A Multi-Radix Approach to Asynchronous Division *

Gianluca Cornetta
Computer Architecture Dept.
Universitat Politècnica de Catalunya
08034 Barcelona—Spain
E-mail: cornetta@ac.upc.es

Jordi Cortadella
Software Dept.
Universitat Politècnica de Catalunya
08034 Barcelona—Spain
E-mail: jordic@lsi.upc.es

Abstract

The speed of high-radix digit-recurrence dividers is mainly determined by the hardware complexity of the quotient-digit selection function. In this paper we present a scheme that combines the area efficiency of bundled data with data-dependent computation time. In this scheme the selection function is very simple and may be implemented using a fast adder. This function speculates the result digit and, when the speculation is incorrect, a correction of the quotient and of the residual must be performed. When the residual satisfies some constraints it is also possible to switch to a higher radix, computing a fraction of the next digit in advance. This results in a division scheme with a variable iteration time and a variable number of iterations and hence with an asynchronous behaviour. Several designs were realized and compared both in terms of execution time and area. The fastest unit considered is a radix-64 divider that may switch to radix 128 or 256. Our evaluations show that $area \times delay$ savings from 25% to 65%, compared to equivalent synchronous designs, may be achieved.

1 Introduction

High-speed arithmetic operations are becoming important also in general purpose processors. While sub-micron technologies are an important step toward addressing the latency problem, they are not a complete solution. As chips become larger, problems such as clock distribution and skew are a major concern. A possible design tendency to tackle these problems in future processors is asynchronization [3]. In this paper we explore a novel design technique suitable for the design of asynchronous arithmetic circuits for division.

Among all the known algorithms for division, the ones involving recurrence [10] exhibit a good tradeoff between performance and area occupation. Such class of algorithms computes a quotient digit per iteration, thus resulting in linear convergence. Consequently, to reduce the number of iterations, it is convenient to use a higher radix for the result digit. Nevertheless, the higher the radix, the higher the complexity of the result-digit selection function. The complexity of the selection function increases the iteration delay as well and eliminates the advantage. Several techniques to improve the execution time of the algorithm have been proposed, including prediction, operands prescaling and overlapping of several selection stages [1, 18, 19, 7, 8, 9, 10].

Previous works related to asynchronous division [19, 13, 12], deal with radix-2 division. This paper, to the authors' knowledge, is the first dealing with a design technique suitable for implementing very-high radix asynchronous dividers. There are several differences between [19, 13, 12] and the algorithm we describe in this paper. The architectures reported in [19, 13, 12] are self-timed loops formed by several pipelined stages. Each stage represents a radix-2 iteration step of the standard recurrence division [10] and operates as soon as the required operands arrive. Synchronization between adjacent stages is achieved by using DCVSL differential logic and a dual-rail handshake protocol [2]. Each stage computes all the possible assimilations of the partial residual in advance, and since this number depends on the maximum quotient digit, this approach is clearly limited to radix 2.

As mentioned previously, the use of a low radix is highly penalizing because it increases the number of iterations necessary to complete a division. The proposed algorithm overcomes this intrinsic limitation of the approaches described in [19, 13, 12] allowing the implementation of very-high radix asynchronous dividers.

The proposed approach is general and suitable for any kind of hardware implementation, including self-timed loops, even if in this paper we present a standard implementation with cascaded CSA stages [10] suitably modified to interact with a bundled-data handshake circuit.

Unlike [19, 13, 12] where each stage is designed to have

*This work has been partially funded by the Ministry of Education of Spain under CICYT, TIC 98-0410, ACiD-WG (ESPRIT 21949), a grant by Intel Corporation and CIRIT 1999SGR-150.

two different delay paths, and to operate as soon as the inputs are valid, the proposed algorithm achieves variable execution times by means of a selection function with data-dependent computation times [5]. This selection function speculates the quotient digit. In case of correct prediction, the completion-detection logic is switched on the delay chain that matches the best-case datapath delay. If the speculation is wrong the error must be corrected. This requires an additional latency and the completion-detection logic is switched on the delay chain matching the worst-case datapath delay.

The correctness of a speculation is checked by an error detection and correction function. However, unlike [5], speculation and error detection are parallel processes, hence a wrong speculation does not need any additional iteration and the execution time results to be shortened also in case of wrong speculation. Parallelism between these processes is achieved by speculating the result digit using the integer part of the shifted partial remainder, whereas the fractional part is used to determine the correctness of the speculation. This also leads to very simple, and hence faster, implementations since the selection function may be realized using an adder.

In a variation of this scheme, we allow the possibility to switch to a higher radix when the partial residual satisfies certain constraints. This permits to increase the number of bits of the result computed in an iteration. This technique, that reminds the approaches proposed in [5, 14] results in an implementation with a variable number of iterations. This approach is a further advantage with respect the implementations described in [19, 13, 12] since those designs perform a division in a fixed number of iterations with a variable iteration delay, whereas the proposed realization performs a division with a variable number of iterations and a variable iteration delay, thus decreasing the overall execution time of the algorithm.

In summary, the variable-delay behaviour is achieved through data-dependent computation times by means of a very simple speculation and error detection and correction function and by a function that permits to switch to a higher radix when the divider input matches certain constraints. Synchronization is achieved by means of bundled-data with a four-phase handshake protocol [2]. This allows to implement division units with a reduced area since we avoid the use of differential logic with dual-rail synchronization.

We develop the method, evaluate alternative possibilities and present some examples of implementation, comparing them with the implementation of conventional and speculative algorithms using the same technological constraints.

