High–Radix Iterative Algorithm for Powering Computation

J.-A. Piñeiro[§], M. D. Ercegovac[¶], J. D. Bruguera[§]

[§] Department of Electronic and Computer Engineering
Univ. Santiago de Compostela, Spain.

[¶] Computer Science Dept.
University of California, Los Angeles (UCLA), USA.
e-mail: alex,bruguera@dec.usc.es, milos@cs.ucla.edu

Abstract

A high-radix composite algorithm for the computation of the powering function (X^Y) is presented in this paper. The algorithm consists of a sequence of overlapped operations: (i) digit-recurrence logarithm, (ii) left-to-right carry-free (LRCF) multiplications, and (iii) on-line exponential. A redundant number system is used, and the selection in (i) and (iii) is done by rounding except from the first iteration, when selection by table look-up is necessary to guarantee the convergence of the recurrences. A sequential implementation of the algorithm is proposed, and the execution times and hardware requirements are estimated for single and double-precision floating-point computations, for radix $r = 128$, showing that powering can be computed with similar performance as high-radix CORDIC algorithms.

1 Introduction

Powering (X^Y) is an important operation in applications such as scientific computing, digital signal processing (DSP) and computer 3D graphics [7]. As other elementary functions, such as square root, inverse square root, logarithm and trigonometric functions, it has been traditionally computed by software routines [2, 6]. These routines provide very accurate results, but they are often too slow for numerically intensive or real-time applications. The timing constraints of these applications have led to the development of dedicated hardware for the computation of elementary functions, including the implementation of table-based algorithms [9, 15], functional iteration methods [10, 11] and digit-recurrence algorithms [1, 4].

However, accurately computing the floating-point powering function is considered difficult [9], and the prohibitive hardware requirements of a table-based implementation

(note that X^Y is a 2-variable function) have led only to partial solutions, such as powering algorithms for a constant exponent [12, 16] or for very low precision [7]. A direct implementation of a digit-recurrence algorithm for powering computation is not feasible due to its high intrinsic complexity.

In this paper we present a composite iterative algorithm for the computation of the powering function (X^Y), for a floating-point input operand $X = M_x 2^{E_x}$ and integer b -bit operand¹ Y . The final result X^Y is computed as $Z = M_z 2^{E_z} = e^{Y \ln(M_x)} 2^{Y E_x}$, through a sequence of overlapped operations. The first step consists of computing $\ln(M_x)$ by using a high-radix recurrence with selection by rounding [13]. An intermediate computation $Y_L \ln(M_x)$, with $Y_L = Y \log_2(e)$, is carried out using a high-radix left-to-right carry-free (LRCF) multiplication [3]. Another LRCF multiplication by $\ln 2$ is performed to guarantee the convergence of the algorithm and, as last step, the exponential of the resulting product is computed by an on-line high-radix ($r = 2^b$) algorithm, with on-line delay $\delta = 2$ [14].

In the stages computing the logarithm and the exponential, selection by table look-up is performed in the first iteration to guarantee the convergence of both algorithms. However, the on-line delay of 2 cycles allows the addressing of the initial tables in the exponential scheme one cycle in advance, reducing the delay of the critical path in this stage.

A sequential architecture implementing our algorithm with radix $r = 128$ is proposed in this paper, and estimates of the total area and execution time for single and double-precision floating-point computations are obtained according to an approximate model for the area and delay of the main components used in the proposed architecture.

¹The computation of the powering operation with integer exponent in the range $[1, 128]$ is useful for instance in computer graphics computations (the lighting specular component in the OpenGL graphics pipeline [7]).

Figure 1. Operation flow of the powering algorithm (single-precision computation, $r = 128$)

2 Algorithm

In this section we present a new composite iterative algorithm for the computation of the powering function (X^Y), for a floating point operand X and an integer exponent Y , describing the algorithm and its error analysis.

