Journal of VLSI Signal Processing 40, 109–123, 2005

High-Radix Logarithm with Selection by Rounding: Algorithm and Implementation

J.-A. PIÑEIRO
Department of Electronic and Computer Engineering, University of Santiago de Compostela, Spain

M.D. ERCEGOVAC
Computer Science Department, University of California, Los Angeles (UCLA), USA

J.D. BRUGUERA
Department of Electronic and Computer Engineering, University of Santiago de Compostela, Spain

Received November 15, 2002; Revised November 11, 2003; Accepted November 13, 2003

Abstract. A high-radix digit-recurrence algorithm for the computation of the logarithm, and an analysis of the tradeoffs between area and speed for its implementation, are presented in this paper. Selection by rounding is used in iterations $j \geq 2$, and by table look-up in the first iteration. A sequential architecture is proposed, and estimates of the execution time and hardware requirements are obtained for $n = 16, 24, 32, 53$ and 64 bits of precision and for radix values from $r = 8$ to $r = 1024$. These estimates are obtained according to an approximate model for the delay and area of the main logic blocks. We show that the most efficient implementations are obtained for radices ranging from $r = 32$ to $r = 256$, reducing the execution time by half with respect to a radix-4 implementation with redundant arithmetic.

Keywords: logarithm, digit-recurrence, high-radix, selection by rounding, computer arithmetic

1. Introduction

Logarithms belong to a type of mathematical functions known as *elementary functions*, and are important for a wide range of applications such as engineering, physics, computational chemistry, logarithmic number systems (LNS), or signal processing [1, 2]. Other elementary functions are square root, reciprocal square root, exponentials and trigonometric functions, all of them employed in scientific computations, DSP and 3D-graphics applications [3–5].

Software routines have been used to evaluate elementary functions with techniques such as polynomial and rational approximations [6–8] and continued fraction expansion [9]. However, although these routines provide very accurate results, they are often too slow for numerically intensive and real-time applications.

Hardware-based methods have been developed as an alternative to the software routines, providing high-speed solutions implemented in dedicated hardware. Low-degree polynomial approximations combined with table look-up [10, 11], functional iteration methods [4, 12, 13] and digit-recurrence algorithms [14, 15] are examples of these methods.

Digit-recurrence algorithms are an interesting alternative due to their low area requirements, especially for high-precision computations, when compared with table-based and functional iteration methods. Their main drawback is a linear convergence of one radix-r digit per step, resulting in long execution times for small radices and high precision. Two factors

determine the execution time: the cycle time of the recurrence and the latency of the algorithm. The cycle time can be reduced by using redundant arithmetic, and the latency can be reduced by a factor of b by employing radix $r = 2^b$. However, these improvements lead to an increase in the cost of implementation and, in particular, the complexity of the selection function for the digits. The only practical method for selection when a high radix is used is selection by rounding [3, 16]. Two alternatives have been used to allow selection by rounding: performing a scaling of the recurrence [16, 17] and performing selection by table in the first iterations until the convergence conditions are met [3].

High-radix digit-recurrence methods have been proposed for the computation of division, square root, reciprocal square root, exponential and trigonometric functions [3, 16–19]. In this paper we give a detailed description of the high-radix algorithm for the computation of the logarithm, with selection by rounding and redundant arithmetic [20]. An abbreviated version was presented in [21]. Convergence conditions limit the use of selection by rounding to iterations $j \geq 2$, with a redundant digit set $|e_j| \leq (r-1)$. The selection of e_1 is performed by table look-up. This table is addressed by the $b+1$ most significant bits of the input operand X.

The overall hardware requirements in the high-radix algorithms increase with the radix, and an analysis of the tradeoffs between area and speed is necessary for determining the values of the radix r which result in efficient implementations. We perform such an analysis in this paper, extended from the one presented in [22], for a sequential architecture implementing the proposed high-radix algorithm for logarithm computation. Some optimizations in the architecture are made in this paper, in order to reduce the delay of the critical path and therefore the overall execution time, increasing the achieved speed-ups regarding low-radix implementations. The analysis performed is based on estimates obtained for precisions $n = 16, 24, 32, 53$ and 64-bits and for radix values from $r = 8$ to $r = 1024$, according to an approximate model for the delay and area cost of the main logic blocks employed in the proposed architecture.

The algorithm for logarithm computation is explained in Section 2, detailing the conditions for selection by rounding. In Section 3, the optimized sequential architecture implementing our algorithm is proposed. The model for the delay and area of the main components of the architecture is presented in Section 4, together with estimates of the execution time and area requirements for several precisions and radix values. A comparison of our algorithm with a conventional radix-4 redundant implementation is presented in Section 5. Finally, the main contributions made in this paper are summarized in Section 6.

2. Algorithm

The algorithm is based on the identity

$$\ln(X) = \ln\left(X \prod f_j\right) - \sum \ln(f_j) \tag{1}$$

which has been frequently used in the literature [14, 23–26].

If the following condition is satisfied:

$$X \prod f_j \to 1 \tag{2}$$

then

$$Y - \sum \ln(f_j) \to Y + \ln(X) \tag{3}$$

The condition (2) can be achieved using a *multiplicative normalization* which consists of determining a sequence f_j such that X is transformed to 1 by successive multiplications. To simplify the multiplications, it is convenient to define the constants as $f_j = 1 + e_j r^{-j}$, where $r = 2^b$ is the radix, and e_j is a radix-r digit. This form of f_j allows the use of a *shift-and-add* implementation. The corresponding recurrences for transforming X and computing the logarithm are

$$\begin{aligned} E[j+1] &= E[j](1 + e_j r^{-j}) \\ L[j+1] &= L[j] - \ln(1 + e_j r^{-j}) \end{aligned} \tag{4}$$

with $j \geq 1$, $E[1] = X$ and $L[1] = Y$. The digits e_j are selected so that $E[j+1]$ converges to 1 while b bits of the final result are obtained per iteration. For a result accurate to n bits, a total number of $N = \lceil n/b \rceil$ iterations are necessary. After performing the last iteration of the recurrence the results are:

$$\begin{aligned} E[N+1] &\approx 1 \\ L[N+1] &\approx Y + \ln(X) \end{aligned} \tag{5}$$

Although the proposed algorithm is designed for the computation of the natural logarithm, once such value has been obtained, the logarithm in any base β can

be computed by using the well-known mathematical property:

$$\log_\beta(X) = \log_\beta(e)\ln(X) \tag{6}$$

To have the selection function for e_j dependent on the same bit positions in all iterations, a scaled remainder (residual) is defined as

$$W[j] = r^j(E[j] - 1) \tag{7}$$

and the recurrence on E is replaced by the residual recurrence

$$\begin{aligned} W[j+1] &= r(W[j] + e_j + e_j W[j] r^{-j}) \\ L[j+1] &= L[j] - \ln(1 + e_j r^{-j}) \end{aligned} \tag{8}$$

with $j \geq 1$, $W[1] = r(X - 1)$ and $L[1] = Y$.

The digits e_j are selected as a function of the leading digits of the scaled residual in such a way that the residual $W[j]$ remains bounded.