This paper is organized as follows. Section 2 summarizes the division algorithm and notation. In Section 3, we introduce the basic idea behind quotient digit speculation as well as the idea to switch to a higher radix when certain constraints are matched. Quotient digit speculation and operand scaling, as a technique to improve the rate of correct predictions, are discussed thoroughly in Section 4. Section 5 gives an example of how the proposed algorithm works. In Section 6, we summarize the evaluation criteria of the implemented units. Section 7 reports the characteristics of a radix-16 unit implementing the proposed algorithm. Finally, in Section 8, we draw up some conclusions. All the implementation details and formal proofs are reported in [4].

2 Division Algorithm and Notation

We now briefly review the well-known division algorithm and outline the notation used along this paper. The standard recurrence used for division is [10]:

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

where $w[j]$ is the full-precision partial residual after the j-th iteration, r is the radix, q_{j+1} is the quotient digit, d the full-precision divisor and x the full-precision dividend. For the sake of simplicity, we assume that x and d are positive normalized fractions and that $x < d$ so that the division always starts with the shift of the partial residual and the quotient Q result a positive normalized fraction as well.

To have a fast iteration, a carry-save adder is used, with the partial residual in carry-save redundant form, although a similar development could be done for other redundant representations of $w[j]$. Redundant quotient digits are converted on-the-fly [6] into a conventional representation.

The quotient digit is a signed digit $|q_{j+1}| \leq a$ with redundancy factor $\rho = \frac{a}{r-1}$ (with $\rho \in (\frac{1}{2}, 1]$). This requires $w[0] \leq \rho d$, which is obtained by shifting the dividend. Moreover, to assure the convergence of the algorithm, the partial residual must be bounded [10], namely:

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

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

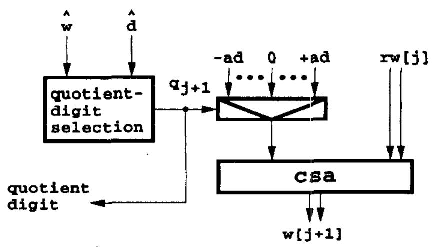
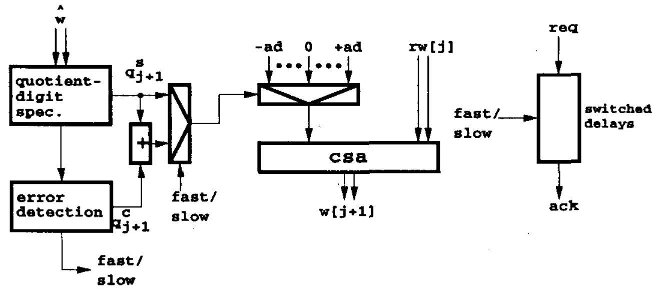


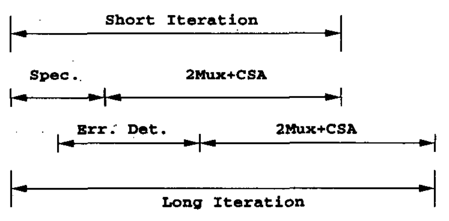
Figure 1. Iteration Step of Digit-Recurrence Division.

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

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



(a)



(b)

Figure 2. Basic Scheme: (a) Block Diagram, (b) Timing Diagram.

The basic scheme that implements the iteration step is shown in Figure 1. There are several ways of implementing the selection function F. Most of them are described in detail in [10]

3 Division Scheme with Quotient Digit Speculation

We now describe the basic scheme, without higher radix switch, and then introduce the scheme with higher radix switch (HRS).

3.1 Basic Scheme

The basic scheme is depicted in Figure 2(a). As already mentioned in the introduction, the quotient digit is speculated. The theory developed to design the speculation function will be described in detail in the next section. The speculated digit q^s_{j+1} is used to compute an *interim residual* $\tilde{w}[j+1] = r \cdot w[j] - q^s_{j+1} \cdot d$. The next residual is then calculated as follows:

$$w[j+1] = \begin{cases} \tilde{w}[j+1] & \text{if } \tilde{w}[j+1] \text{ is bounded} \\ \tilde{w}[j+1] - q^c_{j+1} \cdot d & \text{otherwise} \end{cases} \tag{3}$$

q^c_{j+1} is the correction digit, and is such that $q_{j+1} = q^s_{j+1} + q^c_{j+1}$ with $q^c_{j+1} \in \{-1, +1\}$, whereas in case of correct speculation $q_{j+1} = q^s_{j+1}$. As we will see later in this paper, the bound checking operation, necessary to check whether a speculation is correct or not, is performed by examining $w[j]$ and not $\tilde{w}[j+1]$. This leads to fast execution times. As shown in Figure 2(b) the proposed scheme for division exhibits a variable delay behaviour. Handshake signals are generated using switched delays [15, 16, 11] and a four-phase protocol [2] using *request* (req) and *acknowledge* (ack) as handshake signals. The increased iteration length in case of wrong prediction depends on the hardware complexity of the error-detection function. However since speculation and error detection are overlapped, the time overhead due to a wrong prediction does not affect the performance of the algorithm significantly.

3.2 Scheme with Higher Radix Switch (HRS)

In the basic scheme, two situations may occur: either we have a short-latency iteration if the speculation is correct, or a long-latency iteration if the prediction is wrong and a correction must be carried out. The delay of the implementation is reduced by the scheme with HRS. In this scheme, when the residual is small, a third situation is allowed, namely the switching to a higher radix R, so that a fraction of digit is computed in advance, thus reducing the overall number of iterations necessary to complete the division. Radix R is such that $R = pr$, where r is the original radix and p is a power of 2. The amount of bits of the advance must be selected so that the next residual is bounded. That is, it is possible to calculate $\log_2 p$ extra bits if:

$$|Rw[j+1]| \leq r\rho d.$$

Consequently, taking into account that $R = pr$, we obtain that:

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

Finally, Figure 3 shows the basic scheme of the implementation with HRS. The multiplexer used to select the amount of shifting at the end of an iteration increases the iteration latency, nevertheless the reduction of the overall number of iterations due to the HRS increases the performance of the algorithm.

