Complex Logarithmic Number System Arithmetic Using High-Radix Redundant CORDIC Algorithms

David Lewis
Department of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada M5S 3G4
lewis@eecg.toronto.edu

Abstract

This paper describes the application of high radix redundant CORDIC algorithms to complex logarithmic number system arithmetic. It shows that a CLNS addition can be performed with approximately the same hardware as a high-radix CORDIC operation. A design example comparable to single precision floating point has been designed and simulated.

1 Introduction

Logarithmic number system (LNS) representation has been the subject of considerable theoretical interest since its introduction [10], and a number of implementations described, e.g. [7]. Arnold recently described arithmetic transformations for efficient software implementations, as well as pointing out the advantages of complex valued LNS (CLNS) [11]. CLNS is potentially attractive in areas such as FFTs, where the powers of unity have exact representation, and complex multiplications can be easily performed using fixed point additions. However, previous VLSI implementations of LNS rely on interpolation of a function of a single variable and do not extend to CLNS. Compared to floating point representation, where a complex multiply requires 4 FP multiplies and 2 FP adds, a CLNS multiply requires only 2 fixed point adds. Consequently, if the cost of a CLNS add can be reduced below 4 FP multiplies and 4 FP adds, the total cost of a CLNS multiply-add will be less than FP.

CORDIC algorithms have long been advocated for trigonometric functions as well as complex valued exponentials and logs [1] [2] [3]. Most efforts in CORDIC have focused on real numbers, and used low radix-2 or radix-4 algorithms. Recently, BKM, a low-radix redundant CORDIC algorithm was described and used for trigonometric functions and complex arithmetic using a linear representation [8]. BKM, as most other CORDIC algorithms is a low radix method, and takes many steps to perform an operation. The simplicity of the hardware implementation of CORDIC is attractive, and a number of successful hardware implementations of CORDIC have been also been described [4] [5] [6] [9], however, these typically take a large number of stages. A few high radix methods have been described. Baker [12] described high radix CORDIC based algorithms, later extended to carry-save representation by Antelo *et al* [15]. Ahmed [13] [14] introduced a convergence method that generalized Chen's [3], useful for describing algorithms as transformations on numbers that maintain some invariant. Ahmed described CORDIC algorithms using a single high-radix step to begin, and also using linear interpolation for the latter half of the algorithm.

This paper is most closely related to Antelo's *et. al* high-radix CORDIC algorithm [15]. It applies to CLNS, and modifies this algorithm, as well as introducing some optimizations specific to CLNS that approximately halve the cost of the algorithm. Some specific points of comparison to [15] are: (1) this paper shows how optimizations specific to CLNS can eliminate approximately half the CORDIC stages (2) this paper advocates exact calculation of the minimal usable radix, instead of using a fixed radix (3) this paper extends the high-radix algorithms to include logarithm algorithms, similar to CORDIC vectoring, which requires more complex digit selection and a different sequence of operations.

The remainder of this paper describes the number representation assumed for CLNS, and the transformations that can be performed on these numbers. Hardware structures for high-radix operations are described for complex exponentiation and logarithm, together with bounds on the values at each stage. An example processor has been designed and verified down to the gate level, and its verification is described.

2 Number Representation

A complex valued number $X = X_R + X_I \cdot i$ is represented in CLNS by its logarithm, $x = x_L + x_\theta \cdot i$, such that $X = b^x$, where b is the base of the system. Both x_l and x_θ are fixed point numbers and can be represented

using 2's complement binary numbers.

Given the representations $\langle x_L, x_\theta \rangle$, $x = x_L + x_\theta \cdot i$, and $\langle y_L, y_\theta \rangle$ of two numbers X and Y, it is trivial to find the representation $\langle z_L, z_\theta \rangle$ of $Z = X \times Y$ as $z_L = x_L + y_L$ and $z_\theta = x_\theta + y_\theta$.

CLNS addition is considerably more difficult. To compute the representation of $Z = X + Y$, it is necessary to compute z_l and z_θ.

$$z_l = x_L + f_L(x_L - y_L, x_\theta - y_\theta) \quad (1)$$

$$z_\theta = x_\theta + f_\theta(x_L - y_L, x_\theta - y_\theta) \quad (2)$$

The functions f_L and f_θ are implicitly defined in terms of r, as

$$r = x - y = x_L - y_L + (x_\theta - y_\theta) \cdot i \quad (3)$$

$$f(r) = f_L(r) + f_\theta(r) \cdot i \quad (4)$$

$$f(r) = \log_b(1 + b^r) \quad (5)$$

We will assume that $x_L \geq y_L$ so that the argument to f_L and f_θ lies in the right hand half plane. Subtraction is accomplished by adding $\log_b(-1)$ to the appropriate operand.

2.1 Transformations on Complex Number Representations

In order to compute $f(r)$ as defined in (5), CORDIC-based algorithms can be applied to the computation of the complex exponential and logarithm functions. As in previous descriptions of convergence methods, we define a function that maintains a constant value through each stage in the transformation. Our 4-tuple contains a Cartesian representation of a point using the pair of real values u_j and v_j, and a polar logarithmic representation using the two real values r_j and a_j. The value represented is $T_j = (u_j + v_j \cdot i) \times b^{r_j + a_j \cdot i}$. Two transformations, scaling and rotation, are define such that the value of T_j is kept constant.