For a faster execution of the recurrences, a redundant representation of the residual is used making the delay for addition and multiplication independent of the precision. A redundant digit set is used for e_j to simplify the selection function as discussed next.

The conversion from redundant to conventional representation, if required, can be performed using an on-the-fly method [27], avoiding the need for a carry-propagate addition. Moreover, this on-the-fly unit can also perform the exact rounding of $\ln(X)$, using the information of the available remainder to select the correct rounding direction for the result [28].

2.1. *Selection by Rounding*

Since we are considering a high radix, for the selection of the digits e_j we use selection by rounding to the integer part of the residual, similar to [3, 14, 16, 18, 19]. Since the residual $W[j]$ is in a redundant form, the rounding is performed on an estimate $\hat{W}[j]$. The estimate is obtained by truncating the signed-digit representation $W[j]$ to t fractional bits (it would be equivalent using *carry–save* representation for the residual $W[j]$, and performing the selection on an estimate $\hat{W}[j]$ obtained by truncating $W[j] + 2^{-t}$; that is, adding always a 1 in the position with weight 2^{-t} before rounding).

The selection function is

$$e_j = -round(\hat{W}[j]), \tag{9}$$

The sign of the digit e_j is defined as opposite of the sign of $W[j]$ in order to satisfy a bound on the residual, and thus assuring the convergence. The digit set is $e_j \in \{-(r-1), \ldots, -1, 0, 1, \ldots, (r-1)\}$.

When e_j is selected by rounding as indicated in (9),

$$-\frac{1}{2} - 2^{-t} \leq W[j] + e_j \leq \frac{1}{2} \tag{10}$$

Since the maximum value of $|e_j|$ obtained by the rounding scheme must be $(r-1)$, the value of the estimated residual is bounded by

$$-r + \frac{1}{2} \leq \hat{W}[j] < r - \frac{1}{2}, \tag{11}$$

which results in the condition

$$-r + \frac{1}{2} \leq W[j+1] < r - \frac{1}{2} - 2^{-t} \tag{12}$$

Since

$$\begin{aligned} W[j+1] = r(W[j] + e_j) + e_j r^{-j+1} \\ \times (W[j] + e_j - e_j), \end{aligned} \tag{13}$$

the conditions for convergence are the following:

$$\begin{aligned} \frac{r}{2} + e_j r^{-j+1}\left(\frac{1}{2} - e_j\right) < \min\Bigg(r - \frac{1}{2} - 2^{-t}, \\ P[j+1]\Bigg) \end{aligned} \tag{14}$$

and

$$\begin{aligned} r\left(-\frac{1}{2} - 2^{-t}\right) + e_j r^{-j+1}\left(-\frac{1}{2} - 2^{-t} - e_j\right) \\ \geq \max\left(-r + \frac{1}{2}, Q[j+1]\right) \end{aligned} \tag{15}$$

where $P[j+1]$ and $Q[j+1]$ are the positive and negative ranges of convergence for iteration $j+1$:

$$P[j+1] = r^{j+1}\left(\prod_{k=j+1}^{\infty}(1 + (r-1)r^{-k}) - 1\right), \tag{16}$$

and

$$Q[j+1] = r^{j+1}\left(\prod_{k=j+1}^{\infty}(1-(r-1)r^{-k})-1\right) \quad (17)$$

Since $P[j+1] > r-\frac{1}{2}-2^{-t}$ and $Q[j+1] < -r+\frac{1}{2}$, the conditions for convergence are, on one hand,

$$\frac{r}{2}-\frac{1}{2}-2^{-t}+e_j^2 r^{-j+1}+\frac{1}{2}e_j r^{-j+1} > 0, \quad (18)$$

and on the other:

$$\frac{r}{2}-\frac{1}{2}-2^{-t}r-e_j r^{-j+1}\left(-\frac{1}{2}-2^{-t}+e_j\right) \geq 0 \quad (19)$$

The main concern at this point is to minimize j, the number of iterations when selection by rounding cannot be performed. Once this minimum value has been obtained, the corresponding minimum value of t must be determined and, finally, the corresponding minimum value of r.

The numerical analysis of conditions (18) and (19) has been performed with the computer algebra system *Maple* [29]. This analysis shows that the first condition is satisfied for $j \geq 1$, $t \geq 0$ and any positive radix $r \geq 2$. On the other hand, the condition (19) is satisfied if $j \geq 3$, $t \geq 2$ and $r \geq 8$. Therefore, the selection function by rounding is only valid for iterations $j \geq 3$.

The use of two look-up tables for $j = 1$ and $j = 2$ would result in a significant increase in the overall hardware requirements. Therefore, we take advantage of the fact that selection by rounding can be used if the values of digit e_2 are restricted thus avoiding the use of the look-up table for $j = 2$. If the values $j = 2$ and $t = 2$ (worst case for t value) are substituted in condition (19), the following constraint is obtained:

$$\frac{r}{4}-\frac{1}{2}-\frac{3}{4}e_2 r^{-1}-e_2^2 r^{-1} \geq 0, \quad (20)$$

The roots of this inequation are:

$$e_2 = -\frac{3}{8} \pm \frac{1}{8}\sqrt{16r^2-32r+9} \quad (21)$$

Therefore, if e_1 is read from a look-up table addressed by the input operand X so that

$$|e_2| \leq \frac{\sqrt{3}}{2}\frac{r}{2}-\frac{3}{8}, \quad (22)$$

selection by rounding can be used also for $j = 2$. Note that the value (22) is slightly less than $r/2$, which means that the digit set has been reduced by half. This solution results in a significant reduction in the hardware requirements regarding the use of a look-up table for $j = 2$, although it also results in an over-redundant digit e_1.

In summary:

- For iterations $j \geq 3$ convergence is guaranteed with selection by rounding and a digit set $-(r-1) \leq e_j \leq r-1$.
- For iteration $j = 2$, convergence is achieved with selection by rounding if $|e_2| \leq \frac{\sqrt{3}}{2}\frac{r}{2}-\frac{3}{8}$.
- Iteration $j = 1$ does not converge with selection by rounding, so selection by table look-up is performed, in such a way so that the constraints in the value of $|e_2|$ are enforced.

2.2. *Selection of e_1 by Table Look-Up*

The parameters and constraints for the selection by a table of the first digit e_1 are obtained in this section. We call (A_L, A_U) the interval of convergence for $W[2]$. Therefore, the selection by table in iteration $j = 1$ must guarantee that $A_L < W[2] < A_U$. The values for these bounds are:

$$A_L = e_{2\min}-\frac{1}{2}, \quad A_U = e_{2\max}+\frac{1}{2}-2^{-t} \quad (23)$$

with the values of $e_{2\min}$ and $e_{2\max}$ set by condition (22).