2.1 Overview

The algorithm is based on the well-known identity

$$X^Y = e^{Y \ln(X)}$$

Considering a floating-point² input operand $X = M_x 2^{E_x}$, with M_x the n -bit significand and E_x the exponent, and an integer b -bit input operand Y :

$$\begin{aligned} X^Y &= e^{Y \ln(M_x 2^{E_x})} = e^{Y(\ln(M_x) + \ln(2^{E_x}))} \\ &= e^{Y \ln(M_x)} e^{Y \ln(2^{E_x})} = e^{Y \ln(M_x)} e^{Y E_x \ln 2} \\ &= e^{Y \ln(M_x)} 2^{Y E_x} \end{aligned} \quad (1)$$

According to (1), the powering function can be calculated by a sequence of operations consisting of the logarithm of the significand M_x , a multiplication by Y and the exponential of the resulting product. The form of the result is a significand $e^{Y \ln(M_x)}$ and an exponent $Y E_x$, typical of a floating-point operand³.

For an efficient implementation of the powering function, the computation of the operations involved must be overlapped, which requires a left-to-right most-significant digit first (MSDF) mode of operation and the use of a redundant number system.

A problem of the proposed algorithm is the range of the digit-recurrence exponential, which is $(-\ln 2, \ln 2)$, while

²The algorithm can be adapted to a normalized fixed-point input operand $X = M_x$, $1 \leq M_x < 2$, resulting in $X^Y = e^{Y \ln(M_x)}$

³For fixed-point computations, the value $e^{Y \ln(M_x)}$ can be shifted $Y E_x$ positions.

the argument of the exponential here is $Y \ln(M_x)$, being Y an integer. To extend the range of convergence of this algorithm, and thus guarantee the convergence of the overall proposed method, we use the following mathematical identity [8]:

$$e^\beta = e^{\beta \log_2(e) \ln 2}$$

If $\beta \log_2(e)$ is computed, and its integer I and fractional part F are extracted, e^β becomes:

$$\begin{aligned} e^\beta &= e^{(I+F) \ln 2} = e^{I \ln 2 + F \ln 2} \\ &= e^{\ln(2^I)} e^{F \ln 2} = 2^I e^{F \ln 2} \end{aligned}$$

Since in this case $\beta = Y \ln(M_x)$:

$$e^{Y \ln(M_x)} = 2^I e^{F \ln 2},$$

and therefore, according to (1),

$$X^Y = e^{F \ln 2} 2^{(I+Y E_x)}, \quad (2)$$

with I and F the integer and the fractional part of $Y \ln(M_x) \log_2(e)$, resulting in a bounded argument for the exponential. The product $Y_L = Y \log_2(e)$ can be computed by using a $(b \times n)$ -bit multiplier in parallel with the computation of the first iteration of the logarithm.

In summary, as illustrated in Figure 1 for single-precision computations with $r = 128$, our algorithm for the computation of the powering function consists of the following steps:

1. Computing the logarithm of the input significand, $\ln(M_x)$, by employing a high-radix digit-recurrence algorithm [13].
2. Computing the product $Y_L \ln(M_x)$ by using a a serial left-to-right carry-free (LRCF) multiplication scheme [3], with $Y_L = Y \log_2(e)$ (obtained in parallel with the first iteration of the logarithm). The integer I and the fractional F parts of $Y_L \ln(M_x)$ are extracted serially.

Precision	N_e	N_l	n_e	n_l	n_{m0}	n_{m1}	n_{m2}
SP ($n = 24$)	4	5	27	35	27	33	34
DP ($n = 53$)	9	10	57	65	57	64	64

Table 1. Parameters for powering computation ($r = 128$)

3. Computing the product $H = F \ln 2$ by using a LRCF multiplication scheme [3].
4. Computing the exponential $M_z = e^{F \ln 2}$, by employing an on-line high-radix algorithm [14].

The exponent E_z can be computed in parallel by an integer multiply/add unit. The output of the algorithm is the floating-point normalized result:

$$X^Y = M_z 2^{E_z} = e^{F \ln 2} 2^{(I+Y E_z)} \quad (3)$$

The use of redundancy results in $e^{F \ln 2} \in (0.5, 2)$, and therefore a normalization of the final result may be necessary. However, the condition $F \ln 2 < 0$ can be determined in advance to the last iterations of the exponential, and the final normalization can be performed with no extra delay.

The overall latency of the algorithm, as shown in Figure 1, can be estimated as

$$\text{latency} = N_e + (\delta + 1) + 3, \quad (4)$$

with N_e the number of iterations of the on-line exponential, $(\delta + 1)$ cycles to accommodate the on-line delay, 2 cycles due to the two LRCF multiplications, and an extra cycle due to the computation of the integer part I of $Y_L \ln(M_x)$. I can be obtained in a single cycle because it is guaranteed to have b bits, since Y is a b -bit integer and $r = 2^b$.