The scale transformation performs a linear scaling of the Cartesian value by a factor of $(1 + s_j)$, and compensating reduction in the logarithmic value of $\log_b(1 + s_j)$:

$$u_{j+1} = u_j \times (1 + s_j) \quad (6)$$

$$v_{j+1} = v_j \times (1 + s_j) \quad (7)$$

$$r_{j+1} = r_j + l_j \quad (8)$$

$$a_{j+1} = a_j \quad (9)$$

$$l_j = -\log_b(1 + s_j) \quad (10)$$

The rotation transformation performs a rotation of the Cartesian values based on some value q_j and compensating change in the angle of the polar logarithmic representation:

$$u_{j+1} = u_j + q_j \times v_j \quad (11)$$

$$v_{j+1} = v_j - q_j \times u_j \quad (12)$$

$$r_{j+1} = r_j + m_j \quad (13)$$

$$a_{j+1} = a_j + c_j \quad (14)$$

$$m_j = -\log_b(\sqrt{1 + q_j^2}) \quad (15)$$

$$c_j = \frac{-\text{atan}(q_j)}{\ln(b)} \quad (16)$$

The angle of the rotation is given by $-c_j$. The rotation lengthens the vector by a factor of $\sqrt{1 + q_j^2}$, and the corresponding change in the polar logarithm magnitude is given by m_j. Both the scale transformation and the rotation transformation preserve the invariant $T_{j+1} = T_j$.

Complex exponentiation and complex logarithm are implemented using a sequence of rotation and scaling transformations. In each, a series of N stages are cascaded, each of which may be a rotation or scaling transformation according to the design of the algorithm. The inputs are u_0, v_0, r_0, and a_0, and the outputs are u_N, v_N, r_N, and a_N. In each algorithm, some of the inputs and outputs are constrained to be constants. Thus, the difference in the operation of the algorithms is the way that the values s_j and q_j are determined as a function of the inputs.

2.2 Complex Exponentiation Algorithm

In complex exponentiation, the values of u_0 and v_0 are set to constants, such as 1 and 0 respectively, while r_0 and a_0 are bounded by some intervals. A series of transformations is performed such that r_N and a_N are constants regardless of the inputs r_0 and a_0. From the invariance of

T_j, we then have

$$u_N + v_N \cdot i = (u_0 + v_0 \cdot i) \times b^{(r_0 + a_0 \cdot i - r_N - a_N \cdot i)} \quad (17)$$

Without loss of generality, assume that $r_N = 0$ and $a_N = 0$. Each transformation stage must examine one of the values of r_j and a_j to determine the values of s_j and q_j. Since the goal is to bring $r_j \to r_N$ and $a_j \to a_N$, this is achieved in a series of stages each of which attempts to satisfy $r_{j+1} \approx r_N$ and $a_{j+1} \approx a_N$ to an increasing degree of precision in successive stages. Substituting these goals into (8) and (14) leads to the digit selection functions:

$$s_j \approx b^{r_j - r_N} - 1 \quad (18)$$

$$q_j \approx \tan(\ln(b) \times (a_j - a_N)) \quad (19)$$

2.3 Complex Logarithm Algorithm

In complex logarithm, the values of r_0 and a_0 are constants, the values of u_0 and v_0 are inputs, and the series of transformations is performed such that u_N and v_N are constants. The invariance of T_j leads to eqn. (20).

$$r_N + a_N \cdot i = \log_b\left(\frac{u_0 + v_0 \cdot i}{u_N + v_N \cdot i}\right) + r_0 + a_0 \cdot i \quad (20)$$

A useful choice is $u_N = 1$, $v_N = 0$, $r_0 = 0$, and $a_0 = 0$. As with the exponentiation algorithm, there is no loss in generality in assuming $u_N \neq 0$ and $v_N = 0$.

To insure that u_N and v_N are constants, the transformation stages must examine u_j and v_j to determine s_j and q_j. In this case, the goals are $u_j \approx u_N$ and $v_j \approx v_N$. The digit selection functions are potentially more complex, since both scaling and rotation affect both u_j and v_j. To reduce the complexity of digit selection, each function depends only on one of u_j and v_j. For the particular case described above, we require that s_j depend on u_j alone, and q_j depend on v_j alone. By setting $u_{j+1} \approx u_N$ in (6), and $v_{j+1} \approx v_N$ into (12), we have the digit selection functions:

$$s_j \approx \frac{u_N}{u_j} - 1 \quad (21)$$

$$q_j \approx \frac{v_j - v_N}{u_j} \quad (22)$$

q_j depends on two variables, so a constant approximation to u_j will be introduced later to simplify the function to a single argument.

2.4 CORDIC for CLNS Addition

The CLNS addition function can be constructed using a CORDIC exponentiation, adding one, followed by a CORDIC log, as illustrated in the left side of Fig. 1. Given $\langle r_0, a_0 \rangle$ as input, a series of exponential stages, with input $\langle 1, 0, r_0, a_0 \rangle$ produces output $\langle u_n, v_n, 1, 0 \rangle$. An adder then produces $u'_0 = u_n + 1$ and $v'_0 = v_n$ and performs a logarithm operation on $\langle u'_0, v'_0, 0, 0 \rangle$, producing the final result.

