Arithmetic Co-transformations in the Real and Complex Logarithmic Number Systems

Mark G. Arnold, Thomas A. Bailey, John R. Cowles,
University of Wyoming
Department of Computer Science
Laramie, WY 82071
marnold@uwyo.edu

Mark D. Winkel
Somatogen, Inc.
2545 Central Ave.
Boulder, CO 80301
mwinkel@csn.net

Abstract

The real logarithmic number system, which represents a value with a sign bit and a quantized logarithm, can be generalized to create the complex logarithmic number system, which replaces the sign bit with a quantized angle in a log/polar coordinate system. Although multiplication and related operations are easy in both real and complex systems, addition and subtraction are hard, especially when interpolation is used to implement the system. Both real and complex logarithmic arithmetic benefit from the use of co-transformation, which converts an addition or subtraction from a region where interpolation is expensive to a region where it is easier. Two co-transformations that accomplish this goal are introduced. The first is an approximation based on real analysis of the subtraction logarithm. The second is based on simple algebra that applies for both real and complex values and that works for both addition and subtraction.

1. Introduction

The real logarithmic number system (also known as LNS [32], exponential floating point [26], CRD [7], sign/logarithm number system [29], and Gaussian logarithmic arithmetic[11]) uses a finite approximation for the logarithm of the absolute value of a real and a bit for the sign of that real to represent that value. From a theoretical standpoint, the numerical properties of this number system are, for the most part, an idealization [17] of floating point. It has been shown that floating point and real logarithmic arithmetic are extreme cases in a spectrum of "semi-logarithmic number systems" [20], and logarithmic representation has been proposed as part of a new standard for "composite arithmetic" [12]. From a practical standpoint, the real logarithmic number system has a tremendous advantage over floating point: multiplication and division can be implemented using low cost fixed point addition and subtraction.

The difficulty faced by an implementor deals with addition and especially subtraction [2] of values represented as logarithms. These two operations require computing functions that represent incrementation and decrementation of the corresponding real values. For example, given a fixed point logarithm, Z_L, of a positive real value, Z, the system must compute the *addition logarithm*,

$$S_b(Z_L) = \log_b(1 + b^{Z_L}), \tag{1}$$

in order to obtain the representation of $Z + 1$. For low precision systems, these functions can be pre-computed and placed in a read only memory (ROM). For systems requiring precision approaching that of single precision IEEE 754 floating point (23 bits), interpolation [15, 3] may be used to approximate these functions. For real (i.e., fixed point) Z_L, the $S_b(Z_L)$ function is well suited for linear or higher order interpolation because the extrema of its second and higher order derivatives are reasonably small. On the other hand, the function used for subtraction, known as the *subtraction logarithm*,

$$D_b(Z_L) = \log_b |1 - b^{Z_L}|, \tag{2}$$

has a singularity at zero, and so it is costly to interpolate.

Several approaches for high precision real valued logarithmic subtraction have been proposed. Stouraitis [27] proposed partitioning the ROM at powers of two into increasingly denser intervals as Z_L approaches zero. Lewis [14] fabricated a logarithmic arithmetic unit (with 20 bits of precision) that combines partitioning with interpolation to achieve a constant fixed point accuracy for $D_b(Z_L)$ as Z_L approaches zero. A similar combination of partitioning and interpolation was disclosed independently in [5]. Arnold et. al. [1] proposed that due to catastrophic cancellation that occurs during floating point subtraction, the required accuracy for $D_b(Z_L)$ interpolation diminishes as Z_L approaches zero. Lewis [16] fabricated a logarithmic arithmetic unit (with 23 bits of precision) using second order interpolation that capitalizes on this diminished required accuracy. Paliouras et. al. [24] suggested computing $\log_b((1-b^{Z_L})/Z_L)+\log_b(Z_L)$ for Z_L near zero instead of directly interpolating $D_b(Z_L)$, but the drawback of this approach is that it requires interpolating for two functions instead of one. Orginos et. al. [23] avoided the problem by converting to fixed point all but the maximum operand in a multi–operand subtraction, however this also requires interpolating more than one function.

An alternative approach is to re-arrange a series of computations that involves a reasonable proportion of subtraction in order to reduce the number of subtractions that must be performed. For example, if $X_1, X_2, \ldots, X_n$ are positive real numbers, the computation $(((\cdots(((X_1-X_2)+X_3)-X_4)\cdots)+X_{n-1})-X_n)$ can be re-arranged as $(X_1+X_3+\cdots+X_{n-1})-(X_2+X_4+\cdots+X_n)$. Arnold et. al. [1] proposed a redundant logarithmic number system, which represents both a positive and negative component of each real value. This redundant representation often becomes ill conditioned and loses considerable accuracy compared to the conventional real valued logarithmic number system. It also doubles the storage requirements. For these reasons, the redundant logarithmic number system is only suitable in specialized applications, where the designer has detailed knowledge of the nature of the computations to be performed.

Of course, real logarithmic arithmetic was widely used in manual computation for over three centuries until the widespread adoption of digital electronics. Despite the ubiquity of fixed and floating point hardware available today, real logarithmic arithmetic has found recent practical application primarily in two areas: low precision applications running on specialized hardware (image and digital signal processing [30], graphics [13], aircraft controls [25], hearing aids [21], and neural nets [6]) and higher precision applications running on microprocessors that lack floating point hardware [10], such as the Fujitsu SparcLite 930 [28]. Sections 4 and 6 describe two new *co-transformation* techniques for real valued logarithmic subtraction that improve its cost effectiveness while maintaining reasonable accuracy.

