a synthesis using CAD tools for standard cell libraries. We have extended the use of the proposed class of algorithms to the radix-16 case, where the selection of the radix-16 digit is performed by the overlapping of two radix-4 digits. In this case, to achieve the desired cycle time reduction, additional transformations were required, including extensive speculation. This results in a reduction of about 25 percent in the execution time (for double-precision division) with respect to the fastest of the compared designs. For double precision, the diagram in Fig. 12 shows that the proposed implementations are placed in attractive

TABLE 6
Speedup and Area Ratio for the Units Compared

Scheme	Area ratio	Speedup (DP–SP*)
r=4 (function–table based)	1.0	1.0
r=4 (comparison based)	1.01	1.03
r=4 (a=3)	1.07	1.05
r=4 (**this work**)	1.05	1.32
r=16 (standard)	1.48	1.39
r=16 (hybrid, one clock)	1.85	1.47
r=16 (hybrid, two clocks)	1.73	1.47
r=16 (ref. [7])	1.80	1.58-0.81
r=16 (**this work**, one clock)	2.64	1.88
r=16 (**this work**, three clocks)	2.34	1.95

*DP: double-precision execution; SP: single-precision execution.

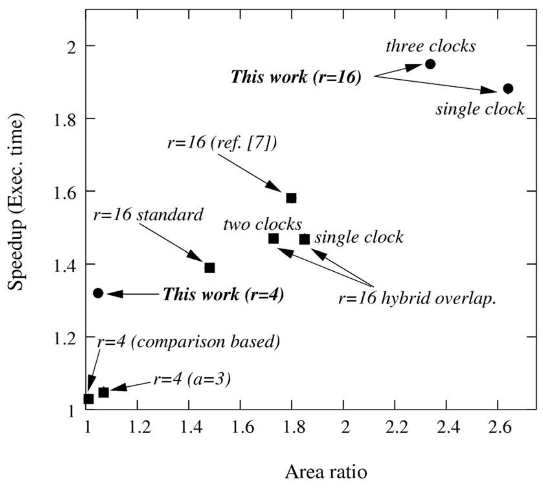


Fig. 12. Diagram of speedup (execution time) and hardware complexity for double precision.

positions in the speedup/area-ratio space. Moreover, with the proposed implementations, it would be easy to also incorporate square root, which is difficult for algorithms based on prescaling.

Acknowledgments

Elisardo Antelo was partially supported by Xunta de Galicia under projects PGIDT-99XI20601A and 03TIC10502PR. Tomás Lang was partially supported by UC MICRO Grant 01-047.

References

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

[2] M. Ercegovac and T. Lang, *Digital Arithmetic.* Morgan Kaufmann, 2003.

[3] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers,* pp. 1646-1650, Nov. 2001.

[4] D. Harris, S. Oberman, and M. Horowitz, "SRT Division Architectures and Implementations," *Proc. 13th IEEE Symp. Computer Arithmetic,* pp. 18-25, 1997.

[5] B. Parhami, "Tight Upper Bounds on the Minimum Precision Required of the Divisor and the Partial Remainder in High-Radix Division," *IEEE Trans. Computers,* vol. 52, no. 11, pp. 1509-1514, Nov. 2003.

[6] P. Kornerup, "Revisiting SRT Quotient Digit Selector," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 38-45, 2003.

[7] E. Rice and R. Hughey, "A New Iterative Structure for Hardware Division: The Parallel Paths Algorithm," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 54-62, 2003.

[8] G. Gerwig, H. Wetter, E.M. Schwarz, and J. Haess, "High Performance Floating-Point Unit with 116 Bit Wide Divider," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 87-94, 2003.

[9] I.E. Sutherland, R.F. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits.* Morgan Kaufmann, 1999.

[10] V.G. Oklobdzija et al., "Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders," *Proc. 16th IEEE Symp. Computer Arithmetic,* pp. 272-279, 2003.