An optimization is possible by considering an intermediate value in the exponential stage, say j, where b^r is represented by $(u_j + v_j \cdot i) \times b^{r_j + a_j \cdot i}$. For sufficiently small r_j and a_j, apply a Taylor series approximation to $\log_b(1 + b^r)$ producing

$$\log_b(1 + b^r) = \log_b\left(1 + (u_j + v_j \cdot i) \times b^{r_j + a_j \cdot i}\right) \quad (23)$$

$$\log_b(1 + b^r) = \log_b\left(b^{-(r_j + a_j \cdot i)} + u_j + v_j \cdot i\right) + r_j + a_j \cdot i \quad (24)$$

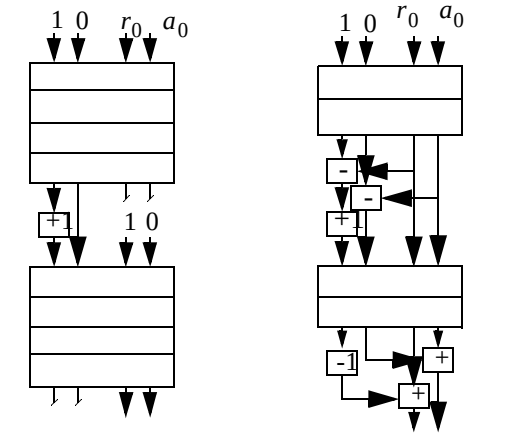
$$\log_b(1 + b^r) \approx \log_b(1 - \ln(b) \times (r_j + a_j \cdot i) + u_j + v_j \cdot i) + r_j + a_j \cdot i \quad (25)$$

For the case that $\ln(b) = 1$, the multiply disappears, and circuit uses $r'_0 = 1 - r_j + u_j$, $v'_0 = a_j + v_j$, $r'_0 = r_j$, and $a'_0 = a_j$ as inputs to the logarithm stages. This provides a strong incentive to using a base of e.

Similar improvements apply to the logarithm stage. Consider some stage j, and the Taylor series approximation for $u'_j \approx 1$ and $v'_j \approx 0$

$$\log_b\left((u'_j + v'_j \cdot i) \times b^{r'_j + a'_j \cdot i}\right) \approx \frac{1}{\ln(b)} \times (u'_j - 1 + v'_j \cdot i) + r'_j + a'_j \cdot i \quad (26)$$

Approximately half the stages can be eliminated for both exponential and logarithm, as shown in the right side of Fig. 1. Consequently, a CLNS addition can be performed with cost comparable to approximately one CORDIC operation.



Straightforward Implementation | Use of Taylor series

Figure 1 CLNS Addition Using Interpolation. Right half shows truncation of iterations using Taylor series approximation.

3 Hardware Implementation

To understand the calculation of the values of s_j and q_j, as well as the bounds on the values and precision in each stage of the computation it is useful at this point to introduce the redundant computation in terms of the hardware implementation.

The algorithms allow redundant representation using of all the quantities involved in the computation; however, the multiplications of u_j and v_j eliminate any advantage to using a redundant representation of these quantities. Instead, only r_j and a_j are represented using a carry-save form, and the non-redundant value is calculated to the accuracy required.

Truncation is explicitly represented in the hardware designs shown below with a truncation operator $\lfloor\ \rfloor$, and a separate symbol for the truncated value. The precision of any variable in the algorithm, for example some variable w, will be expressed as $P(w)$. The precision of a variable is the negative of the position in the binary representation of the least significant bit, i.e., if $w = w_{I-1}\ldots w_0 . w_{-1}\ldots w_{-F}$, then $P(w) = F$.

There are four related hardware blocks, corresponding to the scaling and rotation stages for both the exponentiation and logarithm algorithms. For both algorithms, the values u_j, v_j, r_j, and a_j are assumed to have F bits precision. Internally, each value is represented with an additional G guard bits, for a total of $F + G$ bits. In carry-save form, r_j and a_j are represented by the pairs rc_j and rs_j, and by ac_j and as_j respectively, all of which have precision $F + G$.

In the exponential scaling stage, shown in Figure 2, a reduced precision approximation of r_j is calculated by truncating rct_j and rst_j, which are added to calculate the non-redundant, but lower accuracy approximation rt_j, which is input to the digit calculation block. The purpose of this lower precision approximation is to reduce the amount of hardware required for the adder, but more importantly, to reduce the number of bits that the digit selection block must examine, and consequently reduce its associated hardware and increase its speed.

All stages described here contain digit selection blocks. Before describing the specific functions implemented in them, a brief description of their logic structure is useful. A digit selection block implements some monotonic function of a single input, and is a piecewise constant approximation to some continuous function. Using the specific example of a digit selection block with rt_j as an input and s_j as output, with $s_j = f(rt_j)$ for some $f()$, the function can be expressed in the form (27), where the digit is k and $k \times 2^{-P(s_j)}$ is the piecewise constant approximation to the function over some interval $[thresh_k, thresh_{k+1})$.

$$s_j = k \times 2^{-P(s_j)} \quad thresh_k \le rt_j < thresh_{k+1}, \quad k_{min} \le k \le k_{max} \tag{27}$$

$$thresh_k = f^{-1}\left(\left(k - \frac{1}{2}\right) \times 2^{-P(s_j)}\right) \tag{28}$$

The value of k may take any value in the range k_{min} to k_{max}, where the bounds are chosen to include the entire range of inputs to the function. The number of distinct values that can be produced is $k_{max} - k_{min} + 1$ and is referred to as the radix of the value rt_j. Expressing the function at this level of detail explicitly provides the range over which each input value produces some output value, and makes it straightforward to bound the result of a calculation. The

hardware implementation of this will be discussed later, but it is clear that the two primary factors involved are the number of bits in the input value which need to be examined, and the number of distinct output values that can be produced, or equivalently, the radix.