Many of the practical applications of low precision logarithmic arithmetic involve complex numbers, for instance the Fast Fourier Transform (FFT). The conventional way of implementing complex arithmetic is to work with pairs of real numbers that represent points in a rectangular coordinate system. Multiplication of two complex numbers, X and Y, in the rectangular coordinate system requires separate computation of $\Re[X]\cdot\Re[Y]-\Im[X]\cdot\Im[Y]$ and $\Re[X]\cdot\Im[Y]+\Im[X]\cdot\Re[Y]$, where $\Re[X]$ is the real part of X and $\Im[X]$ is the imaginary part. This approach can be used regardless of the underlying implementation of the real addition, subtraction, and multiplications. Swartzlander et. al. [31] compared floating point, fixed point and real logarithmic number systems for an FFT implemented with rectangular coordinate complex arithmetic. Section 9 describes a new number system, known as the complex logarithmic number system, which represents each complex point in log/polar coordinates. Co-transformation techniques similar to those for the real logarithmic number system are useful for practical implementation of the complex logarithmic number system.

2. The Real Logarithmic Number System

In sections 2 through 8, upper case variables and functions are used for real values, and lower case variables are used for integer data. If $X \neq 0$ and $Y \neq 0$ are real numbers, the following identities hold when the transcendental S_b, D_b and $\log_b$ functions are computed *exactly*:

$$\begin{aligned}\log_b(|X|\cdot|Y|) &= \log_b|X|+\log_b|Y| &(3)\\ \log_b(|X|/|Y|) &= \log_b|X|-\log_b|Y| &(4)\end{aligned}$$

$$\begin{aligned}\log_b(|X|+|Y|) &= \log_b(|Y|\cdot(|X|/|Y|+1)) &(5)\\ &= \log_b|Y| \\ &\quad + S_b(\log_b|X|-\log_b|Y|) \\ \log_b(||X|-|Y||) &= \log_b(|Y|\cdot(||X|/|Y|-1|)) &(6)\\ &= \log_b|Y| \\ &\quad + D_b(\log_b|X|-\log_b|Y|)\end{aligned}$$

where $b > 0$ is the base of the logarithms and S_b and D_b are defined in (1) and (2). In typical modern implementations, $b = 2$, and in the nineteenth-century literature[11], $b = 10$. In an actual implementation, the transcendental functions cannot be computed exactly since $\log_b|X|$ will be stored in a fixed size word. The computation of $\log_b|X|$ can be thought of as an input conversion, analogous to the conversion of a decimal input to IEEE 754 floating point, and so the error is related to the number of bits allowed in the representation. To understand this error, one must describe the effects of input quantization that occur when a real value, X, is converted to a logarithmic representation, x, composed of an integer, x_L and sign bit, x_θ:

$$x_\theta = \begin{cases} 0 & \text{if } X \geq 0 \\ 1 & \text{if } X < 0 \end{cases} \quad (7)$$

and

$$\begin{aligned} x_L &= o_L\left(\left\lfloor \frac{\log_b |X|}{\Delta_L} \right\rfloor\right) = o_L\left(\left\lfloor \frac{X_L}{\Delta_L} \right\rfloor\right) \\ &= o_L(\lfloor \log_B |X| \rfloor) \end{aligned} \quad (8)$$

where overflow and underflow are dealt with by

$$o_L(x) = \begin{cases} m_L & \text{if } x > m_L \\ x & \text{if } -m_L \leq x \leq m_L \\ -m_L & \text{if } x < -m_L \end{cases} \quad (9)$$

and where Δ_L and m_L are chosen so that $B = b^{\Delta_L}$ is the closest real value larger than 1.0 that can be represented exactly ($x_L = 1$) and B^{m_L} is the largest real value that can be represented exactly ($x_L = m_L$). Therefore, $-\log_2(\Delta_L)$ is roughly the number of bits of precision. We introduce B as a notational simplification which may be omitted in an actual implementation. Use of base B with the integer x_L is equivalent to the use of base b with the fixed point, $x_L \cdot \Delta_L$. The latter has $-\log_2(\Delta_L)$ bits after the binary point, as is often described for implementations in the literature. For instance, $B = 1.0000000826$ and $\Delta_L = 2^{-23}$ are roughly equivalent to IEEE 754 single precision.

Given the real logarithmic representations, x and y, the result, z, of division is:

$$\begin{aligned} z_\theta &= (x_\theta - y_\theta) \bmod m_\theta \\ z_L &= o_L(x_L - y_L) \end{aligned} \quad (10)$$

where $m_\theta = 2$ for the real logarithmic number system, and so the modulo two subtraction can be implemented simply with an exclusive OR. When x and y are exact representations, (10) is exact. If x and y are not exact, (10) propagates the error analogously to floating point. A similar situation applies to multiplication, except the "$-$"s are replaced with "$+$"s. Negation is implemented as multiplication by the representation of minus one ($y_\theta = 1, y_L = 0$).

The algorithm for real logarithmic addition and subtraction requires first computing z, the representation of the ratio of the numbers, as shown in (10). The representation, r, of the sum is:

$$\begin{aligned} r_\theta &= f_\theta(y, z) \bmod m_\theta \\ r_L &= o_L(f_L(y, z)) \end{aligned} \quad (11)$$

where

$$f_\theta(y,z) = \begin{cases} (y_\theta + z_\theta) \bmod 2 & \text{if } z_L > 0 \\ y_\theta & \text{if } z_L \leq 0 \end{cases} \quad (12)$$

and

$$f_L(y,z) = \begin{cases} y_L + d_B(z_L) & \text{if } z_\theta = 1 \\ y_L + s_B(z_L) & \text{if } z_\theta = 0 \end{cases} \quad (13)$$

We will use the notation $f(y,z)$ to describe computing the quantized logarithmic representation, $(f_L(y,z), f_\theta(y,z))$, of the sum, $X + Y$. Note $f_\theta(y,z)$ is either x_θ or y_θ, depending on whether $|X| > |Y|$ or $|X| \leq |Y|$. Subtraction is implemented as negation followed by addition. (13) uses the quantized addition logarithm,

$$s_B(z_L) = \lfloor S_B(z_L) + E_s(z_L) \rfloor = \lfloor S_B(z_L) \rfloor + e_s(z_L), \quad (14)$$

and the quantized subtraction logarithm,

$$d_B(z_L) = \lfloor D_B(z_L) + E_d(z_L) \rfloor = \lfloor D_B(z_L) \rfloor + e_d(z_L), \quad (15)$$

where $e_s(z_L)$ and $e_d(z_L)$ are quantization errors determined by the approximation method(s) used. The problem addressed in the sections 4 and 6 is that given equal resources for interpolation, $\max|e_d| \gg \max|e_s|$.