The first iteration for the residual recurrence is

$$W[2] = r(W[1]+e_1+e_1 W[1]r^{-1}) \quad (24)$$

and since $W[1] = r(X-1)$:

$$W[2] = r^2(X-1)+re_1 X \quad (25)$$

We call q the number of fractional input bits to the table. For each interval $[1+v2^{-q}, 1+(v+1)2^{-q})$ of the input operand X, with $1 \leq X < 2$ and $v \geq 0$ integer, a corresponding e_1 value must be read from the table, verifying the following conditions:

$$A_L < r^2 v 2^{-q}+e_1 r(1+v2^{-q}), \quad (26)$$

and

$$r^2(v+1)2^{-q}+e_1 r(1+(v+1)2^{-q}) < A_U \quad (27)$$

These bounds result in the following condition:

$$\frac{A_L - r^2 v 2^{-q}}{r(1 + v2^{-q})} < e_1 < \frac{A_U - r^2(v+1)2^{-q}}{r(1 + (v+1)2^{-q})} \quad (28)$$

The value of q must guarantee that the difference between the upper bound and the lower bound on e_1 is always positive:

$$DIFF = \frac{A_U - r^2(v+1)2^{-q}}{r(1 + (v+1)2^{-q})} - \frac{A_L - r^2 v 2^{-q}}{r(1 + v2^{-q})}, \quad (29)$$

which results in

$$DIFF = \frac{A_U - A_L + A_U v 2^{-q} - A_L(v+1)2^{-q} - r^2 2^{-q}}{r(1 + v2^{-q})(1 + (v+1)2^{-q})} \quad (30)$$

The minimum value of q that satisfies this constraint is $q = b + 1$. Therefore, $b + 1$ bits of the input operand X are necessary to address the initial look-up table for the selection of the digit e_1.

2.3. *Approximation of Logarithm Constants*

The constants $-\ln(1 + e_j r^{-j})$ are usually stored in look-up tables. As the value of the radix r increases, the size of these tables can become prohibitive. Therefore, any effort done in reducing the size of the tables may lead to a significant reduction in the overall hardware requirements, especially for very-high radix values.

Let us consider the series expansion of the logarithm function $\ln(1 + x)$:

$$\ln(1 + x) \approx x - \frac{x^2}{2} + \cdots \quad (31)$$

After iteration $j = k_1$, the values $-\ln(1 + e_j r^{-j})$ can be approximated by $-e_j r^{-j}$, which can be implemented by wired shifts of the digits e_j with opposite sign [18, 20, 30].

We now determine the value of k_1. For a precision of n bits, the series approximation can be used in the iterations when the constraint $x^2/2 < 2^{-n}$ is met:

$$\frac{1}{2} e_j^2 r^{-2j} < 2^{-n} \quad (32)$$

Since $|e_j| < r$, then $e_j^2 < r^2$, and taking into account that $r = 2^b$, the bound becomes:

$$2^{2b(-j+1)} < 2^{-n+1}, \quad (33)$$

which results in

$$j > \frac{n-1}{2b} + 1 \quad (34)$$

Taking into account that $N = \lceil \frac{n}{b} \rceil$ is the total number of iterations, we can conclude that the approximation can be used in iterations $j \geq k_1$ where

$$k_1 = \left\lceil \frac{N}{2} \right\rceil + 1 \quad (35)$$

Summarizing, for $j = 1$ a table storing $-\ln(1 + e_1 r^{-1})$ must be used, the next $N_1 = \lfloor \frac{N-1}{2} \rfloor$ iterations a table storing $-\ln(1 + e_j r^{-j})$ is necessary, and in the last $N_2 = \lceil \frac{N-1}{2} \rceil$ iterations the approximation $-e_j r^{-j}$ is used instead.

3. Architecture

In this section we propose a sequential architecture for the computation of the logarithm based on the algorithm presented in the previous section. This type of architecture is usually suitable for general purpose applications, although a pipelined version of the architecture could be also proposed for numerically intensive applications or specific-purpose implementations. The latency is N cycles, while the throughput is one result per N cycles. Multiplexers are inserted at the input of some logic blocks to allow their reutilization for performing different computations depending on the iteration.

Let us summarize the steps involved in the high-radix algorithm proposed for the computation of the logarithm:

- First iteration ($j = 1$):

$$W[2] = r^2(X - 1) + r e_1 X$$
$$L[2] = Y - \ln(1 + e_1 r^{-1})$$

since $W[1] = r(X - 1)$ and $L[1] = Y$, with e_1 selected by table look-up in such a way that condition (22) is satisfied, and a restricted digit-set is used

in the second iteration. The $b+1$ most significant bits of the input operand X are employed to address the tables $TAB(re_1)$ and $TAB(-\ln(1+e_1 r^{-1}))$.

- Iterations $j=2$ to $j=\lceil \frac{N}{2} \rceil$ ($\lfloor \frac{N-1}{2} \rfloor$ iterations):

$$W[j+1] = r(W[j] + e_j + e_j W[j] r^{-j})$$
$$L[j+1] = L[j] - \ln(1 + e_j r^{-j})$$

with e_j selected by rounding the truncated residual $\hat{W}[j]$ and the constants $-\ln(1+e_j r^{-j})$ read from a look-up table addressed by the digits e_j and the iteration index j.

- Iterations $j=\lceil \frac{N}{2} \rceil + 1$ to $j=N$ ($\lceil \frac{N-1}{2} \rceil$ iterations):

$$W[j+1] = r(W[j] + e_j + e_j W[j] r^{-j})$$
$$L[j+1] = L[j] - e_j r^{-j}$$

with e_j selected by rounding the truncated residual $\hat{W}[j]$.

Figure 1 shows the block diagram of the proposed architecture, with double lines for SD operands, thin for digit-serial operands, and thick for parallel operands. $TAB(re_1)$ and $TAB(-\ln\_e_1)$ are the look-up tables storing re_1 and $-\ln(1+e_1 r^{-1})$, respectively, addressed by the $b+1$ most significant bits of the input operand X. Three main optimizations have been made regarding the architecture proposed in [21, 22]: (i) a SD multiply-add unit is used in the $W[j]$ recurrence, instead of separated SD multiplier and SD adder, (ii) the recoding to SD-4 of the multiplier operand in the $W[j]$ recurrence is done outside the multiply-add unit (by the *round&rec* unit, and directly storing of e_1 in SD-4 representation in the initial look-up table), and (iii) the on-the-fly conversion of the result from redundant to conventional representation and final rounding are included in the architecture [27, 28].