2.2 Error analysis

The final error in the algorithm consists of the accumulation of the errors due to the cascaded implementation of a set of operations, and must be bounded by 2^{-n} .

Let ϵ_l be the error in the computation of the logarithm of the input significand $(\ln(M_x) + \epsilon_l)$, and ϵ_{m0} the error due to the finite arithmetic in the computation of the product Y_L ($Y \log_2(e) + \epsilon_{m0}$). When the first LRCF multiplication is performed, the associated error is:

$$\epsilon_{LRCF1} = \epsilon_l Y \log_2(e) + \epsilon_{m0} \ln(M_x) + \epsilon_l \epsilon_{m0} + \epsilon_{m1},$$

with ϵ_{m1} the error due to the LRCF multiplication scheme in the computation of the product.

The integer I and fractional F parts of the product $Y \log_2(e) \ln(M_x)$ are extracted serially, with no error affecting the integer part. The next operation to be performed

Figure 2. Block diagram of the proposed architecture

is the computation of the product $H = F \ln 2$. The error associated with this computation is:

$$\epsilon_H = \epsilon_{LRCF1} \ln 2 + \epsilon_{m2},$$

with ϵ_{m2} the error due to the LRCF multiplication scheme in the computation of the product.

Finally, the exponential e^H is computed, and therefore the obtained result is:

$$e^H e^{\epsilon_{LRCF1} \ln 2 + \epsilon_{m2}} + \epsilon_e,$$

with ϵ_e the error in the computation of the exponential.

The difference between the exact result and the obtained result must be bounded:

$$|e^H - e^H e^{\epsilon_{LRCF1} \ln 2 + \epsilon_{m2}} - \epsilon_e| < 2^{-n}$$

Taking into account that $e^H \in (0.5, 2)$:

$$|1 - e^{(\epsilon_l Y \log_2(e) + \epsilon_{m0} \ln(M_x) + \epsilon_l \epsilon_{m0} + \epsilon_{m1}) \ln 2 + \epsilon_{m2}} - \epsilon_e| < 2^{-n-1}$$

For a precision of n bits and a radix $r = 2^b$, a set of minimum values for ϵ_l , ϵ_{m0} , ϵ_{m1} , ϵ_{m2} and ϵ_e must be determined to guarantee a final result accurate to n bits. The values of the error parameters set the precision to be reached in each stage, n_l , n_e , n_{m1} , n_{m2} , and the accuracy n_{m0} of the product $Y \log_2(e)$. These parameters determine a minimum number of iterations of the logarithm (N_l) and the exponential (N_e) to be performed. As shown in Figure 1, the number of iterations of the LRCF multiplications to be performed is the same as those of the logarithm, N_l , because all the information must reach the exponential stage. The parameters N_l , N_e , n_l , n_e , n_{m1} , n_{m2} and n_{m0} set the size of the look-up tables, adders and multipliers to be used.

Figure 3. Block diagram of the high-radix logarithm stage

Moreover, g_l , g_e , g_{m1} and g_{m2} guard bits must be employed to guarantee that in each stage of the powering algorithm the iteration errors do not affect the achievement of the required precisions n_l , n_e , n_{m1} and n_{m2} .

The critical parameter to be first minimized is ϵ_e , since it is directly related to the required precision n_e to be reached in the exponential stage, and therefore to N_e , the number of iterations of the on-line exponential to be performed⁴.

For radix $r = 128$, the set of minimum values for the considered parameters is shown in Table 1 for single-precision ($n = 24$) and double-precision ($n = 53$) computations. N_e and N_l are given in cycles, and n_e , n_l , n_{m0} , n_{m1} and n_{m2} are given in number of bits.

3 Implementation

In this section a sequential architecture is proposed for the implementation of our high-radix powering algorithm. We give a detailed explanation of the main computations involved: (i) high-radix logarithm, (ii) high-radix LRCF multiplication, and (iii) on-line high-radix exponential, and then outline the main features of the logic blocks employed. Figure 2 shows the block diagram of the proposed architecture. Single thick lines denote long-word (around n bits) operands/variables in parallel form, single thin lines denote short-word (up to 11 bits) operands/variables in parallel form, and double lines denote single-digit (b bits) variables (R_j , I , F_j and H_j).