3. Co-transformations in Computer Arithmetic

Similarly to Chen [9], we define a co-transformation of two values Y_k and Z_k as:

$$Y_{k+1} = U_k(Y_k, Z_k) \quad (16)$$

and simultaneously:

$$Z_{k+1} = V_k(Y_k, Z_k) \quad (17)$$

where U_k and V_k are functions chosen to preserve some approximate relationship:

$$F(Y_k, Z_k) \approx F(Y_{k+1}, Z_{k+1}) \quad (18)$$

The function F is chosen so that one or more applications of the co-transformation will yield a meaningful result. For a co-transformation to have practical utility in computer arithmetic, the functions U_k and V_k should be relatively economical to implement, and the result of n applications of the co-transformation should make $F(X_{k+n}, Y_{k+n})$ "closer" to a desired goal than the original $F(X_k, Y_k)$. In sections 4, 6, 8 and 10, we introduce novel co-transformations that are specifically designed to make computation of S_b and D_b easier.

Some simple co-transformations that reduce the cost of implementing logarithmic arithmetic are well known, such as that for commutativity. As we will do for the novel co-transformations described later, we begin the discussion of this trivial commutativity co-transformation by giving analytical and/or algebraic background. Then we use this background to derive the actual co-transformation found in the implementation.

In this trivial example, we start with the commutativity of X and Y for real addition, $X + Y = Y + X$. With (5) in mind, commutativity implies $X \cdot (1 + Y/X) = Y \cdot (1 + X/Y)$ for $X \neq 0$ and $Y \neq 0$, In a quantized implementation, this is equivalent to $f(x, \tilde{z}) = f(y, z)$, where $\tilde{z}$ represents the reciprocal of the value represented by z, in other words, $\tilde{z}_L = -z_L$ and $\tilde{z}_\theta = (-z_\theta) \bmod 2$. Because of the properties of modulo two arithmetic, $\tilde{z}_\theta = z_\theta$.

With the algebraic background above, we are ready to define the well known co-transformation that reduces the table size in half for logarithmic addition and subtraction:

$$\begin{aligned}
u_L(y,z) &= \begin{cases} y_L & \text{if } z_L \geq 0 \\ y_L + z_L & \text{if } z_L < 0 \end{cases} \\
u_\theta(y,z) &= \begin{cases} y_\theta & \text{if } z_L \geq 0 \\ (y_\theta + z_\theta) \bmod m_\theta & \text{if } z_L < 0 \end{cases} \qquad (19) \\
v_L(y,z) &= |z_L| \\
v_\theta(y,z) &= \begin{cases} z_\theta & \text{if } z_L \geq 0 \\ (-z_\theta) \bmod m_\theta & \text{if } z_L < 0 \end{cases}
\end{aligned}$$

There are no preconditions for applying the commutativity co-transformation. The postcondition guaranteed as a result of applying it is $v_L(y,z) \geq 0 \wedge f(y,z) = f(u(y,z), v(y,z))$. Note that $v_\theta(y,z) = z_\theta$ because of the properties of modulo two arithmetic. From (19), it is easy to show that $f(y,z) = f(u(y,z), v(y,z))$ because

$$\begin{aligned}
u(y,z) &= \begin{cases} y & \text{if } z_L \geq 0 \\ x & \text{if } z_L < 0 \end{cases} \\
v(y,z) &= \begin{cases} z & \text{if } z_L \geq 0 \\ \tilde{z} & \text{if } z_L < 0 \end{cases}
\end{aligned}$$

This co-transformation is only applied once. The effect is to insure that s_B and d_B are only computed for positive arguments, effectively reducing the implementation cost by one half. To those familiar with previous real logarithmic number system implementations, this derivation may seem a bit tortured, but it will be useful in section 10 when these ideas are generalized to complex values. In a real valued implementation, the determination of the sign, $f_\theta(u(y,z), v(y,z))$ can be simplified considerably.

4. Co-transformation Derived from a Truncated Series

As the bibliography of [2] shows, the following series has been given in the literature several times for real Z_L:

$$\begin{aligned}
D_b(Z_L) &= \log_b |Z_L| - \log_b(\log_b e) + \frac{Z_L}{2} + \frac{Z_L^2 \cdot \ln b}{24} \\
&\quad - \frac{Z_L^4 \cdot (\ln b)^3}{2880} + \frac{Z_L^6 \cdot (\ln b)^5}{181440} - \cdots. \qquad (20)
\end{aligned}$$

However, we will use (20) in a novel way to derive an approximate co-transformation that converts certain logarithmic subtraction (d_B, which is costly for interpolation near the singularity) to logarithmic addition (s_B, which is much cheaper for interpolation). For Z_L near zero [11, 22],

$$D_b(Z_L) \approx \log_b |Z_L| - \log_b(\log_b e) + \frac{Z_L}{2}, \qquad (21)$$

and the error is bounded by $Z_L^2 \cdot \ln b/24$. As described in [1], the required accuracy (due to unavoidable catastrophic cancellation) diminishes as Z_L approaches zero:

$$E_d(Z_L) \approx C/Z_L, \qquad (22)$$

where C is a constant which most naturally would be $\Delta_L \cdot \log_b e$. By solving for $(Z_L^2 \cdot \ln b)/24 < (\Delta_L \cdot \log_b e)/Z_L$, we find that when $Z_L < \sqrt[3]{24 \cdot (\log_b e)^2 \cdot \Delta_L}$, the truncated series produces less error than is inherent in the subtraction being implemented by this approximation. For example, for 23 bits of precision, $Z_L < 0.0181255$.