4 Quotient Digit Speculation and Operands Scaling

As indicated in the previous section, the idea is to reduce the latency of the quotient-digit selection by using a simpler function that gives a correct value with high probability. The complexity of this function is related to its delay, namely the simpler the selection function, the shorter the latency. However, since the probability of a correct prediction decreases as the hardware complexity of the speculation function decreases, all the design space must be accurately

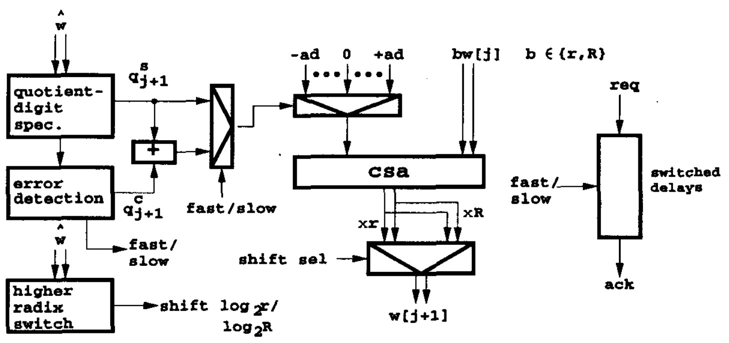


Figure 3. Scheme with Higher Radix Switch.

examined and a tradeoff must be found between accuracy and simplicity.

We develop first the theory for the case of full-precision residuals and divisors and then extend it to the case with residual estimates. In particular, we propose to speculate the quotient digit considering only the integer part of the shifted residual (represented in carry-save form), namely:

$$q^s_{j+1} = Int(rw[j]); \qquad (5)$$

consequently a fast adder may be used to implement the selection function. Since q^s_{j+1} is only a speculation it may be incorrect.

Equation (5) does not depend on d. We can achieve this by operand scaling [8]. Namely we want to find a scaling factor M such that: $X = Mx$ ($X = w[0]$) and $D = Md$. The scaling factor M must be such that the divisor d is scaled in an interval $[D_{min}, D_{max}]$, being $D_{min} = 1 - 2^{-\delta}$ and $D_{max} > D_{min}$, whereas δ is the number of bits of d that must be evaluated in order to compute the scaling factor M. The value of δ must be suitably calculated so that:

$$D_{min} = 1 - 2^{-\delta} \approx 1. \qquad (6)$$

As it will be explained later in this paper, this is necessary to improve the rate of correct predictions of the selection function. The upper bound D_{max} of the scaled divisor range is such that:

$$D_{max} = 1 + \alpha \qquad (7)$$

where $0 < \alpha \ll 1$ depends on ρ, δ and a. In [4] we proved that in order to guarantee the convergence of the algorithm δ must satisfy the following equation:

$$\rho \geq (a + \rho)2^{-\delta}, \qquad (8)$$

whereas α must be such that:

$$\alpha < \frac{1 - \rho(1 - 2^{-\delta})}{a}. \qquad (9)$$

In conclusion, to simplify the selection function and to increase the number of correct predictions, the operands must be preprocessed by a scaling function before executing the recurrence (3). The scaling requires two extra iterations. However this technique is still advantageous because the iteration latency is reduced.

4.1 Selection Function Design

The selection function, speculates and eventually corrects the quotient digit. The error detection and correction is carried out by examining the fractional part $frac(rw[j])$ of the assimilated residual. Considering equation (5), and supposing that $q_{j+1} = q^s_{j+1}$ (hence by equation (3) $w[j+1] = \tilde{w}[j+1]$), the recurrence becomes:

$$\begin{aligned} w[j+1] &= r \cdot w[j] - q^s_{j+1} \cdot d \\ &= (1-d)Int(rw[j]) + frac(rw[j]). \end{aligned}$$

After operand scaling, the recurrence becomes:

$$w[j+1] = (1-D)Int(rw[j]) + frac(rw[j]). \qquad (10)$$

4.1.1 Correct Prediction

Selection of quotient digits is essentially a bound-checking operation where the residual is compared with some bounds that depend on the scaled divisor D. To simplify the selection, making it independent from D, we may restrict the selection bounds imposing that $|w[j]| \leq \rho D_{min}$. The value of D_{min} and hence of δ must be suitably chosen in order to guarantee the convergence of the algorithm. Consequently we may impose that (10) must be bounded. To obtain constant bounds we consider the worst case. In [4] we show how this happens for $D = D_{min}$ and $Int(rw[j]) = a$. Hence we obtain:

$$|frac(rw[j])| \leq (\rho + a)D_{min} - a \qquad (11)$$

If equation (11) holds then the selected digit is $q_{j+1} = q^s_{j+1} = Int(rw[j])$.

4.1.2 Error Detection and Correction

A prediction error occurs when, after a prediction the residual falls out of the bounds fixed by equation (11), or equivalently if:

$$|w[j+1]| > \rho D_{min}.$$

In [4] it has been demonstrated that when this situation occurs the worst-case interim residual is $\tilde{w}[j+1] \approx 1$. Moreover the smallest convergence interval is such that $|w[j+1]| < \frac{1}{2}$, hence to guarantee the convergence of the algorithm the interim residual must be corrected by $\pm D$ and the speculated digit by ± 1.