⁴Note that N_e is the only parameter that affects the latency of the algorithm, since $\delta = 2$ for any n and any $r \geq 8$, as shown in [14].

3.1 High-Radix Logarithm

A high-radix digit-recurrence logarithm is described in detail in [13], although in the algorithm used here some modifications and optimizations have been made according to the operation flow in the powering computation. A slightly different notation from the used in [13] is employed here. The block diagram of the high-radix logarithm stage is shown in Figure 3.

The recurrences for performing the multiplicative normalization of the input operand M_x and computing the logarithm digits are

$$\begin{aligned} W_l[j+1] &= r(W_l[j] + l_j + l_j W_l[j] r^{-j}) \\ R[j+1] &= rR[j] - r^{j-1} \ln(1 + l_j r^{-j}) - rR_j \end{aligned} \quad (5)$$

with $j \geq 1$, $W_l[1] = r(M_x - 1)$, $R[1] = 0$ and $R_1 = 0$. For a result precision of n_l bits, a total number of $N_l = \lceil n_l/b \rceil$ iterations are necessary. The scaled recurrence $R[j]$ has been defined as

$$R[j] = r^{j-2} L[j] \quad (6)$$

in order to extract a radix- r digit R_j per iteration from the same bit-positions in all iterations.

The selection of the digits l_j in iterations $j \geq 2$ is done by rounding an estimate of the residual obtained by truncating $W_l[j]$ to t fractional bits. The selection function is

$$l_j = -\text{round}(\hat{W}_l[j]) \quad (7)$$

The sign of the digit l_j is defined as opposite of the sign of $W_l[j]$ in order to satisfy a bound on the residual, and thus

assuring the convergence. The digit set for the coefficients l_j is $\{-(r-1), \dots, -1, 0, 1, \dots, (r-1)\}$.

Iteration $j = 1$ does not converge with selection by rounding, and therefore the selection of l_1 is performed by table look-up. This table is addressed by the $b + 1$ most significant bits of the input operand M_x , and the selection is done in such a way that the value of $|l_2|$ is bounded according to the convergence conditions [13]. However, this results in an over-redundant digit l_1 ($b + 1$ bits), increasing by one bit the size of the multiplier operand. The convergence conditions also determine a minimum value of $t = 2$ and the radix $r \geq 8$.

A multiply/add unit is used for the computation of the residual recurrence, unlike in the algorithm proposed in [13], where a separated SD multiplier and SDA4 were used. The digit l_1 is stored in the look-up table $TAB(r l_1)$ already in SD-4 recoded form, to reduce the delay of the path containing the table and the multiply/add unit.

The logarithm constants are stored in a look-up table whose size grows exponentially with the radix. However, an approximation $-l_j r^{-1}$ can be used in iterations $j \geq \lceil N_t/2 \rceil + 1$ in order to reduce the overall hardware requirements of the algorithm.

The use of redundant representation is mandatory in our algorithm for powering computation, due to the left-to-right operation flow, and results in faster execution times by making the additions independent of the precision.

3.2 LRCF Multiplication

The left-to-right carry-free (LRCF) multiplication, introduced in [3], produces the product digits from a redundant set in a most-significant-digit-first (MSDF) manner and performs the conversion on-the-fly of the product generated in a redundant form to the conventional form without using a carry-propagate adder and without additional delay. The resulting implementation is fast and regular and is very well suited for VLSI implementations.

We adapt the LRCF multiplication to carry out the intermediate multiplications in the high-radix powering, utilizing its left-to-right operation flow, with redundant representation of the operands and recoding to a high-radix of the multiplier operand. However, since the high-radix exponential is of the on-line type, the digits produced by LRCF multiplier are used without conversion. The block diagram of the two LRCF multiplication stages is shown in Figure 4.

The adapted LRCF algorithm has two operands A and D . A is the radix-2 representation of the multiplicand a , such that $a = \sum_{i=1}^{n_m} A_i 2^{-i}$, with $A_i \in \{0, 1\}$. D the recoded radix- r representation of the multiplier d , such that $d = \sum_{i=1}^{n_m/b} D_i r^{-i}$, with $D_i \in \{-r/2, \dots, r/2\}$ and $r = 2^b$.