The constant $-\log_b(\log_b e)$ and the linear term, $Z_L/2$, are trivial to implement. The only difficulty is $\log_b(Z_L)$. Let $g = \lfloor -\log_2(24 \cdot (\log_b e)^2 \cdot \Delta_L)/3 \rfloor$ be the number of bits of Z_L after the binary point that are zero when this truncated series is used. Suppose there are $2n$ additional bits in the fixed point Z_L. It is then possible to break Z_L apart into two n bit integers, z_1 and z_2, such that $Z_L = 2^{-g-n} \cdot z_1 + 2^{-g-2n} \cdot z_2$ and $\Delta_L = 2^{-g-2n}$. For example, with 23 bit precision, $g = 5$ and $n = 9$. Applying (5), we have:

$$\begin{aligned}
\log_b(Z) &= \log_b(2^{-g-n} \cdot z_1 + 2^{-g-2n} \cdot z_2) \\
&= \log_b(2^{-g-2n} \cdot z_2) + \qquad (23) \\
&\quad S_b(\log_b(2^{-g-n} \cdot z_1) - \log_b(2^{-g-2n} \cdot z_2)) \\
&= -\log_b(2^{g+2n}) + \log_b(z_2) \\
&\quad + S_b(\log_b(z_1) - \log_b(z_2) + \log_b(2^n)).
\end{aligned}$$

Since n is a small integer, the size of the lookup table for $\log_b(z_1)$ and $\log_b(z_2)$ is affordable.

The real analytical background above justifies the co-transformation described below which assumes the quantized addition logarithm, s_B, is used rather than the exact S_b. Furthermore, it assumes that the commutativity co-transformation described in (19) has already been applied so that $z_L > 0$. When $z_L \cdot \Delta_L < 2^{-g}$ (equivalently, $z_L < 2^{2n}$), $z_L = 2^n \cdot z_1 + z_2$. From this, we can define the co-transformation:

$$
\begin{aligned}
u_L(y,z) &= \begin{cases} y_L & \text{if } z_\theta = 0 \\ & \quad \vee z_L \geq 2^{2n} \\ c_2 & \\ + o_L(\lceil \log_B(z_2) \rceil) & \text{if } z_\theta = 1 \\ + \lceil \frac{z}{2} \rceil + y_L & \quad \wedge z_L < 2^{2n} \end{cases} \\
u_\theta(y,z) &= y_\theta \qquad\qquad\qquad\qquad\qquad (24) \\
v_L(y,z) &= \begin{cases} z_L & \text{if } z_\theta = 0 \\ & \quad \vee z_L \geq 2^{2n} \\ o_L(\lceil \log_B(z_1) \rceil) & \\ - o_L(\lceil \log_B(z_2) \rceil) & \text{if } z_\theta = 1 \\ + c_1 - c_2 & \quad \wedge z_L < 2^{2n} \end{cases} \\
v_\theta(y,z) &= \begin{cases} z_\theta & \text{if } z_\theta = 0 \vee z_L \geq 2^{2n} \\ 0 & \text{if } z_\theta = 1 \wedge z_L < 2^{2n} \end{cases} .
\end{aligned}
$$

where

$$
\begin{aligned}
c_1 &= -\lfloor \log_B(2^{g+n}) - \log_B(\log_B e) \rfloor, \\
c_2 &= -\lfloor \log_B(2^{g+2n}) - \log_B(\log_B e) \rfloor,
\end{aligned}
$$

and so $c_1 - c_2 = \lfloor \log_B(2^n) \rfloor = n \cdot 2^{g+2n}$ and $c_2 = (g + 2^n) \cdot 2^{g+2n} - \lfloor \log_B(\log_B e) \rfloor$ for $b = 2$. Note that (24) insures that $v_L(y,z) > 0$ for $z_L > 0$. The precondition for applying this series co-transformation is that the commutativity co-transformation has already been applied, i.e., $z_L \geq 0$. The postcondition that is guaranteed as a result of applying it under these circumstances is $(v_L(y,z) \geq 2^{2n} \vee (v_\theta(y,z) = 0 \wedge v_L(y,z) \geq 0)) \wedge f(y,z) \approx f(u(y,z), v(y,z))$.

5. Implementation of Series Co-Transformation

The terms $o_L(\lfloor \log_B(z_1) \rfloor)$ and $o_L(\lfloor \log_B(z_2) \rfloor)$ in (24) can be implemented with one small (2^n word) ROM containing $-m_L$ as an approximation for $-\infty$ in its first word. Such an approach assumes that two or more clock cycles may be spent computing $v_L(y,z)$, as would be acceptable in a software implementation. An advantage that (24) has for software implementation is that, unlike hardware partitioning schemes [16, 5], the point of separation between z_1 and z_2 is fixed, and therefore easier to implement on a processor that lacks normalization hardware. The address calculation for a fully partitioned d_B table involves normalization of z_L, as does Paliouras' approach [24]. Although some high performance CISC processors have instructions that find the position of the leading bit in a word, on many processors that lack floating point this can only be accomplished by a loop that shifts z_L while maintaining a count. There is often a penalty associated with loops on such processors. For these processors, co-transformation software may be faster than full partitioning of d_B since the table addresses can be computed without a loop.

A real logarithmic arithmetic software package [10] that utilizes this co-transformation was implemented in 80x86 assembly language for 23 bit precision ($g = 6, n = 9$).

A slight variation of (24) allows for a hardware implementation [5] that computes $v_L(y,z)$ in one clock cycle by storing $o_L(\lfloor \log_B(z_1) \rfloor) + c_1$ and $o_L(\lfloor \log_B(z_2) \rfloor) + c_2$ in two separate ROMs.

6. Algebraic Co-transformation

There is an alternative co-transformation that achieves a similar effect to the one described in the previous section, but which has many additional desirable properties. Rather than being based on analysis of a series expansion as in the last section, this alternative co-transformation is derived from simple algebra.