As a consequence, taking into account equation (11) and imposing that $(\rho + a)D_{min} - a = \overline{B}$ and that $-(\rho + a)D_{min} + a = \underline{B}$, the selection function becomes:

$$q_{j+1} = \begin{cases} s|Int(rw[j])| & \text{if } \underline{B} \leq frac(rw[j]) \leq \overline{B} \\ s|Int(rw[j]) + 1| & \text{otherwise} \end{cases} \tag{12}$$

where s is such that:

$$s = sign(rw[j]) = \begin{cases} +1 & \text{if } rw[j] \geq 0 \\ -1 & \text{if } rw[j] < 0 \end{cases} \tag{13}$$

From equation (11) it is evident that the larger D_{min} (and hence the larger δ), the larger the rate of correct predictions, since this increases the range of the bounds of $frac(rw[j])$.

4.1.3 Using Residual Estimates

Until now we have considered full precision residuals. However, to reduce the adder size and to simplify the selection, we use a truncation $\hat{w}[j]$ to the t-th fractional bit of the full-precision redundant residual to select the quotient digit q_{j+1}. We perform a truncation error that affects the upper bound of the convergence interval. Since the maximum truncation error ε, in case of redundant residuals in carry-save form is 2^{-t+1}, the convergence is assured if:

$$-\rho D_{min} \leq \hat{w}[j] \leq +\rho D_{min} - 2^{-t+1}.$$

Considering this new bounds constraint, in [4] we found that:

$$\rho(r+1)(1-2^{-\delta}) - a(1+\alpha) \geq 2^{-t+1}. \tag{14}$$

Obviously δ must also satisfy the constraint defined by equation (8). Hence for a radix-r divider we need to evaluate $\log_2 r + t + 1$ bits of the residual, in order to speculate and eventually correct a quotient digit. Naturally, $\log_2 r + 1$ is the number of bits of the integer part of the residual. According to these considerations, the selection function with truncated residuals becomes:

$$q_{j+1} = \begin{cases} \hat{s}|Int(r\hat{w}[j])| & \text{if } \underline{B} \leq frac(r\hat{w}[j]) \leq \overline{B} - \varepsilon \\ \hat{s}|Int(r\hat{w}[j]) + 1| & \text{otherwise} \end{cases} \tag{15}$$

Where $\hat{s} = sign(\hat{w}[j])$ and $\varepsilon = 2^{-t+1}$ is the maximum truncation error. Equation (14) fixes a lower bound for t. Also in this case the design space must be accurately explored since t affects the rate of correct predictions. In particular, from equation (15) we note that the larger t, the higher the rate of correct predictions. However a large t implies a large adder and hence an increase of both hardware complexity and latency of the implementation, so a tradeoff must be found.

4.1.4 Simple Scaling

To compute the scaling factors we use an approach very similar to the one described in [8, 10]. Proofs and implementation details are reported in [4]. We want to multiply the divisor d and the dividend x by a radix-4 scaling factor $M = \sum_{i=0}^{3} m_i 4^{-i}$, such that $m_0 \in \{1,2\}$, $m_1 \in \{-1,0,1,2\}$, $m_2 \in \{-2,-1,0,1,2\}$ and $m_3 \in \{-2,-1,0,1,2\}$ and that $Md \geq 1 - 2^{-\delta}$. The radix-4 scaling is the one that offers the best tradeoff between performance and ease of implementation [10]. In [4] we showed that the scaling factors M_n may be computed using the following relation:

$$M_n = \left\lceil \frac{2^{\delta} - 1}{2^{\delta} - 1 - n} \right\rceil_{\delta_s} \tag{16}$$

n being an integer such that $1 \leq n \leq 2^{\delta-1} - 1$, whereas $\lceil \gamma \rceil_{\delta_s} = \frac{1}{2^{\delta_s}} \lceil 2^{\delta_s} \gamma \rceil$ and $\delta_s = 3 \log_2 4$ (3 being the number of fractional radix-4 digits of the scaling factor). We also proved [4] that the scaling factors, and hence δ, limit the maximum digit value a; namely that:

$$a \leq \frac{r-1}{(r-1)\left(\max_{1 \leq n \leq 2^{\delta-1}-1} \left\{ M_n \left(1 - \frac{n}{2^{\delta}}\right)\right\} - 1\right) + 1}. \tag{17}$$

Equation (17) is very important because it determines the value a of the maximum digit and hence the redundancy factor ρ.

5 Example

We give now an example of how the proposed algorithm works. For the sake of simplicity we perform a division with $r = 8$, $a = 7$, $x = (0.6670666)_8$ and $d = D = (0.7644162)_8$. From equations (8), (14) and (15) we deduce that $\delta = 5$ and $t = 4$, hence it results that $D_{min} = 0.96875$ and that a speculation is correct if $-0.75 \leq frac(rw[j]) \leq 0.625$. If $-0.1875 \leq \hat{w}[j] \leq 0.0625$ a change to radix 32 is performed, whereas if $-0.4375 \leq \hat{w}[j] < -0.1875$ or $0.0625 < \hat{w}[j] \leq 0.3125$ the algorithm switches to radix 16. The higher radices are determined by equation (4). Since according to this equation the higher the radix, the smaller the bounds, to obtain an implementation with a switching