The main features of this architecture are:

- All variables are in redundant representation to allow faster execution of iterations, since the additions become independent of the precision. Signed-digit (SD) representation is used in our architecture, although a similar approach could be proposed with the use of carry-save (CS) representation [31].
- All products of the type re_j, $r^2(X-1)$ or $rW[j]$ are performed as wired shifts, since $r=2^b$, while products of the type $W[j]r^{-j}$ and $e_j r^{-j}$ are also performed as shifts, but barrel shifters must be used to carry out these computations.
- g guard bits are used in the look-up tables, registers and other main logic blocks to prevent the truncation error from affecting the final result. Since N is the total number of iterations, $g = \lceil \log_2(N) \rceil + 1$ bits.
- SDAα is a *signed-digit binary adder* with α input bit-vectors. The addition of two SD operands requires a SDA4 adder, since the input operands are represented by two bit-vectors each. SDA3 adders are used in our architecture for accumulating an operand in SD

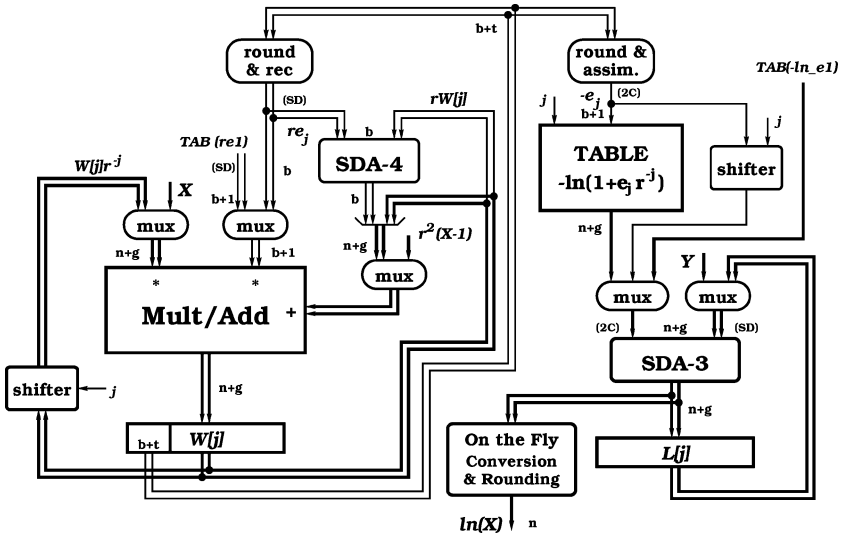

Figure 1. Block diagram of the proposed architecture.

representation and an operand in two's complement ($2C$) representation.
- The multiplier operand in the multiply-add unit is in SD radix-4 representation to reduce by half the number of partial products to be accumulated by the tree. However, since the multiplicand is represented in SD radix-2, the size of this unit is roughly double of a regular multiplier with non-redundant multiplicand, and one extra level in the partial products accumulation tree is required.
- The constants $-\ln(1+e_j r^{-j})$ are approximated by $-e_j r^{-j}$ for iterations $j \geq \lceil \frac{N}{2} \rceil + 1$. Therefore, no table is needed in those iterations.
- The selection by rounding of digits e_j is performed by two different units. The first one, the *round&rec* unit, takes the $(b+t)$ most significant bits of $W[j]$ ($t=2$) and produces a SD radix-4 representation of the digit e_j, to be used as multiplier in the multiply-add unit. On the other hand, the *round&assim* unit takes the same $(b+t)$ input word, but produces a two's complement (2C) representation of the coefficient with opposite sign $-e_j$, by rounding the input word and performing its assimilation to non-redundant representation. This $2C$ representation is used for addressing the look-up table storing the logarithm constants and as input for the barrel shifter producing $-e_j r^{-j}$.

3.1. Cycle Time

The cycle time of the sequential architecture is set by the delay of the slowest path in the circuit. Depending on the value of the radix, it can be any of the main paths of the implementation. These paths are shown in Fig. 1:

- Table (re_1) + mux + Mult/Add + *regW*
- Table ($-\ln(1+e_1 r^{-1})$) + mux + SDA3 + *regL*
- shifter + mux + Mult/Add + *regW*
- *round&rec* + SDA4 + mux + Mult/Add (accum. tree) + *regW*
- *round&assim* + Table ($-\ln(1+e_j r^{-j})$) + mux + SDA3 + *regL*

3.2. Hardware Requirements

The hardware requirements for this architecture consist of the table storing the digits e_1 ($2^{b+1} \times \beta$ bits, with $\beta = 3\lceil (b+1)/2 \rceil$), the table storing their logarithm ($2^{b+1} \times (n+g)$ bits), the table storing the logarithm constants for the next $N_1 = \lfloor \frac{N-1}{2} \rfloor$ iterations ($(2^{b+1+i} \times (n+g)$ bits, with $i = \lceil \log_2(N_1) \rceil$), one $(b+1) \times (n+g) + (n+g)$ bit SD multiply-add unit, one b-bit SDA4 adder, one $(n+g)$-bit SDA3 adder, one $(n+g)$-bit shifter for the $N-1$ iterations when $W[j]r^{-j+1}$ is needed, one $(b+1)$-bit shifter for the $\lceil \frac{N-1}{2} \rceil$ iterations when the approximation $-e_j r^{-j}$ to the logarithm is required, the registers $regW$ and $regL$, the *round&rec* and *round&assim* units, the *on-the-fly* conversion unit and several multiplexers.

4. Evaluation and Analysis

In this section we present estimates of the execution time and the area costs of the architecture proposed in the previous section, for precisions of $n = 16, 32$ and 64 bits (of interest in DSP applications), and $n = 24$ and 53 bits (for conventional floating-point operations), for radix values from $r = 8$ to $r = 1024$. These estimates are based on an approximate model for the cost and delay of the main logic blocks used.

4.1. Delay and Area Model for the Main Logic Blocks

The actual delays and area costs depend on the technology used and on the actual implementation. However, this model provides a good first–order approximation to the actual execution time and area values, and has been widely used in technology-independent comparisons [4, 32, 33]. The units employed are the delay τ and the area fa of a full-adder. More details about the model used can be found in [20].

4.1.1. Delay Estimates. The following assumptions have been made:

- *SD Multipliers.* The delay on their critical path is composed of two terms instead of three, since no final assimilation to non-redundant representation is performed. The first term corresponds only to the multiple generation and buffering (1τ), since the multiplier operand is already in SD radix-4 representation (the recoding is performed in the *round&rec* unit, and the digits e_1 are stored in the look-up table already in SD-4 form). The second term comes from the partial products accumulation. We assume a reduction tree to be used as the multioperand adder,

composed of SDA4 and SDA3 adders, with delays of 1.5τ and 1τ, respectively.
- The delay of the shifters depends on the number of possible shifts of the input vectors. For up to 4 shifts, the delay will be 0.5τ (one level of 4:1 multiplexers). Between 5 and 16 shifts, the delay will be 1.5τ (two levels of 4:1 multiplexers, plus buffering and control overhead), and for combinations between 17 and 64, we assume a delay of 3τ. The delay of the *round&rec* unit is 2τ (1.5τ from the rounding and recoding scheme and 0.5τ from the initial bit-inversion of the truncated residual $\hat{W}[j]$, necessary to produce the digit e_j with opposite sign). The *round&assim* unit has a delay of 2τ for an input up to 8 bits, and 3τ if the input has up to 16 bits. The 2:1 and 3:1 multiplexers have a delay of 0.5τ, and the delay of the registers is about 1τ.
- *Look-up tables*. According to [19, 32] and to our own estimates obtained from implementation [4, 22], a delay of about 3τ for 7 input bit tables is assumed, 3.5τ for 8 input bit tables, 4τ for 9 input bit tables, 4.5τ for 10-11 input bit tables, 5τ for 12-13 input bit tables and 6τ for 14-15 input bit tables.

4.1.2. Area Estimates. The model we use for the area estimates is taken from [33]. The main contributions to the area of the architecture come from the SD multiplier, SDA adders, shifters, registers, multiplexers and look-up tables. The area of the *round&rec* and *round&assim* units is small when compared to the area of the main blocks and can be neglected in this approximate model.