If $Z_L > 0$ is the fixed point argument to the real S_b or D_b functions, there are many ways we can select two positive fixed point numbers, Z_1 and Z_2 such that $Z_L = Z_1 + Z_2$. Equivalently, there are corresponding ways to select the integers, z_1 and z_2, such that $z_L = z_1 + z_2$. Unlike the previous co-transformation (24), the co-transformation derived below does not depend on any particular choice of Z_1 and Z_2.

$$
\begin{aligned}
D_b(Z_L) &= \log_b |1 - b^{Z_L}| \\
&= \log_b |1 - b^{Z_1 + Z_2}| \\
&= \log_b |1 - b^{Z_2} + (b^{Z_2} - b^{Z_1} \cdot b^{Z_2})| \\
&= \log_b |1 - b^{Z_2} + b^{Z_2} \cdot (1 - b^{Z_1})| \\
&= \log_b |1 - b^{Z_2} + b^{Z_2} \cdot b^{\log_b (1 - b^{Z_1})}| \\
&= \log_b |1 - b^{Z_2} + b^{Z_2} \cdot b^{D_b(Z_1)}| \\
&= \log_b |1 - b^{Z_2} + b^{Z_2 + D_b(Z_1)}| \\
&= \log_b \left| 1 - b^{Z_2} + \frac{1 - b^{Z_2}}{1 - b^{Z_2}} \cdot b^{Z_2 + D_b(Z_1)} \right| \\
&= \log_b \left| (1 - b^{Z_2}) \cdot (1 + \frac{b^{Z_2 + D_b(Z_1)}}{1 - b^{Z_2}}) \right| \\
&= \log_b \left| b^{\log_b |1 - b^{Z_2}|} \cdot (1 + \frac{b^{Z_2 + D_b(Z_1)}}{b^{\log_b |1 - b^{Z_2}|}}) \right|
\end{aligned}
$$

$$\begin{aligned}
&= \log_b \left| b^{D_b(Z_2)} \cdot (1 + \frac{b^{Z_2+D_b(Z_1)}}{b^{D_b(Z_2)}}) \right| \\
&= \log_b |b^{D_b(Z_2)} \cdot (1 + b^{Z_2+D_b(Z_1)-D_b(Z_2)})| \\
&= \log_b |b^{D_b(Z_2)}| \\
&\quad + \log_b (1 + b^{Z_2+D_b(Z_1)-D_b(Z_2)}) \\
&= D_b(Z_2) + S_b(Z_2 + D_b(Z_1) - D_b(Z_2))
\end{aligned}$$

As a notational convenience, we will define an auxiliary function,

$$H(P,Q,R) = Q + S_b(R + P - Q), \qquad (25)$$

and so,

$$D_b(Z_L) = H(D_b(Z_1), D_b(Z_2), Z_2) \qquad (26)$$

is an identity that holds when H is computed *exactly.* Similarly, we define a quantized auxiliary function,

$$h(p,q,r) = q + s_B(r + p - q), \qquad (27)$$

and so,

$$d_B(z) \approx h(d_B(z_1), d_B(z_2), z_2). \qquad (28)$$

Analogously to (15), we can define the error in (28) as $e_h(z_L)$. Since $d_B(z_1)$ and $d_B(z_2)$ are implemented by table lookup, $0 \le e_d(z_1) \le 1$ and $0 \le e_d(z_2) \le 1$. Since $0 \le S'_B \le 1$,

$$\begin{aligned}
e_h(z_L) &= h(d_B(z_1), d_B(z_2), z_2) - \lfloor D_b(z_L \cdot \Delta_L) \rfloor \\
&\le e_d(z_2) + S'_B \cdot (e_d(z_1) - e_d(z_2)) \\
&\quad + e_s(z_2 + d_B(z_1) - d_B(z_2)) \\
&\le 1 + e_s(z_2 + d_B(z_1) - d_B(z_2)).
\end{aligned}$$

Using (28) to approximate d_B introduces no more than one additional machine unit of relative error (perceived by the end user as a relative error of $B - 1$) beyond whatever error the s_B approximation method introduced. Therefore, (28) is a much more accurate approximation of the subtraction logarithm than (21) in most cases.

There are many possible variations of co-transformations that can be derived from (28). For example, the one which is most analogous to (24) is:

$$\begin{aligned}
u_L(y,z) &= \begin{cases} y_L & \text{if } z_\theta = 0 \vee z_L \ge 2^{2n} \\ y_L + d_B(z_2) & \text{if } z_\theta = 1 \wedge z_L < 2^{2n} \end{cases} \\
u_\theta(y,z) &= y_\theta \qquad (29) \\
v_L(y,z) &= \begin{cases} z_L & \text{if } z_\theta = 0 \vee z_L \ge 2^{2n} \\ z_2 + d_B(z_1) & \text{if } z_\theta = 1 \wedge z_L < 2^{2n} \\ \quad - d_B(z_2) & \end{cases} \\
v_\theta(y,z) &= \begin{cases} z_\theta & \text{if } z_\theta = 0 \vee z_L \ge 2^{2n} \\ 0 & \text{if } z_\theta = 1 \wedge z_L < 2^{2n} \end{cases}.
\end{aligned}$$

The choice of n in (29) is arbitrary. The hardware is similar to that in the last section, except there is no need to add $z/2$. The preconditions and postconditions for this form (29) of the algebraic co-transformation are the same as for the series co-transformation (24).