The recurrence produces a sequence of two accumulated

(a) LRCF multiplication 1

(b) LRCF multiplication 2

Figure 4. Block diagram of the LRCF multiplication stages

products w and p as follows:

$$\begin{aligned} w[j+1] &= r(\text{fraction}(w[j] + aD_{j+1})) \\ K_{j+1} &= \text{integer}(w[j] + aD_{j+1}) \\ p[j+1] &= p[j] + K_{j+1}r^{-j-1} \end{aligned} \quad (8)$$

with $j = 1, \dots, n_m/b$ and initial values $w[0] = p[0] = 0$. The digits of the multiplier are used from most to least significant, unlike conventional multiplication schemes which use a right-to-left mode. After n_m/b steps, $p[n_m/b]$ is the most significant part of the product while $w[n_m/b]$ is the least significant part.

Figure 5. Block diagram of the on-line high-radix exponential stage.

A fast implementation of the LRCF multiplication scheme requires: (i) use of a redundant adder (either CS or SD) to compute the residual $w[j]$, and (ii) since the maximum value of K_j in (8) is in general larger⁵ than $(r - 1)$, a recoding of K_j and K_{j+1} into P_j in the range $[-(r - 1), (r - 1)]$ is necessary. That is, the resulting digit will be $P_j = f(K_j, K_{j+1})$.

Regarding the implementation proposed in [3], the main modification in the LRCF multiplication stages in the architecture for powering computation is the use of a multiply/add unit for computing the recurrences, instead of separated redundant multiplier and adder.

3.3 On-Line High-Radix Exponential

The on-line high-radix exponential is described in detail in [14], although a slightly different notation is used here. The block diagram of the on-line high-radix exponential stage is shown in Figure 5.

The exponential is computed on the basis of partial information:

$$H[0] = \sum_{j=1}^{\delta} H_j r^{-j}, \quad (9)$$

with δ the on-line delay and H_j the radix- r digits of the input operand H .

The recurrences for performing the additive normaliza-

tion of H and computing the exponential are

$$\begin{aligned} W_e[j+1] &= r(W_e[j] - r^j \ln(1 + e_j r^{-j}) + H_{j+\delta} r^{-\delta}) \\ E[j+1] &= E[j](1 + e_j r^{-j}) \end{aligned} \quad (10)$$

with $j \geq 1$, $W_e[1] = rH[0]$ and $E[1] = 1$. For a result precision of n_e bits, a total number of $N_e = \lceil n_e/b \rceil$ iterations are necessary. An approximation $-e_j r$ to the logarithm constants can be used in iterations $j \geq \lceil N_e/2 \rceil + 1$ reducing the overall hardware requirements of the algorithm.

The selection of digits e_j is done in iterations $j \geq 2$ by rounding an estimate of the residual:

$$e_j = \text{round}(\hat{W}_e[j]), \quad (11)$$

with $e_j \in \{-(r-1), \dots, -1, 0, 1, \dots, (r-1)\}$ and $\hat{W}_e[j]$ obtained by truncating $W_e[j]$ to t fractional bits.

The use of selection by table look-up is required in the first iteration. The table is addressed by the b most significant bits of the input operand H_1 , and the selection is performed so that $-(r-3) \leq e_2 \leq (r-2)$, which results in an overredundant first digit e_1 . The convergence conditions also determine a minimum value of $t = 2$, an on-line delay $\delta = 2$, and a bound on the radix $r \geq 8$.

The look-up tables used in the first iteration are addressed one cycle in advance, since $H[0]$ is already known while $H_{\delta+1}$ is being computed, and the obtained values can be stored in the registers reg_{-e1} and $\text{reg}_{\ln e1}$.

The conversion from redundant to conventional representation of the final result is performed using an on-the-fly method [4].

⁵The range of K_j is $|K_j| < 3r/2$.