- *SD Multipliers and SD Adders.* As an example of the model used, let's consider a 8×56 bit multiplier, with an area estimate of $636\,fa$. There are two stages in the multiplier, with a different contribution ($300\,fa + 336\,fa$) to the total area:
  - In the initial SD radix-4 recoding and multiple generation, 6 *nand/nor* gates are required for each 3–bit recoding group. Since 8 partial products have to be generated (4 partial products for each SD word of the multiplicand operand), with a wordlength of 56-bits each, the total number of *nand/nor* gates used is about 2688. A standard 1–bit full–adder has a hardware complexity equivalent to 9 *nand/nor* gates. Therefore, the first stage of the multiplier can be estimated as having a total area of about $300\,fa$.
  - The accumulation tree is composed of 2 levels ($8 \rightarrow 4 \rightarrow 2$ operands), employing three SDA4 adders and no SDA3 adders. Considering a wordlength of 56 bits for each of these adders, and since a n-bit SDA3 adder has an area of $n\ fa$, while a n-bit SDA4 is composed of $2n\ fa$, the total area of this stage is $6 \times 56 = 336\,fa$.
- The shifters have an area of $0.5 \times n$ for n-bit vectors up to four possible shifts, $0.9 \times \log_2(j) \times n$ if the number j of possible shifts is between 5 and 16, and $2.1 \times \log_2(j) \times n$ if j goes from 17 to 64. The contribution of the shifters to the area must be duplicated when considering redundant operands. The area of the multiplexers is about $0.25 \times k \times n$, with k the number of input vectors and n their wordlength. The area of a n-bit register can be estimated as $0.5 \times n\ fa$, and an *on-the-fly* conversion and rounding unit requires the use of 9 n-bit registers [15].
- *Look-up tables*. Estimates for look-up tables can be found in [19], but according to our own estimates [4, 22] they seem too pessimistic. Our model assumes a 40 fa/Kbit rate for tables addressed by up to 6-bit words, a 35 *fa*/Kbit rate for 7–11 input bit tables with the delays described above, a 30 *fa*/Kbit rate for 12–13 input bit tables and a 25 *fa*/Kbit rate for 14–15 input bit tables.

4.2. Analysis of the Area and Speed Tradeoffs

Tables 1 to 5 show the latency, cycle time, execution time, combinational area, look-up table area and total area for the proposed architecture, for precisions $n = 16, 24, 32, 53$ and 64-bits, respectively, and for radix values from $r = 8$ to $r = 1024$. The latency is given in cycles, the cycle time and execution time are expressed in terms of τ, and the area unit is fa. The dependence of both the execution times and the total areas with the value of the radix r is graphically represented in Fig. 2 for all considered precisions.

The analysis of the obtained results can be summarized in the following points:

- The main trends in both the execution time and the total area are consistant for all analyzed precisions.
- The cycle time mainly depends on the radix, and not on target precision, due to the use of redundant arithmetic.
- The main contribution to the total area for the proposed architecture comes from the combinational

Table 1. Execution time and total area for the proposed architecture ($n = 16$).

Radix	Latency	Cycle time (τ)	Exec. time (τ)	Comb. area (fa)	Table area (fa)	Total area (fa)
8	6	7.0	42.0	407	40	447
16	**4**	**7.5**	**30.0**	**425**	**54**	**479**
32	4	7.5	30.0	430	116	546
64	**3**	**9.5**	**28.5**	**493**	**219**	**702**
128	3	10.0	30.0	497	437	934
256	**2**	**10.5**	**21.0**	**528**	**542**	**1070**
512	2	11.0	22.0	530	1155	1685
1024	2	11.5	23.0	594	2380	2974

Table 2. Execution time and total area for the proposed architecture ($n = 24$).

Radix	Latency	Cycle time (τ)	Exec. time (τ)	Comb. area (fa)	Table area (fa)	Total area (fa)
8	8	7.0	56.0	557	90	647
16	6	7.5	45.0	653	113	766
32	**5**	**7.5**	**37.5**	**616**	**214**	**830**
64	4	9.5	38.0	688	279	967
128	4	10.0	40.0	693	577	1270
256	**3**	**10.5**	**31.5**	**782**	**1171**	**1953**
512	3	11.0	33.0	784	2415	3199
1024	3	11.5	34.5	878	4900	5778

Table 3. Execution time and total area for the proposed architecture ($n = 32$).

Radix	Latency	Cycle time (τ)	Exec. time (τ)	Comb. area (fa)	Table area (fa)	Total area (fa)
8	11	7.5	82.5	728	187	915
16	8	7.5	60.0	831	210	1041
32	**7**	**8.0**	**56.0**	**836**	**427**	**1263**
64	6	9.5	57.0	958	515	1473
128	**5**	**10.0**	**50.0**	**909**	**1050**	**1959**
256	**4**	**10.5**	**42.0**	**1003**	**1451**	**2454**
512	4	11.0	44.0	1005	2975	3980
1024	4	11.5	46.0	1127	6021	7147

area for small radices, but the tables become the primary factor for radix $r = 256$ when $n = 16, 24$, for $r = 128$ when $n = 32, 53$, and for $r = 64$ when $n = 64$-bits of precision. This is due to the exponential growth in the table size with the radix.

- The cycle time increases with the radix, but the latency decreases. A good tradeoff between latency and cycle time can be achieved for specific values of r, leading to low execution times. The area requirements must also be taken into account when trying to determine the most efficient implementations. According to Tables from 1 to 5 and to Fig. 2, the best tradeoffs correspond to the radix values highlighted in the tables and the charts. We can conclude that in applications where speed is the main constraint, the most efficient implementations correspond to radix $r = 256$, for $n = 16, 24, 32$ and 64 bits of precision, and radix $r = 128$ for $n = 53$-bit computations. On the other hand, for applications with tighter area constraints, implementations with radix $r = 16$ for $n = 16$, $r = 32$ for $n = 24, 32, 64$, and with $r = 64$ for $n = 53$-bits are probably more efficient.

Table 4. Execution time and total area for the proposed architecture ($n = 53$).

Radix	Latency	Cycle time (τ)	Exec. time (τ)	Comb. area (fa)	Table area (fa)	Total area (fa)
8	18	8.0	144.0	1527	302	1829
16	14	8.0	112.0	1349	597	1946
32	**11**	**8.5**	**93.5**	**1355**	**1211**	**2556**
64	**9**	**9.5**	**85.5**	**1544**	**1333**	**2877**
128	**8**	**10.0**	**80.0**	**1522**	**2642**	**4164**
256	7	10.5	73.5	1717	5302	7019
512	6	11.0	66.0	1719	6615	8334
1024	6	11.5	69.0	1918	12140	14058

Table 5. Execution time and total area for the proposed architecture ($n = 64$).