There is no limitation on the size of Z_L in (26), and so there is no limitation that restricts the number of bits in z_1 or z_2, which means z_1 and z_2 could comprise all of z. So, (28) may be used to derive a simpler co-transformation that converts every logarithmic subtraction into a logarithmic addition:

$$\begin{aligned}
u_L(y,z) &= \begin{cases} y_L & \text{if } z_\theta = 0 \\ y_L + d_B(z_2) & \text{if } z_\theta = 1 \end{cases} \\
u_\theta(y,z) &= y_\theta \qquad (30) \\
v_L(y,z) &= \begin{cases} z_L & \text{if } z_\theta = 0 \\ z_2 + d_B(z_1) - d_B(z_2) & \text{if } z_\theta = 1 \end{cases} \\
v_\theta(y,z) &= 0,
\end{aligned}$$

There are no preconditions for this alternate form (30) of the algebraic co-transformation. The postcondition is $v_\theta(y,z) = 0 \wedge f(y,z) \approx f(u(y,z), v(y,z))$. (30) eliminates the need to interpolate for d_B at all if the table sizes for $d_B(z_1)$ and $d_B(z_2)$ are considered acceptable.

7. Iterative Co-transformations

The function H can be applied iteratively to reduce these table sizes. For example, if $Z_L = Z_1 + Z_2 + Z_3$,

$$\begin{aligned}
D_b(Z_L) &= H(D_b(Z_1 + Z_2), D_b(Z_3), Z_3) \qquad (31) \\
&= H(H(D_b(Z_1), D_b(Z_2), Z_2), D_b(Z_3), Z_3)
\end{aligned}$$

Software [10] that approximates $D_b(Z_L)$ using (32) was implemented for 23 bits of precision, where there are ten bits in Z_1, and nine bits each in Z_2 and Z_3. These 28 bits are sufficient to approximate the entire range of D_b within the 32 bit word because of the essential zero concept [32].

When (26) is carried to its logical conclusion, each bit of Z_L could be processed separately in an iterative series of co-transformations,

$$U_k(Y,Z) = \begin{cases} H(Y, D_b(2^{k-n}), 2^{k-n}) & \text{if } Z > 2^{k-n} \\ Y & \text{if } Z \le 2^{k-n} \end{cases} \qquad (32)$$

and

$$V_k(Y,Z) = \begin{cases} Z - 2^{k-n} & \text{if } Z > 2^{k-n} \\ Z & \text{if } Z \le 2^{k-n} \end{cases} \qquad (33)$$

where now n is the number of fractional bits, $F(Y,Z) = Z - S_b(Y)$, $Z_0 = Z_L$ and $Y_0 = -\infty$. When after j iterations $Z_j = 0$ we have $Y_j \approx D_b(Z_L)$.

8. Addition Co-transformations

There are relationships analogous to (26) that describe how to convert a particular logarithmic addition into another, possibly simpler, logarithmic addition:

$$S_b(Z_L) = H(S_b(Z_1), D_b(Z_2), Z_2) \tag{34}$$

and

$$S_b(Z_L) = H(S_b^{-1}(Z_1), S_b(Z_2), Z_2) \tag{35}$$

where $S_b^{-1}(Z_1) = D_b(Z_1)$ because Z_1 is real.

Assuming the commutativity co-transformation (19) and the algebraic co-transformation (30) have already been applied, a third co-transformation, derived from (34), can, under certain circumstances, eliminate the need for any interpolation. The precondition for applying the following addition co-transformation is that the commutativity and algebraic co-transformations have already been applied, i.e., $z_L \geq 0 \wedge z_\theta = 0$:

$$\begin{aligned} u_L(y,z) &= y_L + d_B(z_2) \\ u_\theta(y,z) &= y_\theta \\ v_L(y,z) &= z_2 + s_B(z_1) - d_B(z_2) \\ v_\theta(y,z) &= 0 \end{aligned} \tag{36}$$

where z_1 contains the high order bits of z_L and z_2 contains the low order bits of z_L. The postcondition that is guaranteed as a result of applying the addition co-transformation given the above preconditions is $v_L(y,z) \gg 0 \wedge f(y,z) \approx f(u(y,z), v(y,z))$. For $b = 2$, $v_L(y,z) \cdot \Delta_L$ is at least equal to the number of fractional bits[1] in Z_1, which means (13) can be computed more easily. In particular, for 23 bits of precision, the s_B in (13) can be approximated without interpolation using the same table that gives $s_B(z_1)$ when z_2 contains twelve bits. If z_2 contains more bits[2], interpolation would be required, but would use a much smaller multiplier than if (36) had not been applied.

9. Complex Logarithmic Number System

There is a natural generalization of the real logarithmic number system that allows representation of complex values. Instead of allocating a single bit for the sign, one can allocate an appropriate number of bits to represent an angle in the complex plane. To convert a complex value, X, to a quantized complex logarithmic representation, $x = x_L + x_\theta \cdot i$, composed of the quantized logarithm of the length of a vector in the complex plane, x_L, and the quantized angle of that vector, x_θ:

$$x_\theta = \left\lfloor \frac{\arctan(\Re[X], \Im[X])}{2\pi} \cdot m_\theta \right\rfloor \bmod m_\theta \tag{37}$$

and

$$\begin{aligned} x_L &= o_L\left(\left\lfloor \frac{0.5 \cdot \log_b(\Re[X]^2 + \Im[X]^2)}{\Delta_L} \right\rfloor \right) \\ &= o_L(\lfloor \log_B |X| \rfloor) \end{aligned} \tag{38}$$

where $i = \sqrt{-1}$ and $\arctan(\Re[X], \Im[X])$ returns an angle between $-\pi$ and π. This four quadrant arctan function handles all cases such as $\Re[X] = 0$ or $\Im[X] = 0$. The constant m_θ determines the precision with which the angle is represented. Obviously, when $m_\theta = 2$, this is equivalent to (7) and (9), but to represent imaginary numbers, $m_\theta \bmod 4 = 0$. Multiplication and division, as in (10), generalize without difficulty. The conjugate can be formed by negating $x_\theta \bmod m_\theta$.