Precision	latency	cycle time (τ)	exec. time (τ)	LOG area (fa)	EXP area (fa)	total area (fa)
SP ($n = 24$)	4+6=10	10.0	100.0	1886	1202	4138
DP ($n = 53$)	9+6=15	10.0	150.0	4496	3990	10131

Table 2. Execution time and total area for the proposed architecture ($r = 128$)

3.4 Implementation details

The main features of the proposed architecture are:

- All variables are in redundant representation (signed-digit) to allow faster execution of iterations, since the additions become independent of the precision.
- All products of the type $Q[j]r^{-j}$ and q_jr^{-j} (with $q_j = l_j$ or e_j) are performed as shifts, since $r = 2^b$.
- SDA α is a *signed-digit binary adder* with α input bit-vectors. The addition of two SD operands requires a SDA4 adder, while SDA3 adders can be used for accumulating a SD operand and an operand in two's complement ($2C$) representation.
- An internal recoding from SD radix-2 to SD radix-4 representation of the multiplier operand is performed in the multiply/add units to reduce by half the number of partial products to be accumulated.
- The *round&rec* units take the $(b + t)$ most significant bits of the residuals $W_l[j]$ and $W_e[j]$ ($t = 2$ in both cases) and produce a SD-4 representation of the digits l_j and e_j , to be used as multipliers in the multiply/add units.
- The *round&assim* units take the same $(b + t)$ input words, but produce a two's complement representation of the coefficient with opposite sign ($-l_j$ or $-e_j$), by rounding the input word and performing its assimilation to non-redundant representation. This $2C$ representation is used for addressing the look-up tables storing the logarithm constants and as a input for the barrel shifters producing $-l_jr^{-1}$ or $-e_jr$.

4 Evaluation and comparison

In this section we present estimates of the execution time and the area costs of the proposed architecture, for single and double-precision floating-point computations ($n = 24$ and $n = 53$ bits) and radix $r = 128$. These estimates are based on an approximate model for the cost and delay of the main logic blocks used [5, 11, 13].

The actual delays and area cost depend on the technology used and on the actual implementation. However, this technology-independent model provides a good first-order approximation to the actual execution time and area values.

Table 2 shows the execution time and area cost estimates of the proposed architecture for single and double-precision FP computations, with $r = 128$. The units employed are the delay τ of a complex gate, such a full-adder, and the area of a 1-bit full-adder (fa).

The latency, as explained in (4), is $N_e + 6$ cycles, since $\delta = 2$ in this case, and therefore we have 10 and 15 cycles respectively for single and double-precision⁶. The cycle time of 10.0τ corresponds in both cases to the high-radix logarithm stage, where the critical path is the one composed of the initial table look-up for selecting the digit l_1 , a multiplexer, the multiply/add unit and the register W_l . The total execution times can therefore be estimated as 100τ and 150τ , as shown in Table 2.

The total area of our architecture consists of the hardware requirements of the individual stages (high-radix logarithm, LRCF multiplications and on-line high-radix exponential, shown in Figures 3 to 5), plus the integer $(b \times n_{exp})$ -bit multiply/add unit to compute the exponent E_z , an extra $(b \times n_{m0})$ -bit multiplier required to compute $Y_L = Y \log_2(e)$, the control logic, and a b -bit CPA adder to assimilate H_1 into conventional representation before addressing the initial tables in the on-line high-radix exponential stage.

The total table size is $46Kbits$ ($5.75KB$) and $161Kbits$ ($20KB$), respectively, for single and double-precision computations, with a contribution to the total area, according to the model used, of $1673fa$ and $5660fa$. The total area estimates, as shown in Table 2, are $4138fa$ and $10131fa$.

Comparison with high-radix CORDIC

For the sake of reference, we give now execution time and area estimates for a high-radix vectoring CORDIC implementation [1]. This algorithm has been chosen as a reference because it allows the computation of two-variable functions (the modulus of a 2D vector). To allow for a fair comparison, the same assumptions are made for both algorithms, a radix-128 implementation is considered for the CORDIC⁷, and we consider a fixed-point implementation of our powering algorithm for $n = 32$.

Table 3 shows the latency, cycle time, execution time and total area for our powering algorithm and the radix $r = 128$ CORDIC implementation. When computing the modulus, CORDIC [1] requires 5 microrotations, 2 scaling operations

⁶Note that $N_e = 4$ and 9 for SP and DP, as shown in Table 1.

⁷The algorithm proposed in [1] performs $n = 32$ -bit computations, using radix $r = 512$, with a small radix $R = 32$ only for the first CORDIC microrotation, and two scaling operations to guarantee the convergence.