Radix	Latency	Cycle time (τ)	Exec. time (τ)	Comb. area (fa)	Table area (fa)	Total area (fa)
8	22	8.0	176.0	1777	658	2435
16	16	8.0	128.0	1566	697	2263
32	**13**	**8.5**	**110.5**	**1572**	**1401**	**2973**
64	11	9.5	104.5	1806	2759	4565
128	**10**	**10.0**	**100.0**	**1812**	**3123**	**4935**
256	**8**	**10.5**	**84.0**	**2108**	**6177**	**8185**
512	8	11.0	88.0	2110	11065	13075
1024	7	11.5	80.5	2242	22210	24442

The main conclusion that can be drawn from this analysis is that little advantage, or no advantage at all, is obtained from using very-high radix values such as $r = 512$ and $r = 1024$, because the execution times are similar to those achieved with $r = 128$ or $r = 256$, but the area requirements become prohibitive.

5. Comparison

In this section we perform a comparison of the proposed architecture with a conventional radix-4 digit-recurrence implementation for a precision of $n = 32$ bits, using redundant arithmetic and assuming the same model for the delay and area used to obtain the estimates for our architecture. We also compare the execution time and area estimates of our optimized architecture with the architecture proposed in [21, 22].

The radix-4 recurrence for computing the logarithm is:

$$\begin{aligned} W[j+1] &= 4W[j] + 4e_j + e_j W[j]4^{-j+1} \\ L[j+1] &= L[j] - \ln(1 + e_j 4^{-j}) \end{aligned} \tag{36}$$

with $j \geq 1$, $e_j = \{-2, -1, 0, +1, +2\}$, $W[1] = 4(X - 1)$ and $L[1] = Y$.

The selection of the digits e_j is performed by comparing the truncated residual $\hat{W}[j]$ with a set of constants. The employed digit set allows the computation of the iterations without performing any multiplication (a conventional *shift-and-add* algorithm). The block diagram of the radix-4 digit-recurrence architecture is shown in Fig. 3.

A look-up table of size $2^{s+1} \times (n + g)$ bits is used for storing the elementary values of the logarithms $-\ln(1 + e_j 4^{-j})$, with $s = \lceil \log_2(k) \rceil$ and $k = \lfloor \frac{n}{4} \rfloor$ the number of iterations when the value of the logarithm cannot be approximated by the shifted coefficient $-e_j 4^{-j}$. The *cond_inv* unit carries out the computation of the product $e_j W[j]4^{-j+1}$.

The cycle time of the radix-4 architecture is 6.5τ, as shown in Fig. 4. Since 2 bits of the result are extracted per iteration, and assuming the use of $g = 5$ guard bits, the latency of a standard sequential architecture for $n = 32$-bits of precision is 16 cycles. Therefore, the execution time can be estimated as 104τ.

The hardware requirements for this radix-4 architecture are also detailed in Fig. 4. The estimate for the total area is around $843 fa$, with important contributions from the selection table, the on-the-fly conversion and rounding unit [27, 28], and the shifter, due to the

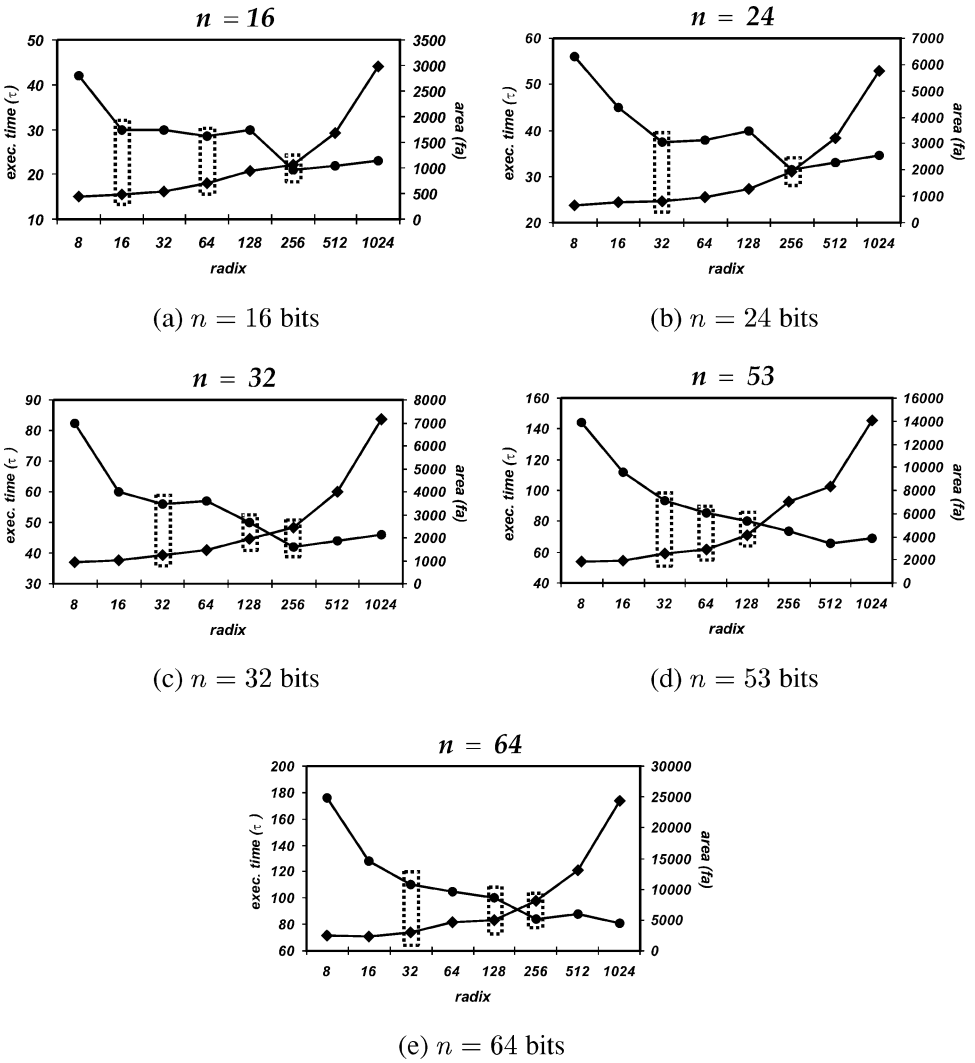

Figure 2. Execution time and area cost for the proposed architecture.

high number of iterations required and to the fact that the input operand to this unit, $W[j]$, is in redundant representation.

A speed-up over 2 can be achieved by using our high-radix algorithm, as shown in Table 6, which means that the execution time of the radix-4 redundant implementation is reduced by half. We also show in this table that the optimizations made in the architecture allow significant reductions in the cycle time and therefore the execution time regarding the architecture proposed in [21, 22], leading to higher speed-ups. We can conclude that our high-radix scheme is an interesting alternative for applications with speed requirements that cannot be met by traditional low-radix implementations, although area restrictions may apply, and therefore the actual value of the radix to be used must be carefully selected taking into account the analysis of the tradeoff between area and speed presented.

The conclusions drawn from the analysis performed can possibly be extended to other high-radix algorithms with similar features (three or more look-up tables addressed by $(b + 1)$ bits and a rectangular $(b + 1) \times (n + g)$-bit multiplier). Examples of these type of algorithms are the CORDIC implementations proposed in [3, 18].

We have estimated [20] that a radix-128 implementation could lead to an area reduction by a factor of

Table 6. Comparison with a radix-4 redundant architecture for $n = 32$ bits.