In 1895, Mehmke [18] described the idea of complex addition logarithms,

$$\begin{aligned} \Re[S_b(Z_C)] &= 0.5 \cdot \log_b(1 + b^{Z_L}\cos(Z_\theta) + b^{2 \cdot Z_L}) \\ \Im[S_b(Z_C)] &= \arctan(1 + b^{Z_L}\cos(Z_\theta), b^{Z_L}\sin(Z_\theta)), \end{aligned} \tag{39}$$

where $Z_C = Z_L + i \cdot Z_\theta$, $Z_L = z_L \cdot \Delta_L$ and $Z_\theta = 2\pi \cdot z_\theta / m_L$, but to the authors' knowledge, this complex logarithm number system was never analyzed further and has never been used in a modern implementation until now[3].

Complex relationships, such as

$$S_b(Z_C) = D_b(Z_C + k \cdot \pi i), \tag{40}$$

where k is any odd integer, simplify the description of the addition algorithm, because there is no need to mention the D_b function. Given the complex quantized representations, x and y, of two complex values, X and Y, the addition algorithm is the same as (11), except the definition of (13) and (12) generalize to:

$$f_\theta(y,z) = (y_\theta + \Im[s_B(z)]) \bmod m_\theta \tag{41}$$

$$f_L(y,z) = y_L + \Re[s_B(z)]. \tag{42}$$

or more simply, (41) and (42) can be combined:

$$f(y,z) = y + s_B(z), \tag{43}$$

where $z = z_L + z_\theta i$, and s_B is the quantized complex addition logarithm.

[1] $v_L(y,z) \cdot \Delta_L > Z_2/2 - \log_b(Z_2)$ from (21), and $-\log_2(Z_2)$ is at least the number of fractional bits in Z_1.

[2] So $Z_1 = z_1 \cdot \Delta_L$ would contain fewer fractional bits.

[3] Except for application of polar representation to complex SLI arithmetic [33], which bears some similarity to the complex logarithmic arithmetic described here.

10. Complex Co-Transformation

Prior low precision implementations of the real logarithmic number system have typically used ROM to implement S_b without interpolation. Having both real and imaginary parts of Z_C means the number of ROM address bits required for a non-interpolated complex S_b is almost double what would be required for a non-interpolated real S_b. Such large ROM sizes are unreasonable, even for moderate precision systems, and so interpolation is important. Furthermore, for values that approach $\pm\pi i$, S_b has the same singularity that D_b has for values that approach zero. Therefore, interpolation of S_b in one half of the complex plane is much more difficult than interpolation of the real S_b function, a fact [18] failed to note. A novel co-transformation, similar to the real ones described earlier, is required to make complex interpolation affordable.

The relationship defined in (35) holds for complex Z_C, and so it may be used as the basis for such a complex co-transformation. If we let $Z_1 = Z_L$ and $Z_2 = i \cdot Z_\theta$, we have

$$\begin{aligned} S_b(Z_C) &= H(S_b^{-1}(Z_L), S_b(i \cdot Z_\theta), i \cdot Z_\theta) \\ &= S_b(i \cdot Z_\theta) \\ &\quad + S_b(i \cdot Z_\theta + S_b^{-1}(Z_L) - S_b(i \cdot Z_\theta)) \\ &= S_b(i \cdot Z_\theta) + S_b(T) \qquad (44) \end{aligned}$$

where

$$T = i \cdot Z_\theta + \pi i + D_b(Z_L) - S_b(i \cdot Z_\theta) \qquad (45)$$

because $S_b^{-1}(Z_L) = D_b(Z_L) + \pi i$. The commutativity co-transformation (19) allows us to insure that $Z_L \geq 0$ prior to applying (44), therefore consider how T can be simplified:

$$\begin{aligned} \Re[T] &= \Re[D_b(Z_L)] - \Re[S_b(i \cdot Z_\theta)] \\ &= D_b(Z_L) \qquad (46) \\ &\quad - \frac{\log_b(2) + \log_b(1 + \cos(Z_\theta))}{2}, \\ \Im[T] &= Z_\theta + \pi + \Im[D_b(Z_L)] - \Im[S_b(i \cdot Z_\theta)] \\ &= Z_\theta + 2\pi - \Im[S_b(i \cdot Z_\theta)] \\ &= \frac{Z_\theta}{2}. \end{aligned}$$

The relationship $\Im[D_b(Z_L)] = \pi$ holds only for $Z_L \geq 0$ but the other [18] identity used above, $\Im[S_b(i \cdot Z_\theta)] = Z_\theta/2$, holds for all $Z_\theta \neq k \cdot \pi$. Because $|Z_\theta| \leq \pi$ in the complex logarithm number system, $|\Im[T]| \leq \pi/2$, which has the desired effect of making interpolation of $S_b(T)$ much easier than interpolation of $S_b(Z_C)$ when $\pi/2 < |Z_\theta| \leq \pi$. Therefore, the following is equivalent to (44) when $Z_L \geq 0$:

$$\begin{aligned} \Re[S_b(Z_C)] &= \frac{\log_b(2) + \log_b(1 + \cos(Z_\theta))}{2} \\ &\quad + \Re[S_b(T)] \\ \Im[S_b(Z_C)] &= \frac{Z_\theta}{2} + \Im[S_b(T)] \qquad (47) \end{aligned}$$

If (44) were applied inappropriately, when $Z_L < 0$, it would have the undesirable effect of making $\pi/2 < |\Im[T]| \leq \pi$.

From (47), we can derive a co-transformation that avoids the interpolation difficulties for complex logarithmic addition. Assuming the commutativity co-transformation (19) has already been applied, the following describes the complex addition co-transformation:

$$\begin{aligned} u_L(y,z) &= y_L + \\ &\quad \left\lfloor \frac{\log_B(1 + \cos(z_\theta \cdot 2\pi/m_\theta)) + \log_B(2)}{2} \right\rfloor \\ u_\theta(y,z) &= \left(y_\theta + \left\lfloor \frac{z_\theta}{2} \right\rfloor\right) \bmod m_\theta \qquad (48) \\ v_L(y,z) &= d_B(z_L) - \\ &\quad \left\lfloor \frac{\log_B(1 + \cos(z_\theta \cdot 2\pi/m_\theta)) + \log_B(2)}{2} \right\rfloor \\ v_\theta(y,z) &= \left\lfloor \frac{z_\theta}{2} \right\rfloor. \end{aligned}$$

The precondition for applying the complex co-transformation is that the commutativity co-transformation has already been applied, i.e., $z_L \geq 0$. The postcondition that is guaranteed as a result of applying the complex co-transformation under these circumstances is $|v_\theta(y,z)| \leq m_\theta/4 \wedge f(y,z) \approx f(u(y,z), v(y,z))$.