The digit selection block in the scaling exponentiation stage produces s_j and $l_j = -\log_b(1 + s_j)$. Assuming that the goal is $r_N = 0$, we desire that $l_j \approx -r_j$ to the precision possible given the fixed number of bits in the representation of s_j. Note that l_j is expressed to the full precision of the datapath, while it is the limited number of bits in s_j that restrict the set of possible values of l_j. This is due to the desire to reduce the size of the digit estimation logic for s_j.

The values of u_j and v_j are also truncated to a lower precision forming ut_j and vt_j, and a pair of multiply-adders is used to calculate the results specified in Eqns. (6) and (7). The multiplier output is left in carry-save form, which avoids a carry-propagate addition inside it. This is advantageous as the result of the multiplier is later added using a CSA and CPA. The multiplier result is also truncated to the precision of u_j and v_j. Eqn (8) is calculated using a CSA because of the redundant representation of r_j.

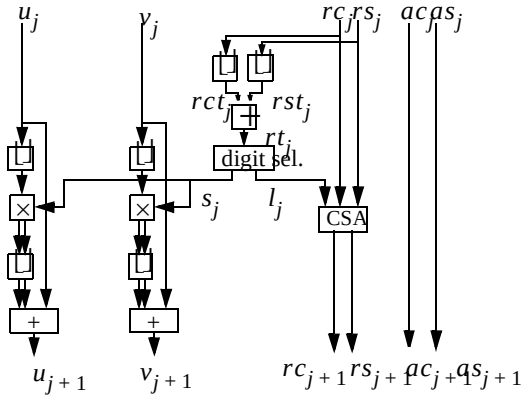


Figure 2 Scaling Stage for Exponentiation

Figure 3 illustrates the hardware for the rotation stage for exponentiation. A reduced precision approximation to a_j is calculated and input to the digit calculation block. This produces q_j and is used, together with reduced precision values of u_j and v_j to perform the calculation in Eqns. (11) through (14). As before, calculations involving r_j and a_j are preformed with CSAs.

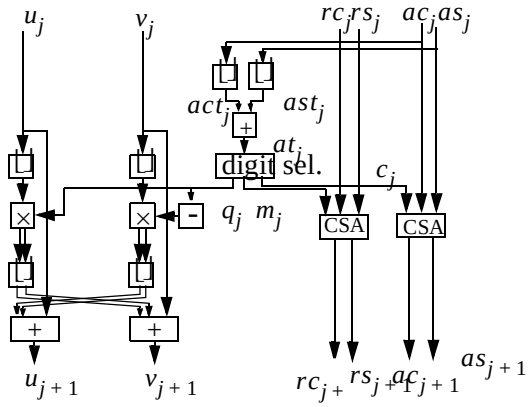


Figure 3 Rotation Stage for Exponentiation

Figures 4 and 5 illustrate the rotation and scaling stages for the logarithm algorithm. These are similar to the previous two, except that u_j and v_j are used for determining s_j and q_j respectively. This complicates the analysis of the bounds on the intermediate results, as each digit selection block can only access part of the problem state.

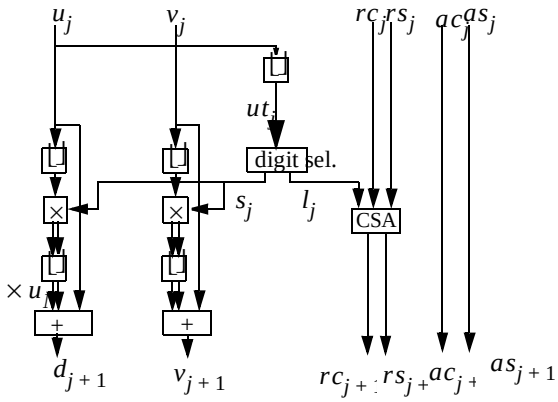


Figure 4 Scaling Stage for Logarithm

4 Bounds on Intermediate Results

Antelo *et al* advocate high-radix CORDIC using a maximally redundant digit set. They suggest choosing a radix, and show that the algorithm will converge using the maximally redundant digit set. In this paper, we suggest explicit calculation of the minimal possible redundant digit set, and exact digit selection based on the truncated data for two reasons First, explicitly calculating the radix means that the