Scheme	Latency	Cycle time (τ)	Exec. time (τ)	Area (fa)	Speed-up	Area ratio
radix-4 red	16	6.5	104	843	1.0	1.0
radix-32 [22]	7	9.0	63	1182*	1.7	1.4*
radix-128 [22]	5	11.0	55	1827*	1.9	2.2*
radix-256 [22]	4	12.5	50	2284*	2.1	2.7*
radix-32	7	8.0	56	1263	1.9	1.5
radix-128	5	10.0	50	1959	2.1	2.3
radix-256	4	10.5	42	2454	2.5	2.9

**on-the-fly* conversion not included.

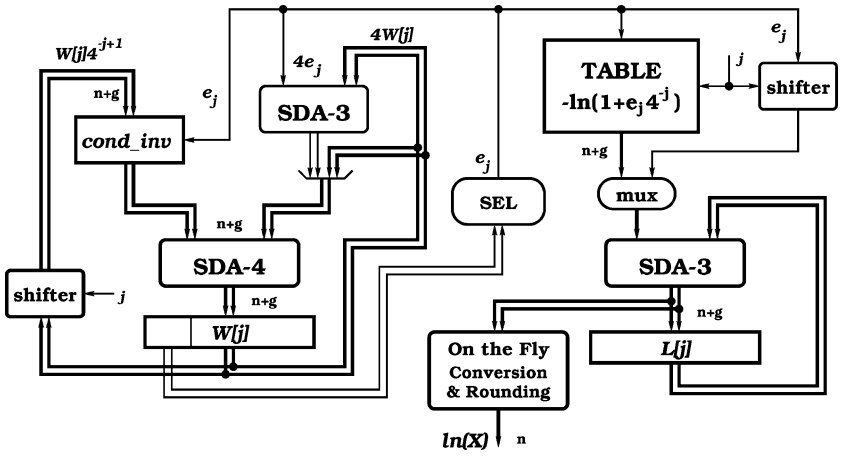

Figure 3. Block diagram of a sequential radix-4 architecture.

Delay estimates

path_A: 1.5τ (shifter) + 1τ (*cond_inv*) + 1.5τ (SDA4) + 1τ (regW) = 5τ
path_B: 1.5τ (SEL) +1τ (SDA3) + 1.5τ (SDA4) + 1τ (regW) = 5τ
path_C: 1.5τ (SEL) + 2.5τ ($TAB\_ - \ln\_e_j$) + 0.5τ (mux) + 1τ (SDA3) + 1τ (regL) = 6.5τ **(critical path)**
execution time: latency × cycle time = $16 \times 6.5\tau = \mathbf{104}\tau$

Area estimates

Look-Up Table ($(2^{5+1} \times 37)$) + (37+2)-bit SDA3 + 37-bit SDA4 + SEL + 4x 37-bit reg + red. shifter + 37-bit 2:1 mux + *on-the-fly* conv. + *cond_inv* =
$90fa + 39fa + 74fa + 200fa + 74fa + 157fa + 14fa + 167fa + 28fa = \mathbf{843}fa$

Figure 4. Delay and area estimates for a radix-4 logarithm computation ($n = 32$-bits).

3 regarding the radix-512 implementation proposed for the CORDIC vectoring algorithm [18], with similar execution times (120τ and 80τ for modulus and angle computation, respectively, instead of 115τ and 73.5τ).

6. Conclusion

A digit-recurrence algorithm for the computation of the logarithm has been presented in this paper, with the use of high-radix and selection by rounding of

the digits e_j. Selection by table in the first iteration is necessary to guarantee the convergence of the algorithm. The use of high-radix significantly reduces the latency of the algorithm, leading to faster execution times.

The value of the logarithm is obtained through the computation of a multiplicative normalization, by accumulating the values of elementary logarithms stored in a look-up table addressed by the coefficient e_j and the iteration index j. The size of this table can be reduced by using an approximation to the logarithm in iterations $j \geq \lceil \frac{N}{2} \rceil + 1$. Redundant representation is used for all variables in order to reduce the cycle time, making the delay for addition independent of the precision, and an *on-the-fly* unit performs the conversion of the result into non-redundant conventional representation and the final rounding.

A sequential architecture implementing our algorithm has been proposed, and an analysis of the tradeoffs between area and speed in the implementation of this architecture has been performed. Such analysis is based on estimates of the execution time and area obtained for precisions $n = 16, 24, 32, 53$ and 64-bits and for radix values from $r = 8$ to 1024, according to an approximate model for the delay and area of the main logic blocks employed in the architecture.

The main results of our analysis are that: (i) the best tradeoffs are obtained for radix values $r = 16, 32, 64$ for applications with moderate performance and area constraints, (ii) the best tradeoffs are obtained for radix values $r = 128, 256$ for applications with high-speed requirements, and (iii) there is no advantage in using very-high radix values $r = 512$ and $r = 1024$, since they provide similar execution times as those obtained with $r = 128, 256$, while their hardware requirements are significantly higher (due to the exponential growth in the size of the tables with the radix).

A comparison with a redundant radix-4 implementation of the recurrence for computing the logarithm shows a reduction by half in the execution time with our high-radix scheme. Therefore, our algorithm provides a high-speed alternative to traditional digit-recurrence implementations. Moreover, the conclusions drawn from our analysis can also be used to obtain significant improvements in the implementation of similar algorithms, such as high-radix CORDIC.

Acknowledgment

Jose-Alejandro Piñeiro would like to thank Belen Espiña for her support. This work was developed while J.-A. Piñeiro was with the University of California, Los Angeles (UCLA), USA. J.-A. Piñeiro and J. D. Bruguera were partially supported by the Ministry of Science and Technology (MCyT-FEDER) under contract TIC2101-3694-C02.

References

1. M. Arnold, "Reduced Power Consumption in MPEG Decoding using LNS," in *Proc. IEEE 13th Intl. Conference on Application-specific Systems, Architectures and Processors (ASAP02)*, 2002, pp. 65–76.
2. D.M. Lewis, "114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 12, 1995, pp. 1547–1553.
3. E. Antelo, T. Lang, and J.D. Bruguera, "High-Radix CORDIC Rotation Based on Selection by Rounding," *Journal of VLSI Signal Processing*, vol. 25, no. 2, 2000, pp. 141–153.
4. J.-A. Piñeiro and J.D. Bruguera, "High-Speed Double-Precision Computation of Reciprocal, Division, Square Root and Inverse Square Root," *IEEE Transactions on Computers*, vol. 51, no. 12, 2002, pp. 1377–1388.
5. H.C. Shin, J.A. Lee, and L.S. Kim, "A Minimized Hardware Architecture of Fast Phong Shader using Taylor Series Approximation in 3D Graphics," in *Proc. International Conference on Computer Design, VLSI in Computers and Processors*, 1998, pp. 286–291.
6. W. Cody and W. Waite, *Software Manual for the Elementary Functions*, Prentice-Hall, 1980.
7. P. Markstein, *IA-64 and Elementary Functions*, Hewlett-Packard Professional Books, 2000.
8. P.T.P. Tang, "Table Look-up Algorithms for Elementary Functions and Their Error Analysis," in *Proc. IEEE 10th Int. Symp. Computer Arithmetic (ARITH10)*, 1991, pp. 232–236.
9. C.T. Fike, *Computer Evaluation of Mathematical Functions*, Prentice-Hall, 1968.
10. J.M. Muller, *Elementary Functions, Algorithms and Implementation*, Birkhauser, 1997.
11. M.J. Schulte and J.E. Stine, "Symmetric Bipartite Tables for Accurate Function Approximation," in *Proc. 13th Symp. Computer Arithmetic (ARITH13)*, 1997, pp. 175–183.
12. M.J. Flynn, "On Division By Functional Iteration," *IEEE Transactions on Computers*, vol. 19, 1970, pp. 702–706.
13. S.F. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor," in *Proc. 14th Symp. Computer Arithmetic (ARITH14)*, 1999, pp. 106–115.
14. M.D. Ercegovac, "Radix-16 Evaluation of Certain Elementary Functions," *IEEE Transactions on Computers*, vol. 22, no. 6, 1973, pp. 561–566.