$r = 8 \quad a = 7 \quad \rho = 1$				
$x = (0.6670666)_8 \ D = (0.7644162)_8 \ M = 1 \ D_{min} = 0.96875$				
$-0.75 \leq frac(r\hat{w}[j]) \leq 0.625$				
Selection			# Bit Adv. (p)	Iter.
$w[0] =$	$(0.6670666)_8$	$q_0 = 0$	$p = 0$	
$2^p rw[0] =$	$(6.670666)_8$			
$-q_1 D =$	$(-6.6575436)_8$			
$r\hat{w}[0] =$	6.8125			
$w[1] =$	$(0.0111222)_8$	$q_1 = Int(r\hat{w}[0]) + 1 = 7$	$p = 2$	SLOW
$2^p rw[1] =$	$(0.44511)_8$			
$-q_2 D =$	$(-0.0)_8$			
$r\hat{w}[1] =$	0.5			
$w[2] =$	$(0.44511)_8$	$q_2 = Int(r\hat{w}[1]) = 0$	$p = 0$	FAST
$2^p rw[2] =$	$(4.4511)_8$			
$-q_3 D =$	$(-3.722071)_8$			
$r\hat{w}[2] =$	4.5625			
$w[3] =$	$(0.527007)_8$	$q_3 = Int(r\hat{w}[2]) = 4$	$p = 0$	FAST
$2^p rw[3] =$	$(5.27007)_8$			
$-q_4 D =$	$(-4.7065072)_8$			
$r\hat{w}[3] =$	5.3125			
$w[4] =$	$(0.3613606)_8$	$q_4 = Int(r\hat{w}[3]) = 5$	$p = 0$	FAST
$2^p rw[4] =$	$(3.613606)_8$			
$-q_5 D =$	$(-3.722071)_8$			
$r\hat{w}[4] =$	3.75			
$w[5] =$	$(-0.106263)_8$	$q_5 = Int(r\hat{w}[4]) + 1 = 4$	$p = 2$	SLOW
$2^p rw[5] =$	$(-4.31314)_8$			
$-q_5 D =$	$(3.7220710)_8$			
$r\hat{w}[5]) =$	-4.4375			
$w[6] =$	$-(0.371047)_8$	$q_6 = -(\lvert Int(r\hat{w}[5]\rvert + 1) = -4$	$p = 1$	FAST
$Q = (0.704544)_8 = 0.877288$				

Table 1. Example of Radix-8 Division.

probability sufficiently high we have to keep these bounds as large as possible. This limits the advance to 2 bits, and hence the radices to switch to are $2r$ and $4r$, namely $R = 16$ and $R = 32$ in this particular case. Table 1 reports the steps performed by the algorithm to calculate the correct quotient Q. As we will see in more detail in Section 7.1.1, the

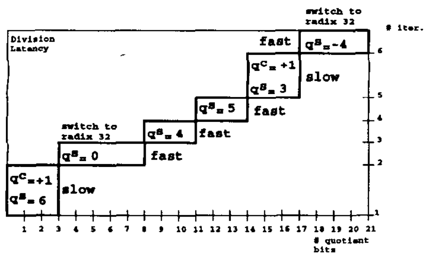


Figure 4. Example of Radix-8 Division with HRS.

adder used for quotient-digit speculation is implemented by cascading two smaller adders that compute the integer and the fractional part of the partial residual in carry-assimilated form. Hence we have three possible iteration times: a *fast iteration* if the prediction is correct and no carry ripples between the two adders, a *medium iteration* if the prediction is correct but a carry is propagated from an adder to the other, and a *slow iteration* if the speculation is incorrect and a correction must be performed. Figure 4 depicts the evolution of the example of Table 1. By observing the figure, it results clear that the algorithm has a variable execution time. This is due to the three different iteration types that determine the latency of an iteration and to the HRS. In fact, the possibility to switch to a higher radix permits to increase the number of bits computed per iteration (as for iterations 2 and 6) reducing the overall number of iterations necessary to complete a division and achieving an asynchronous behaviour. In the example described in this section, the division ends after 6 iterations instead of 7.

6 Evaluation

To evaluate the performance of the schemes that implement the algorithm we have described, we used a standard

	conventional				speculative						
					no adv.		adv.			hrs	
radix	2	16 scal	4×4	512	16	512	32	64	512	16	64
a	1	10	2×2	511	12	320	22	37	320	9	41
iter./digit	1	1	1	2.2	1.3	1.8	1.14	1.21	1.17	0.96	0.98
iter. delay	20.4	28.8	31	40	28.8	43.6	34.2	36	43.8	27.2	35.2
delay/bit	20.4	8.6	7.8	9.8	9.4	8.8	7.8	7.2	5.6	6.5	5.7
cell area	3100	4700	4900	13600	4900	8400	6800	8900	10500	5870	6800
speedup	1.0	2.4	2.6	2.1	2.2	2.3	2.6	2.8	3.6	2.7*	3.1*
area factor	1.0	1.52	1.6	4.4	1.6	2.7	2.2	2.9	3.4	1.9	2.2
delay $\times$ area	1.0	0.63	0.61	2.1	0.73	1.17	0.84	1.03	0.95	0.7	0.71

* Includes two iterations for operand scaling.

Table 2. Characteristics of the Designs.

cell family and designed two double-precision floating-point dividers (the width of the datapath affects only the area estimates). Since a design depends on many parameters, we have not done a complete analysis of the solution space, but performed some reasonable designs to evaluate and compare.

In Section 1, we have dealt with one of the major concerns when designing a recurrence divider, that is the choice of the radix. A high radix may lead to improved performances only if the increase of the iteration time is kept as small as possible. As a consequence a tradeoff between radix and cycle time must be found, since using a high radix does not lead necessarily to improved speedups. For this reason the performances of the different algorithms reported in Table 2 are analysed for several radices, in order to identify the solution that better matches the design constraints.