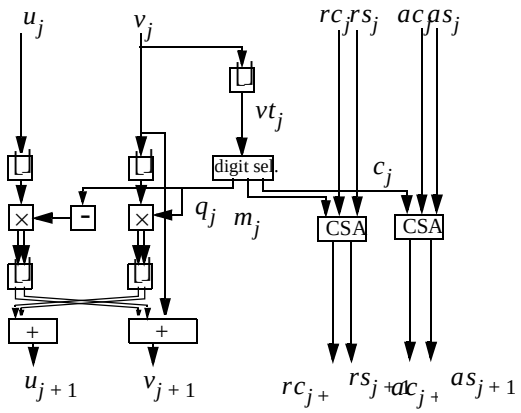


Figure 5 Rotation Stage for Logarithm

tables need only be large enough to accommodate the actual number of values, which can be smaller than the radix implied by the maximally redundant digit set. Second, the size of the multipliers can potentially be reduced if the set of values to be multiplied is bounded by a smaller range than the representation allows. The latter of these depends on the observation that the core of an n-bit modified Booth multiplier value can be used to multiply values in the range $\left[-\frac{5}{4}\times 2^{n-1}, \frac{5}{4}\times 2^{n-1}-1\right]$. This range would conventionally require an $n+2$ bit multiplier. For example, multiplication by integers in $[-40, 39]$ can be performed with a 6-bit multiplier, but would conventionally require an 8-bit multiplier.

The approach in this section is constructive. We define the bounds of the operands at each stage, and determine the relationship between the bound on the input and output for each type of transformation. For each possible digit k, the range of inputs that use the digit are specified, and the resulting bound on the output is determined. Given the overall bound on the range of inputs, and the precision of the digits s_j and q_j, it is possible to determine the set of values k required to span the entire input range. and consequently determine the radix of the digit set. By computing the union of all output bounds for every possible digit in a stage, an overall bound on the output of a stage can be calculated. It is also necessary to specify the precision of the truncated quantities input to the digit selection logic.

It will be useful to have a concise notation for the truncation or rounding of quantities to various fixed point precisions. We use $\lfloor x \rfloor_P$ to mean the value of x truncated down to P bits precision, so the definition is given as:

$$\lfloor x \rfloor_P = \left\lfloor x \times 2^P \right\rfloor \times 2^{-P} \tag{29}$$

Similar notation for rounding up, and round to nearest are also used:

$$\lceil x \rceil_P = \left\lceil x \times 2^P \right\rceil \times 2^{-P} \tag{30}$$

$$[x]_P = \left\lfloor x \times 2^P + 2^{P-1} \right\rfloor \times 2^{-P} \tag{31}$$

It is also necessary to bound the variables at each stage. We introduce min and max subscripts such that any variable is bounded by its min and max as in $u_{min,j} \le u_j < u_{max,j}$.

4.1 Scaling Stage for Exponential

As mentioned, it is desirable that $l_j \approx r_j$, or equivalently, $s_j \approx b^r - 1$. Ideally, the digit selection function would implement

$$s_j = \left\lfloor b^{rt_j} - 1 \right\rfloor_{P(s_j)}$$

Transforming this into the piecewise-constant expression leads to the exact computation of the digit s_j:

$$\begin{aligned} s_j &= k \times 2^{-P(s_j)} \\ &\log_b\left(1 + \left(k - \frac{1}{2}\right) \times 2^{-P(s_j)}\right) \le rt_j \\ &< \log_b\left(1 + \left(k + \frac{1}{2}\right) \times 2^{-P(s_j)}\right) \end{aligned}$$

$$k_{min} \le k \le k_{max} \tag{32}$$

The range of values of k is required to be sufficiently large that all possible values of rt_j will fall in one such interval. The truncation of the carry save value of r_j bounds on rt_j:

$$r_j - 2^{1-P(rt_j)} + 2^{1-F-G} \le rt_j \le r_j \tag{33}$$

Substituting (33) into produces constraints for k_{min} and k_{max}, which can be further simplified into explicit requirements for these two bounds on k:

$$\begin{aligned} &\log\left(1 + \left(k_{min} - \frac{1}{2}\right) \times 2^{-P(s_j)}\right) \\ &+ 2^{1-P(rt_j)} - 2^{1-F-G} \le r_{min,j} \end{aligned} \tag{34}$$

$$\log\left(1+\left(k_{max}+\frac{1}{2}\right)\times 2^{-P(s_j)}\right)\geq r_{max,j} \tag{35}$$

Note that in these equations the rounding of $\log_b()$ to $F+G$ precision is not specified; however, whatever method is used must be applied consistently across all of these equations to obtain correct results.

Bounds on r_{j+1} are calculated by first noting that rt_j is bounded by (33) due to the truncation of the carry and sum components from $F+G$ precision before their addition. For each such value of k that is required, it is possible to use (33) to determine the bounds on r_j

$$rt_j \leq r_j \leq rt_j + 2^{1-P(rt_j)} - 2^{1-F-G} \tag{36}$$

The definition of s_j is then substituted into (8) and (10) to produce bounds for each given value of k:

$$\log_b\left(1+\left(k-\frac{1}{2}\right)\times 2^{-P(s_j)}\right) - \log_b\left(1+k\times 2^{-P(s_j)}\right)\leq r_{j+1} \tag{37}$$

$$r_{j+1} < \log_b\left(1+\left(k+\frac{1}{2}\right)\times 2^{-P(s_j)}\right) + 2^{1-P(rt_j)} - 2^{1-F-G} - \log_b 1+k\times 2^{-P(s_j)} \tag{38}$$

It is possible to attempt to construct an explicit bound for all possible k by substituting in the appropriate k_{min} and k_{max}, but it is difficult to guarantee an exact bound in the presence of multiple roundings to various precisions. Instead, we simply iterate across all values, taking the minimum and maximum of these bounds to determine overall bounds on r_{j+1}

It is useful to ignore the redundancy and take a first order Taylor series approximation to (37) and (38) into obtain insight into the operation of the algorithm, although the exact form of equation must be used for computing bounds.

$$\left|r_{j+1}\right| \leq \ln(b)\times 2^{-1-P(s_j)} \tag{39}$$

Eqn (39) shows that each stage reduces the magnitude of r_{j+1} to roughly one-half ULP in the digit's representation.