15. M.D. Ercegovac and T. Lang, *Division and Square Root: Digit Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.
16. M.D. Ercegovac, T. Lang, and P. Montuschi, "Very High-Radix Division with Selection by Rounding and Prescaling," *IEEE Transactions on Computers*, vol. 43, no. 8, 1994, pp. 909–918.
17. E. Antelo, T. Lang, and J. D. Bruguera, "Computation of $\sqrt{x/d}$ in a Very-High Radix Combined Division/Square-Root Unit with Scaling and Selection by Rounding," *IEEE Transactions on Computers*, vol. 47, no. 2, 1998, pp. 152–161.
18. E. Antelo, T. Lang, and J.D. Bruguera, "Very-High Radix CORDIC Vectoring with Scalings and Selection by Rounding," in *Proc. 14th Symp. Computer Arithmetic*, 1999, pp. 204–213.
19. T. Lang and P. Montuschi, "Very-High Radix Square Root with Prescaling and Rounding and a Combined Division/Square Root Unit," *IEEE Transactions on Computers*, 1999, pp. 827–841.
20. J.-A. Piñeiro, "Algorithms and Architectures for Elementary Function Computation," PhD Dissertation, University of Santiago de Compostela, 2003.
21. J.-A. Piñeiro, M.D. Ercegovac, and J.D. Bruguera, "High-Radix Logarithm with Selection by Rounding," in *Proc. IEEE 13th Intl. Conference on Application-specific Systems, Architectures and Processors (ASAP02)*, 2002, pp. 101–110.
22. J.-A. Piñeiro, M.D. Ercegovac, and J.D. Bruguera, "Analysis of the Tradeoffs in the Implementation of a High-Radix Logarithm," in *Proc. 2002 IEEE Intl. Conference on Computer Design (ICCD02)*, 2002, pp. 132–137.
23. P.W. Baker, "Parallel Multiplicative Algorithms for Some Elementary Functions," *IEEE Transactions on Computers*, 1975, pp. 322–325.
24. R.W. Bemer, "A Subroutine Method for Calculating Logarithms," *CACM*, vol. 1, 1958, pp. 5–7.
25. J.N. Mitchell, "Computer Multiplication and Division using Binary Logarithms," *IEEE Transactions on Electronic Computers*, 1962, pp. 512–517.
26. W.H. Specker, "A Class of Algorithms for $\mathrm{Ln}(x)$, $\mathrm{Exp}(x)$, $\mathrm{Sin}(x)$, $\mathrm{Cos}(x)$, $\mathrm{Tan}(x)$ and $\mathrm{Cot}(x)$," *IEEE Transactions Electronic Computers*, vol. 14, 1965, pp. 85–86.
27. M.D. Ercegovac and T. Lang, "On-the-fly Conversion of Redundant into Conventional Representations," *IEEE Transactions on Computers*, vol. C-36, no. 7, 1987, pp. 895–897.
28. M.D. Ercegovac and T. Lang, "On-the-fly Rounding," *IEEE Transactions on Computers*, vol. 41, no. 12, 1992, pp. 1497–1503.
29. Waterloo Maple Inc, *Maple 8 Programming Guide*, 2002.
30. T.C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios and Square-Roots," *IBM Journal Research and Development*, 1972, pp. 380–388.
31. P. Kornerup, "Reviewing 4-to-2 Adders for Multi-Operand Addition," in *Proc. IEEE 13th Intl. Conference on Application-specific Systems, Architectures and Processors (ASAP02)*, 2002, pp. 218–229.
32. M.D. Ercegovac, T. Lang, J.M. Muller, and A. Tisserand, "Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers," *IEEE Transactions on Computers*, vol. 49, no. 7, 2000, pp. 628–637.
33. W.F. Wong and E. Goto, "Fast Hardware-Based Algorithms for Elementary Function Computations," *IEEE Transactions on Computers*, vol. 43, no. 3, 1994, pp. 278–294.

Jose-Alejandro Piñeiro was born in Domayo, Spain. He received the Ph.D. degree in Computer Engineering in 2003, and the M.Sc. degree (1999) and B.Sc. degree (1998) in Physics (Electronics), from the University of Santiago de Compostela, Spain. Since 2004, he has been with Intel Barcelona Research Center, Intel Labs-UPC, whose research focuses on new microarchitectural paradigms and code generation techniques for IA-32, EM64T and IPF families. His research interests are also in the area of computer arithmetic, VLSI design, computer graphics and numerical processors.
alex@dec.usc.es

Miloš D. Ercegovac is a Professor and Chair in the UCLA Computer Science Department. He earned his MS ('72) and Ph.D. ('75) in computer science from the University of Illinois, Urbana-Champaign, and BS in electrical engineering ('65) from the University of Belgrade, Yugoslavia. Dr. Ercegovac specializes in research and teaching in digital arithmetic, digital design, and computer system architecture. His research contributions have been extensively published in journals and conference proceedings. He is a coauthor of two textbooks on digital design and of a monograph in the area of digital arithmetic. Dr. Ercegovac has been involved in organizing the IEEE Symposia on Computer Arithmetic since 1978. He served as an editor of the IEEE Transactions on Computers and as a subject area editor for the *Journal of Parallel and Distributed Computing*. Dr. Ercegovac is a senior member of the IEEE Computer Society and a member of the ACM.
milos@cs.ucla.edu

Javier D. Bruguera received the B.S. degree in Physics and the Ph.D. degree from the University of Santiago de Compostela (Spain) in 1984 and 1989, respectively. Currently, he is a professor in the Department of Electronic and Computer Engineering at the University of Santiago de Compostela. Previously, he was an assistant professor in the Department of Electrical, Electronic and Computer Engineering at the University of Oviedo, Spain, and an assistant professor in the Department of Electronic Engineering at the University of A Coruña, Spain. He was a visiting researcher in the Application Center of Microelectronics at Siemens in Munich, Germany, and in the Department of Electrical Engineering and Computer Science at the University of California, Irvine. Dr. Bruguera's primary research interests are in the area of computer arithmetic, processor design, digital design for signal and image processing and parallel architectures.
bruguera@dec.usc.es