In particular, in order to compare our designs with the ones described in [10, 5], we have used the same 1-μm standard cell CMOS library and the same design tools. Delays and areas are expressed as multiples of the delay and the area of a two-input NAND gate with a fanout of three NAND gates (the size of a two-input NAND gate is 12.5×47.5 μm^2, the delay of the unloaded gate is 0.24 ns). Some simple modules have been designed by hand (multiplexers and CSA's), whereas SIS [17] has been used for the synthesis of the decoders and error detection function. SIS has always been guided to optimize delay at the expense of increasing the area. Fan-in and fan-out capacitances (but not routing) have been considered for delay calculations. In Table 2, we report the final results. In our estimations we take into account also the extra iterations necessary for operand scaling. We consider the conventional dividers described in [10] and the speculative dividers both with partial advance and no advance reported in [5] and compare these implementations with the speculative asynchronous dividers that realize the proposed algorithm with HRS. We compare all the implementations both in terms of speedup and area and the results are normalized with respect to the conventional radix-2 case. All the measures performed are summarized in Figure 7. To estimate the performance we use three parameters [4]: the iteration delay, the average number of iterations per quotient digit and the average delay per quotient bit. The average number of iterations per quotient digit includes both speculation and correction latencies. In [4] we show how to compute this parameter. Moreover, from Table 2, we note that the proposed schemes have a $delay \times area$ saving of about 65% with respect the conventional radix-512 unit and of about 25% with respect to the speculative radix-512 unit with partial advance. It is important to remark how, from Table 2, the approach with HRS leads to a number of iterations per quotient digit that is smaller than "1", since there are iterations where some quotient bits are computed in advance.

Module	Area
`2 x 3b CSA`	40
`3 x 3b CPA`	66
`1 x 54b 3-input MUX`	240
`1 x 54b 2-input MUX`	160
`1 x 54b CSA`	360
`QSEL`	9
`total`	875

Table 3. Area of Basic Block of a Radix-2 Self-Timed Stage [19].

6.1 Comparison with Other Asynchronous Design Methodologies

Although the designs described in [19, 13, 12] are pipelined and use differential DCVSL gates and a dual-rail handshake for stage synchronization, it would be interesting to perform some rough performance and area estimations in order to compare these design methodologies with the one proposed in this paper. Since the designs reported in [13, 12] are shared units for division and square root we prefer to compare our implementation only with [19].

Table 3 reports normalized delays and area of the basic blocks that form a stage of the self-timed loop. In [19], it has been determined an analytical expression to compute the

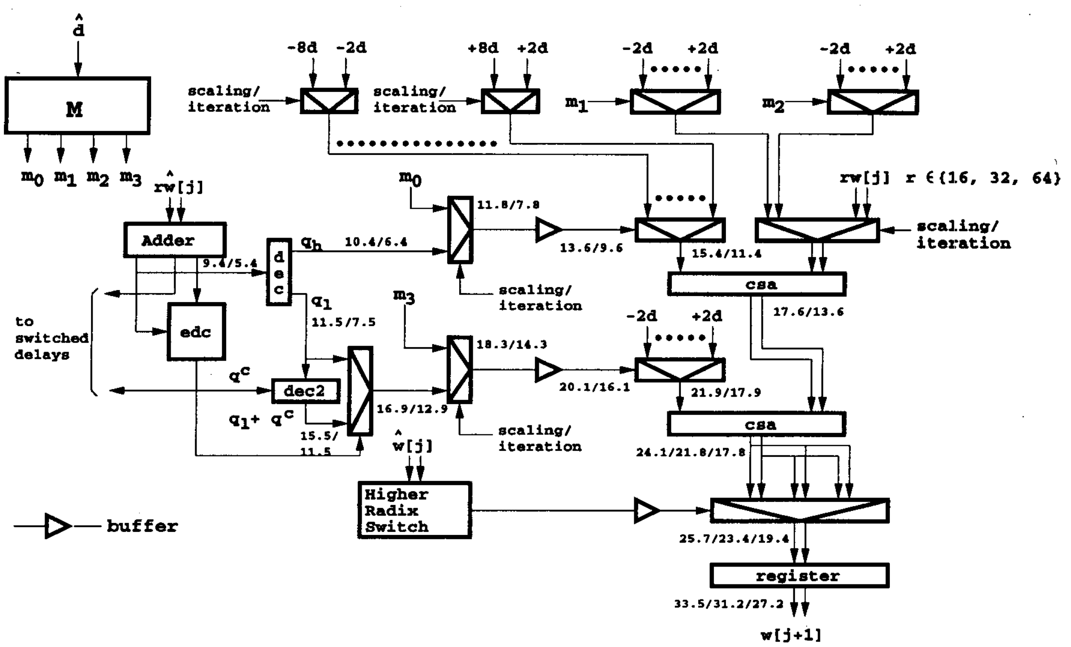


Figure 5. Block Diagram for a Speculative Radix-16 Divider with HRS.

average case delay of each stage of the pipeline that according to our estimations is about 11.5 2-input NANDs delay. Table 4 reports the estimated average delay and area occupation of the schemes proposed in [19]. Hence, according to

Module	Area	Delay
5 × Div. Stage	4375	11.5
1 × 54b 2-input MUX	160	1.4
2 × 54b register	430	not critical
total	4965	648
speedup	1.7	
area factor	1.6	

Table 4. Characteristics of the Self-Timed Radix-2 Divider [19].

our estimations, the pipelined divider with self-timed loop has a speedup of 1.7 and an area factor of 1.6 with respect a conventional radix-2 synchronous divider but is still uncompetitive with the design proposed in this paper for which we estimated a speedup of 3.1 with respect a conventional radix-2 synchronous divider.