4.2 Rotation Stage for Exponentiation

A similar approach can be taken for the rotation stage, with the notable difference that the rotation is computed as a function of a_j, but affects both a_{j+1} and r_{j+1}. In the rotation stage, we desire $q_j = [\tan(\ln(b)\times a_j)]_{P(q_j)}$. The piecewise constant approximation is given as

$$q_j = k\times 2^{-P(q_j)}$$

$$\frac{\operatorname{atan}\left(\left(k-\frac{1}{2}\right)\times 2^{-P(q_j)}\right)}{\ln(b)} \leq at_j < \frac{\operatorname{atan}\left(\left(k+\frac{1}{2}\right)\times 2^{-P(q_j)}\right)}{\ln(b)}$$

$$k_{min}\leq k\leq k_{max} \tag{40}$$

Exact bounds on k can be determined by the previous approach and will not be presented.

Bounds on a_{j+1} can be found by a similar approach to the scaling stage. This leads to the following bounds on a_{j+1}, where the union of all such bounds must be taken for the entire range of k

$$\frac{\operatorname{atan}\left(\left(k-\frac{1}{2}\right)\times 2^{-P(q_j)}\right)}{\ln(b)} - \frac{\operatorname{atan}\left(k\times 2^{-P(q_j)}\right)}{\ln(b)} \leq a_{j+1} \tag{41}$$

$$a_{j+1} < \frac{\operatorname{atan}\left(\left(k+\frac{1}{2}\right)\times 2^{-P(q_j)}\right)}{\ln(b)} - \frac{\operatorname{atan}\left(k\times 2^{-P(q_j)}\right)}{\ln(b)} + 2^{1-P(at_j)} - 2^{1-F-G} \tag{42}$$

For small a_j, ignoring redundancy and taking a first-order Taylor series approximation shows that a_{j+1} is bounded by approximately half an ULP of q_j.

$$\left|a_{j+1}\right| < \frac{2^{-1-P(q_j)}}{\ln(b)} \tag{43}$$

The choice of q_j is independent of r_j, so the bound on r_{j+1} can only be given by subtracting the minimum and

maximum possible values of m_j from the bounds on r_j,

$$r_{min,j} - \max_k \left(\log_b \left(\sqrt{1 + k^2 \times 2^{-2 \times P(at_j)}} \right) \right) \le r_{j+1} \quad (44)$$

$$r_{j+1} \le r_{max,j} - \min_k \left(\log_b \left(\sqrt{1 + k^2 \times 2^{-2 \times P(at_j)}} \right) \right) \quad (45)$$

Because k may be signed, it is not possible to express these in a form only using one of k_{min} or k_{max} in each equation.

4.3 Scaling Stage for Logarithm

The logarithm stages are more complicated to understand as both scaling and rotation affect both u_{j+1} and v_{j+1}, but each stage must perform a transformation based on only one value. Fig. 6 illustrates the scaling transformation. The vertical dashed lines represent the bounds between values of u_j that lead to distinct values of s_j. Each such bound leads to an interval in u_j and v_j that is a rectangular interval subject to scaling by the same value of $1 + s_j$. The arrows show the linear scaling by $1 + s_j$ for the upper left point in two of the rectangles. Although the range of u_{j+1} is reduced compared to the range of u_j, the range of v_{j+1} is increased compared to v_j.

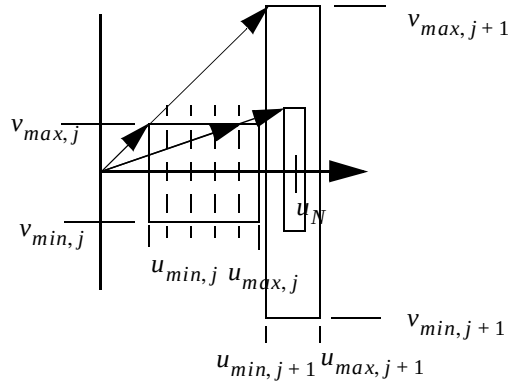


Figure 6 Operation of Scaling Stage in Logarithm Algorithm

Recall that we desire $s_j \approx \frac{u_N}{u_j} - 1$. In the piecewise constant form, this can be achieved by

$$s_j = k \times 2^{-P(s_j)}$$

$$\frac{u_N}{\left(k + \frac{1}{2}\right) \times 2^{-P(s_j)} + 1} \le ut_j < \frac{u_N}{\left(k - \frac{1}{2}\right) \times 2^{-P(s_j)} + 1},$$

$$k_{min} \le k \le k_{max} \quad (46)$$

This establishes following bound on u_{j+1}. Although $k = k_{min}$ will give the largest range, the calculation should be performed to determine the union of all ranges for all values of k using exactly the same rounding as the hardware in order to obtain precise bounds.