[11] J.P. Fishburn, "Clock Skew Optimization," *IEEE Trans. Computers,* vol. 39, no. 7, pp. 945-951, July 1990.

[12] X. Liu, M.C. Papaefthymiou, and E.G. Friedman, "Retiming and Clock Scheduling for Digital Circuit Optimization," *IEEE Trans. CAD of Integrated Circuits and Systems,* vol. 21, no. 2, pp. 184-203, 2002.

[13] E. Antelo, T. Lang, M. Montuschi, and A. Nannarelli, "Fast Radix-4 Retimed Division with Selection by Comparisons," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures, and Processors,* pp. 185-196, 2002.

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

[15] E. Antelo, T. Lang, M. Montuschi, and A. Nannarelli, Appendix to "Digit-Recurrence Dividers with Reduced Logical Depth," Analysis of the Register Position, Clock Scheduling, and Cycle Time, supplemental material available at http://computer.org/tc/archives/htm, 2005.

[16] M.D. Ercegovac, T. Lang, and P. Montuschi, "Very-High Radix Division with Prescaling and Selection by Rounding," *IEEE Trans. Computers,* vol. 43, no 8, pp. 909-918, Aug. 1994.

Elisardo Antelo received the BS degree in 1991 and the PhD degree in 1995, both in physics, from the Universidade de Santiago de Compostela, Spain. In 1992, he joined the Departamento de Electrónica e Computación at the Universidade de Santiago de Compostela. From 1992 to March 1996, he was an assistant professor and, since March 1996, he has been an associate professor in this department. His research interests are in computer arithmetic and processor engineering. He is a member of the IEEE.

Tomás Lang received the electrical engineering degree from the Universidad de Chile in 1965, the MS degree from the University of California, Berkeley, in 1966, and the PhD degree from Stanford University in 1974. He is a professor in the Department of Electrical and Computer Engineering at the University of California, Irvine. Previously, he was a professor in the Computer Architecture Department at the Polytechnic University of Catalonia, Spain, and a faculty member of the Computer Science Department at the University of California, Los Angeles. His primary research and teaching interests are in digital design and computer architecture with current emphasis on high-speed and low-power numerical processors and multiprocessors. He is the coauthor of two textbooks on digital systems, two research monographs, one IEEE Tutorial, and the author or coauthor of research contributions to scholarly publications and technical conferences. He is a member of the IEEE Computer Society.

Paolo Montuschi graduated with a degree in electronic engineering in 1984 and received the PhD degree in computer engineering in 1989 from the Politecnico di Torino, Italy. Since January 2000, he has been a full professor with the Politecnico di Torino and, since 2003, he has been the chair of the Department of Computer Engineering , Politecnico di Torino. His research interests cover several aspects of computer arithmetic, with a special emphasis on algorithms and architectures for fast elementary function evaluations, and computer graphics, with particular regard to algorithms and architectures for ray tracing and visualization. He is the coauthor of one textbook on computer graphics and the author or coauthor of several papers published in technical conferences and journals. He served on the program committees for the 13th through 16th IEEE Symposia on Computer Arithmetic and is program cochair of the 17th IEEE Symposium on Computer Arithmetic. From 2000 to 2004, he served as an associate editor of the *IEEE Transactions on Computers*. He is a member of the IEEE Computer Society.

Alberto Nannarelli is an associate professor at the Technical University of Denmark. He graduated with a degree in electrical engineering from the University of Roma "La Sapienza," Italy, in 1988 and received the MS and PhD degrees in electrical and computer engineering from the University of California at Irvine in 1995 and 1999, respectively. He worked for SGS-Thomson Microelectronics and for Ericsson Telecom as a design engineer and for Rockwell Semiconductor Systems as a summer intern. From 1999 to 2003, he was with the Department of Electrical Engineering, University of Roma "Tor Vergata," Italy, as a postdoctoral researcher. His research interests include computer arithmetic, computer architecture, and VLSI design. He is a member of the IEEE Computer Society.