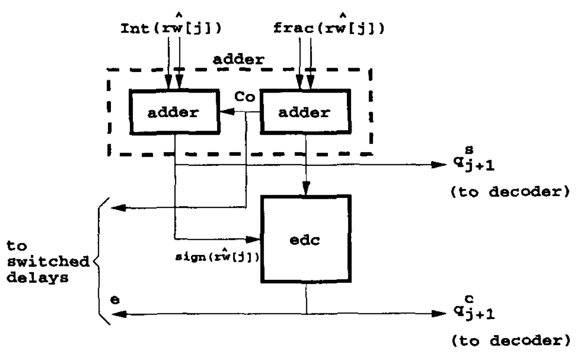
7 A Design Example

This section describes the implementation of a radix-16 divider with HRS. Further details, as well as the description of a radix-64 unit for division, may be found in [4].

7.1 Radix-16 with HRS

We preferred a bundled-data approach basically for two reasons. The former is to limit the area occupation of the implementations, thus achieving better $area \times delay$ products. The latter is the need to compare the proposed design approach with the ones reported in [10, 5] using objective criteria. We think that further investigation is necessary to study the impact of a dual-rail implementation on the performances of the proposed algorithm, since the increased wiring load typical of such realization may not lead necessarily to a better throughput. The block diagram of the proposed implementation is shown in Figure 5, and the characteristics are shown in Table 2. The quotient digit speculated by the selection function is $q_h + q_l$, where $q_h \in \{-8, -4, 0, +4, +8\}$ and $q_l \in \{-1, 0, +1\}$. Therefore we choose $a = 9$, since, according to equation (17) and to the design parameters of Table 2, results that $a \leq 10$. Moreover, in this way we avoid the generation of $q_h + q_l = 11$ that would imply the use of three carry-save adders increasing both iteration time and hardware complexity. Table 5 reports area and delay characteristics of this design. Since q_h is the highest-weight component of the quotient digit, the decoding logic DEC is simpler and faster than for q_l. During the synthesis of the decoder DEC, SIS has been guided to reducing the delay of q_h since it is the critical path.

Note that two modules are required: one for quotient digit speculation and one for switching to an other radix. The latter controls the amount of the shift at the end of an iteration and permits an advance of one bit in case of switch-



e	Co	Iter. Type
0	0	fast
0	1	medium
1	—	slow

Figure 6. Block Diagram of the Selection Logic.

Module	Area	Delay
speculation	115	10.4/6.4* (for q_h) 11.5/7.5 (for q_l)
error detection	22	15.5/11.5
mux (for q_jd)	2 × 302	1.8*
mux (scaling)	1650	1.4*, 1.6 or 1.8
mux (err. det.)	12	1.4
mux (HRS)	230	1.6*
csa	2 × 360	4.2* (from $rw[j]$) 2.2* (from q_jd)
buffers	8 × 2.6	1.8*
HRS	26	not critical
switched delays	71	not critical
registers convert [10]	170 × 3 1900	7.8*
total	5870	27.2

* Denote blocks in the critical path.

Table 5. Area and Delay of the Radix-16 Scheme with HRS.

ing to radix 32 or two bits in case of switching to radix 64. These two modules are conceptually identical, but evaluate different ranges of $w[j]$. The CSA has been designed as a radix-2 full adder and its worst-case delay is determined by two cascaded XOR gates. However, the outputs of the q_jd multiplexers have been connected to the last gate in order to reduce the critical path delay (the same optimization has been used in the other designs). This approach may not be used for the residual since it is represented in redundant form and requires two signals. Moreover, to save hardware, we use the divider carry-save chain to compute the scaling factors as well, this implies the insertion of multiplexers to switch between scaling and division iteration.

For what concerns the computation of the delays that must be matched by the switched-delay chain we assume that all the gates along the three possible delay paths have a propagation delay equal to the worst-case one. It must be remarked that, in order to compare our designs with [10, 5], in our performance analysis we neglect the delay due to wiring. This means that we give only a rough estimation of the real delays.

We think that more realistic estimations may be performed by assuming a draft layout plan in order to compute the worst-case load due to wires, that, especially in the case of components with large fan-out such as the buffers that drive the q_jd multiplexers, may lead to significant delays.

7.1.1 Selection Function

The block diagram is shown in Figure 6. The integer and the fractional part of $r\hat{w}[j]$ are assimilated by two fast adders, so that the carry-out signal Co may be used jointly with signal the error signal e to select the iteration type as shown in Figure 6. In fact, when Co is "0", digit q^s_{j+1} is generated after the delay of one adder, whereas when Co is "1", q^s_{j+1} is generated after the delay of two adders.

Digit q^s_{j+1} is represented in two's complement on $\log_2 r+1$ bits, whereas digit $q^c_{j+1} \in \{-1,+1\}$ is encoded on two bits (each line encodes one of the possible values).

8 Summary and Conclusions

The division method that has been presented in this paper is based on the speculation of the result digit. Speculation is performed by using an adder, this results in implementations that are faster than conventional ones and that require less hardware resources with respect to other speculative approaches analyzed in this paper. The selection function permits to switch between three kinds of iterations: a slow, a medium and a fast one. The slow iteration is the one that is performed when a correction has to be carried out. The performance of the algorithm may be enhanced using HRS, namely switching to a higher radix when the partial residual falls into certain bounds. The HRS permits to obtain implementation with a variable number iterations. We performed several designs using the same technology and determined the relative speed and area. The results are summarized in Figure 7.