$$u_N \times \frac{k \times 2^{-P(s_j)} + 1}{\left(k + \frac{1}{2}\right) \times 2^{-P(s_j)} + 1} \le u_{j+1} \quad (47)$$

$$u_{j+1} < \left(\frac{u_N}{\left(k - \frac{1}{2}\right) \times 2^{-P(s_j)} + 1} + 2^{1 - P(ut_j)} - 2^{1 - F - G} \right) \times \left(k \times 2^{-P(s_j)} + 1 \right) \quad (48)$$

The value of v_j is scaled by the same factor, so v_{j+1} can be bounded by the union of the intervals for all values of k

$$v_{min,j} \times \left(k \times 2^{-P(s_j)} + 1 \right) \le v_{j+1} < v_{max,j} \times \left(k \times 2^{-P(s_j)} + 1 \right) \quad (49)$$

4.4 Rotation Stage for Logarithm

The rotation stage for logarithm presents the most difficulty, as the goal $q_j \approx \frac{v_j}{u_j}$ depends on two values. We simplify this to a function of a single value by using the bound on u_j and taking the midpoint of this range as an approximation. This results in $q_j = \left\lfloor \frac{v_j}{\left(\frac{u_{min,j} + u_{max,j}}{2} \right)} \right\rfloor_{P(q_j)}$ as an approximate goal for q_j. This operation of this stage is shown graphically in Fig. 7. The horizontal dashed lines represent the thresholds for values of v_j that result in dis-

tinct values of q_j, and that apply to a rectangular region in u_j and v_j. The figure shows the rotation and stretching of two of these regions, illustrating the center point of each, together with the bounding box for the rotated and stretched rectangle. Because each of the intervals is rotated, the tightest possible bounds of the resulting values u_{j+1} and v_{j+1} does not form a rectangle; however, for simplicity of analysis, it is considered to be the smallest rectangle that encloses all of the rotated and stretched intervals, and is illustrated with the dotted rectangle in the Fig. 7. The figure is not to scale, and typically the rectangle bounding the would be much smaller than the input bounds.

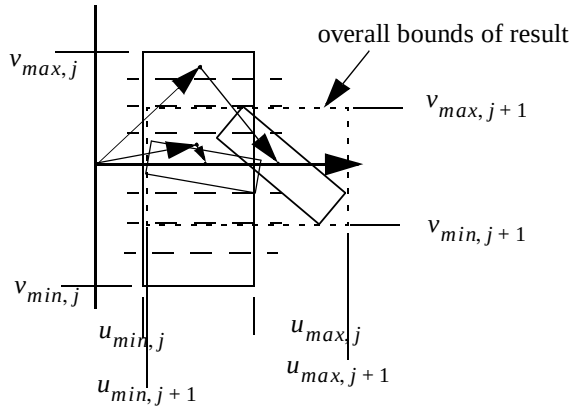


Figure 7 Operation of Rotation Stage in Logarithm Algorithm

The exact definition of q_j is

$$q_j = k \times 2^{-P(q_j)}$$

$$\left(k-\frac{1}{2}\right)\times 2^{-P(q_j)} \times \left(\frac{u_{min,j}+u_{max,j}}{2}\right) \le vt_j < \left(k+\frac{1}{2}\right)\times 2^{-P(q_j)} \times \left(\frac{u_{min,j}+u_{max,j}}{2}\right)$$

$$k_{min} \le k \le k_{max} \tag{50}$$

This leads to the following bounds on u_{j+1} and v_{j+1}, where, as usual, all values of k must be considered:

$$\left(k-\frac{1}{2}\right)\times 2^{-P(q_j)} \times \left(\frac{u_{min,j}+u_{max,j}}{2}\right) - k \times 2^{-P(q_j)} \times u_{max,j} \le v_{j+1} \tag{51}$$

$$v_{j+1} < \left(k+\frac{1}{2}\right)\times 2^{-P(q_j)} \times \left(\frac{u_{min,j}+u_{max,j}}{2}\right) + 2^{1-P(vt_j)} + 2^{1-F-G} - k \times 2^{-P(q_j)} \times u_{min,j} \tag{52}$$

$$u_{min,j} + k \times 2^{-P(q_j)} \times v_{min,j} \le u_{j+1} \tag{53}$$

$$u_{j+1} < u_{max,j} + k \times 2^{-P(q_j)} \times v_{max,j} \tag{54}$$

5 Example for 32-bit Complex Numbers

The exponential stages calculate both digits based on a single operand, but the value of a_j affects the value of r_{j+1} as well. It is optional whether to interleave rotation and scaling stages, or to perform all of the scaling at the end, as in [9]. In the logarithm stage, both u_{j+1} and v_{j+1} depend on u_j and v_j. It is necessary to alternate scaling and rotation stages in order to tighten bounds on both simultaneously.

As a demonstration of the feasibility of this approach, we have designed a high-radix multiply-add CLNS arithmetic unit with precision comparable to IEEE-754 single precision. The unit performs complex multiplication using two fixed point adders, and uses high-radix exponentiation and logarithm to perform complex addition using the function $f(r)$ defined in (4). The design uses $F = 24$ and $G = 4$, and some other minor changes to the constants assumed in the derivation above. The number representation uses a mixed base for the representation of the numbers to simplify range reduction. A number X is represented by its complex logarithm $x = x_L + x_\theta \cdot i$, where x_L is a 32-bit 2's complement fixed point number, and x_θ is a 27-bit unsigned fixed point number, both of which have 24 fractional bits.