Scheme	latency	cycle time	exec. time	area(fa)
powering ($r=128$)	11	10.0τ	110τ	5400
CORDIC ($r=128$)	12	10.0τ	120τ	6000

Table 3. Comparison with high-radix CORDIC vectoring algorithm ($n = 32$ bits)

(taking 3 cycles due to the complexity of the second scaling), and 5 extra iterations for compensating the scaling factor. This results in a latency of 12 cycles ($5 + 3 + 4$, since the first extra iteration can be overlapped with the last microrotation). According to the approximate model used, the critical path has a delay of 10τ , and therefore the execution time can be estimated as 120τ . The total area of this implementation is about $6000fa$, with a table size of $142Kb$.

We can conclude that with our algorithm, the powering function can be computed efficiently in dedicated hardware with cost and performance comparable to those of the evaluation of other elementary functions.

5 Summary

A new composite algorithm for the computation of the powering function has been presented in this paper. The algorithm consists of computing X^Y as $e^{Y \ln(M_x)} 2^{YE_x}$, through a sequence of overlapped operations: logarithm, multiplication and exponential.

The computation of the logarithm is carried out by a high-radix digit-recurrence unit, with selection by rounding. The intermediate multiplications are computed by high-radix left-to-right carry-free (LRCF) multipliers. Finally, the exponential is computed by an on-line high-radix unit with selection by rounding. The computation of the exponent of the result is carried out in parallel by an integer multiply/add unit.

A sequential architecture implementing our algorithm has been proposed. Estimates of the execution times and hardware requirements have been obtained, based on an approximate model for the delay and the area of the main components, for single and double-precision floating-point computations with radix $r = 128$.

A comparison with a high-radix CORDIC architecture shows that the execution times and area costs of the unit computing the powering are similar to those of other iterative algorithms for the computation of elementary functions.

Acknowledgement

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 Spanish Ministry of Science and Technology (MCyT-FEDER) under contract TIC2001-3694-C02.

References

- [1] 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*, pages 204–213, April 1999.
- [2] W. Cody and W. Waite. *Software Manual for the Elementary Functions*. Prentice-Hall, 1980.
- [3] M. D. Ercegovac and T. Lang. Fast Multiplication Without Carry Propagate Addition. Technical report, Computer Science Dept., UCLA, 1987.
- [4] M. D. Ercegovac and T. Lang. *Division and Square Root: Digit Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, 1994.
- [5] M. D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand. Reciprocal, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers. *IEEE Trans. on Computers*, 49(7):628–637, 2000.
- [6] J. F. Hart et al. *Computer Approximations*. New York: John Wiley and Sons, 1968.
- [7] D. Harris. A Powering Unit for an OpenGL Lighting Engine. In *Proc. 35th Asilomar Conference on Signals, Systems, and Computers*, pages 1641–1645, 2001.
- [8] E. G. Kogbetliantz. Computation of e^N for $-\infty < n < +\infty$ Using an Electronic Computer. *IBM Journal Res. and Develop.*, pages 110–115, 1957.
- [9] J.-M. Muller. *Elementary Functions. Algorithms and Implementation*. Birkhauser, 1997.
- [10] S. F. Oberman. Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor. In *Proc. 14th Symp. Computer Arithmetic (ARITH14)*, pages 106–115, April 1999.
- [11] 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*, 51(12):1377–1388, 2002.
- [12] J.-A. Piñeiro, J. D. Bruguera, and J.-M. Muller. Faithful Powering Computation using Table Look-up and Fused Accumulation Tree. In *Proc. 15th Intl. Symposium on Computer Arithmetic (ARITH15)*, pages 40–47, 2001.
- [13] J.-A. Piñeiro, M. D. Ercegovac, and J. D. Bruguera. High-Radix Logarithm with Selection by Rounding. In *Proc. IEEE ASAP'02 Conference*, pages 101–110, 2002.
- [14] J.-A. Piñeiro, M. D. Ercegovac, and J. D. Bruguera. On-Line High-Radix Exponential with Selection by Rounding. In *Proc. IEEE ISCAS'2003 Conference*, 2003.
- [15] M. J. Schulte and J. E. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, 1999.
- [16] N. Takagi. Powering by a Table Look-up and a Multiplication with Operand Modification. *IEEE Trans. Computers*, 47(11):1216–1222, 1998.