From the analysis of the experimental results reported in Figure 7 we may infer that the proposed approach (HRS)

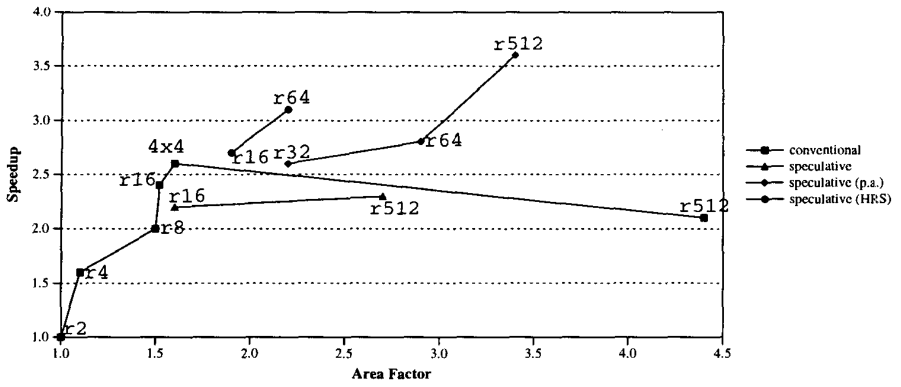


Figure 7. Summary of the Implementations for Different Dividers.

has, in case of radix-64, a latency comparable with the speculative radix-512 divider with partial advance (denoted with "p.a.") described in [5] but an area saving of about 35%. In addition, as showed in Section 6, the schemes that implement the proposed algorithm have a $delay \times area$ saving of about 65% with respect the conventional radix-512 unit and of about 25% with respect to the speculative radix-512 unit with partial advance. They also outperform the efficiency of other known asynchronous designs.

References

[1] D. E. Atkins. Design of the Arithmetic Unit of Illiac III: Use of Redundancy and Higher Radix Methods. *IEEE Transaction on Computers*, C-19(8):720–733, August 1970.

[2] J. A. Brzozowski and C.-J. H. Seger. *Asynchronous Circuits*. Springer-Verlag, New York, 1994.

[3] B. Chappell. The Fine Art of IC Design. *IEEE Spectrum*, 36(7):30–34, July 1999.

[4] G. Cornetta and J. Cortadella. Multi-Radix Asynchronous Dividers. Technical Report UPC-DAC-2000-57, Universitat Politècnica de Catalunya, Departament d'Arquitectura de Computadors, September 2000.

[5] J. Cortadella and T. Lang. High-Radix Division and Square Root with Speculation. *IEEE Transaction on Computers*, C-43(8):919–931, August 1994.

[6] M. D. Ercegovac and T. Lang. On the Fly Conversion of Redundant into Conventional Representations. *IEEE Transaction on Computers*, C-36(7):895–897, July 1987.

[7] M. D. Ercegovac and T. Lang. Fast Radix-2 Division with Quotient Digit Prediction. *Journal of VLSI Signal Processing*, 2(1):169–180, January 1989.

[8] M. D. Ercegovac and T. Lang. Simple Radix-4 Division with Operands Scaling. *IEEE Transaction on Computers*, C-39(9):1204–1207, September 1990.

[9] M. D. Ercegovac, T. Lang, and P. Montuschi. Very-High Radix Division with Prescaling and Selection by Rounding. *IEEE Transaction on Computers*, C-43(8):909–918, August 1994.

[10] M.D. Ercegovac and T. Lang. *Division and Square Root. Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Norwell, MA, 1994.

[11] D. Kearney and N. W. Bergmann. Bundled Data Asynchronous Multipliers with Data Dependent Computation Times. In *3rd IEEE Symposium on Advanced Research in Asynchronous Circuit and Systems*, pages 186–197, 1997.

[12] G. Matsubara and N. Ide. A Low Power Zero-Overhead Self-Timed Division and Square Root Unit Combining a Singe-Rail Static Circuit with a Dual-Rail Dynamic Circuit. In *Symposium on Asynchronous Design Methodologies*, pages 198–209, 1997.

[13] G. Matsubara, N. Ide, et al. 30-ns 55-b Radix-2 Division and Square Root Using a Self-Timed Circuit. In *12th Symposium on Computer Arithmetic*, pages 198–209, 1995.

[14] P. Montuschi and T. Lang. Boosting Very High-Radix Division with Prescaling and Selection By Rounding. In *14th IEEE Symposium on Computer Arithmetic*, pages 52–59, 1999.

[15] S. M. Nowick. Design of a Low-Latency Asynchronous Adder Using Speculative Completion. *IEE Proceedings on Computer Digital Techniques*, 143(5):301–307, September 1996.

[16] S. M. Nowick, K. Y. Yun, P. A. Beerel, and A. E. Dooply. Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders. In *Symposium on Asynchronous Design Methodologies*, pages 210–223, 1997.

[17] E. M Sentovich, K. J. Singh, Lavagno L., et al. SIS: A System for Sequential Circuit Synthesis. Technical report, Electronics Research Laboratory-EECS Department University of California, Berkeley, May 1992.

[18] G. S. Taylor. Radix-16 SRT dividers with Overlapped Quotient Selection Stages. In *7th IEEE Symposium on Computer Arithmetic*, pages 64–71, 1985.

[19] T. E. Williams and M. Horowitz. A 160ns 54-bit CMOS Division Implementation Using Self-Timing and Simmetrically Overlapped SRT Stages. In *10th IEEE Symposium on Computer Arithmetic*, pages 210–217, 1991.