The details of the algorithm were designed with the assistance of a program that has as input an architecture description file containing all of the precisions of each variable, and performs exact bit-level modeling of the architecture.

Our design uses a total of 10 stages to perform an exponentiation and a logarithm as required by the logarithmic addition function. Datapath widths were based on 6 bit digits requiring two stages of each of rotation and scaling for exponentiation, and three of each for logarithm. Antelo *et al* [9] would require 8 stages using 7-bit multipliers to perform a rotation to the same precision; thus, our architecture requires little more hardware to perform a CLNS addition.

Beyond this, only two more fixed point adders are required to perform a CLNS multiply-add. Table 1 shows the key parameters of the design. The digit precision refers to the digit generated, either s_j or p_j, and the digit selection precision refers to the input to the digit selection block, such as at_j or other values.

Table 1: Hardware Parameters of Example CLNS ALU

Stage Type	Digit Precision	Digit Selection Precision	Radix
Exponential Algorithm			
R	6	10	65
S	6	10	118
R	12	16	74
S	12	16	77
Logarithm Algorithm			
S	6	10	65
R	6	10	65
S	6	10	67
R	12	16	82
S	12	16	69
R	18	22	110

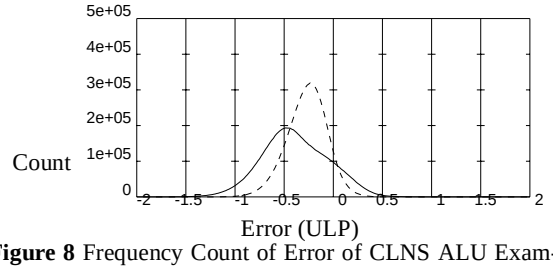


Figure 8 Frequency Count of Error of CLNS ALU Example: Solid line: real; Dashed: Imaginary. Bin width is .01

Fig. 8 illustrates the simulator's real and imaginary error histogram in ULPs for 10^7 pseudo-random tests. A bias is clear, and is due to the use of truncation rather than rounding in the datapath. Mean error and bias are each less than 0.4 ULP, but worst case error is 1.5 ULP.

6 Conclusions

This paper has demonstrated high-radix CORDIC algorithms adapted for CLNS addition. A design example producing six bits per stage as an illustration shows that a CLNS addition can be performed for approximately the same cost as a conventional high-radix CORDIC rotation. Since a CLNS multiply is inexpensive, this allows a CLNS multiply-accumulate to be performed for the cost of a single CORDIC operation.

7 References

[1] J. Volder, "The CORDIC Computing Technique", *IRE Trans. Comput.*, Sept. 1959, pp. 330-334

[2] J. Walther, "A Unified Algorithm for Elementary Functions", *Spring Joint Comp. Conf.*, 1971, pp 379-385

[3] T. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios, and Square Roots", *IBM J. Res. Dev*, 1972, pp 380-388

[4] A. Madisetti, A. Kwentus, and A. Willson, "A Sine/Cosine Direct Digital Frequency Synthesizer Using an Angle Rotation Algorithm", *ISSCC-95*, pp. 262-263

[5] A. Skaf, J.-M. Mullar, and A. Guyot, "On-line Hardware Implementation for Complex Exponential and Logarithm", *Twentieth European Solid-State Circ. Conf.*, 1994, pp 252-255

[6] D. Timmermannm B. Rix, H. Hahn, and B. Hosticka, "A CMOS Floating-Point Vector Arithmetic Unit", *IEEE J. Solid State Circuits*, May 1994, pp 634-639

[7] D. Lewis,"A 114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications", *IEEE J. Solid-State Circuits*, Dec 1995, pp 1547-1553

[8] J.-C. Bajard, S. Kla, and J.-M. Muller, "BKM: A New Hardware Algorithm for Complex Elementary Functions", *IEEE Trans. Comput*, Aug 1994, pp 955-964

[9] E. Antelo, J. Villalba, J. Bruguera, and E. Zapata, "High Performance Rotation Architectures Based on the Radix-4 CORDIC Algorithm", *IEEE Trans. Comput.*, Aug 1997, pp 855-870.

[10] E. Swartzlander and A. Alexopolous, "The Sign/Logarithm Number System", *IEEE Trans. Comput.*,Dec 1975, pp. 1238-1242

[11] M. Arnold, T. Bailey, J. Cowles, and M. Winkel, "Arithmetic Co-Transformation in the Real and Complex Logarithmic Number Systems", *IEEE. Trans. Comput.*, July 1998, pp 777-786

[12] P. Baker, "Parallel Multiplicative Algorithms for Some Elementary Functions", *IEEE Trans. Comput.*, March 1975, pp 322-325

[13] H. Ahmed, *Signal Processing Algorithms and Architectures*, PhD Thesis, Stanford University, 1982

[14] H. Ahmed, *Efficient Elementary Function Generation with Multipliers*, Proc. 9th Symp. Comp. Arith, 1989, pp 52-59

[15] E. Antelo, J. Brugerea, T. Lang, J. Villalba, and E. Zapata, "High Radix Cordic Rotation based on Selection by Rounding", *Intl. European Conf. on Parallel Proc., Euro-Par 96*, pp 155-164