11. Coleman's Co-transformation

After this paper was submitted, the authors became aware of the work of Coleman [8], who independently discovered a co-transformation that reduces the cost of interpolating the real valued subtraction logarithm. Coleman's technique, unlike those described here and in [5], transforms the problem of computing $D_b(Z_L)$ for real Z_L near zero into interpolation of D_b for an argument further away from zero. All of the co-transformations given here convert cases near the singularity into interpolation of S_b. Coleman's technique is more limited than those given here because a D_b computation is required in every case. For this reason, Coleman's technique would not be as useful for implementing the complex logarithmic number system as the ones described in this paper.

12. Conclusions

We have shown that the well known real logarithmic number system is a special case of the more obscure complex logarithmic number system. Both real and complex systems share the common problem that using interpolation near the singularity (as happens when subtracting nearly equal values) produces more error than is acceptable. We have proposed two co-transformations to eliminate this problem. The first is based on the analysis of the subtraction logarithm, and the second is based on simple algebra. For 23 bits of precision, co-transformation with the real logarithmic number system produces acceptable results using small tables, and is more suitable for software than the partitioning previously disclosed for hardware.

The iterative co-transformation described in section 7 bears some similarity to CORDIC and similar algorithms [9, 19]. Further research into this may be warranted.

Co-transformation with the complex logarithmic number system offers a practical approach to implement this interesting system which has not yet been fully explored. In particular, there is a question on how to represent values near zero, for which there may be analogies to [4]. We hope that future investigation into the complex logarithmic number system will discover how useful it is for DSP applications, such as the FFT, that make extensive use of complex arithmetic. The FFT seems particularly promising since the roots of unity used by this algorithm have exact representations in the complex logarithmic number system, and would not have to be stored in a ROM.

Acknowledgements

The authors wish to thank Somatogen, Inc. for providing support for the writing of this paper. Also, the authors wish to thank Pharlap and Microway.

From 1988 to 1996, two of the authors[4] attempted to market a software package [10] that embodies the real algorithms [5] disclosed here. These authors wish to thank the following people who assisted us in this endeavor: David Bazuki (Knowledge Revolution), Robin Bolz (Microsoft), Walter Bright (Symantec), Tom Brightman (Cyrix), Joel Egberg (Wind River Systems), Nat Goldhaber (Apple), Dan Hammerstrom (ASI), Randy Isaac (IBM - Austin Research Lab), Ray and Sabin Larsen (Altra), David Matula (Texas A and M), Tom Newsom (Hewlett Packard), Everett Roach (Weitek), Jerry Rogers (Cyrix), Larry Silverman (Intel), Earl Swartzlander (Univ. of Texas—Austin), Duncan Terry (Hewlett Packard), Richard Tompane (3DO), Susheela Vasan (3DO), Don Watt (Hewlett Packard) and Gideon Yiguel (Microsoft).

During this time, the authors became aware of the significant contributions [26] made to the field by a competitor, Les Pickett (LogPoint Systems, Inc., `www.logpoint.com`), who developed one of the most widely used commercial applications of logarithmic arithmetic: the cabin air pressure controls for the Boeing 767 and several other aircraft designed since 1977 [25].

[4]Arnold and Winkel

References

[1] M. G. Arnold, T. A. Bailey, J. R. Cowles and J. J. Cupal, "Redundant logarithmic arithmetic," *IEEE Trans. Comput.*, vol. 39, pp. 1077–1886, Aug. 1990.

[2] M. Arnold, T. Bailey, J. Cowles and J. Cupal, "Initializing RAM-based logarithmic processors," *J. VLSI Signal Processing,* vol. 4, pp. 243-252, 1992.

[3] M. Arnold, T. Bailey and J. Cowles, "Comments on 'an architecture for addition and subtraction of long word length numbers in the logarithmic number system'," *IEEE Trans. Comput.*, vol. 41, pp. 786-788, June 1992.

[4] M. G. Arnold, T. A. Bailey, J. R. Cowles and M. D. Winkel, "Applying features of IEEE 754 to sign/logarithm arithmetic," *IEEE Trans. Comput.*, vol. 41, pp. 1040–1050, Aug. 1992.

[5] M. G. Arnold, "Method and Apparatus for Fast Logarithmic Addition and Subtraction", U. S. Patent 5,337,266, Aug. 9, 1994.

[6] M. G. Arnold, T. A. Bailey, J. J. Cupal and M. D. Winkel, "On the Cost Effectiveness of Logarithmic Arithmetic for Back-Propagation Training on SIMD Processors", accepted at 1997 International Conference on Neural Networks, Houston, TX, June 9-12, 1997.

[7] J. L. Barlow and E. H. Bareiss, "On roundoff distribution in floating point and logarithmic arithmetic," *Computing,* vol. 34, pp. 325–364, 1985.

[8] J. N. Coleman, "Simplification of table structure in logarithmic arithmetic," *Electron. Lett.*, vol. 31, pp. 1905–1906, Oct. 1995.

[9] T. C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots," *IBM Jour. Res. Develop.*, pp. 380-388, Jul. 1972.

[10] "FastMath: Software Faster than a Coprocessor," *C User's Journal*, vol. 9, no. 7, p. 12, July 1991.

[11] S. Gundelfinger, "Zur Berechnung der Gaussschen Logarithmen für Kliene Werthe von B resp. zugehörige Werthe von A", *Journal für die reine und angewandte Mathematik*, vol. 124, pp. 87-92, 1902.

[12] W. N. Holmes, "Composite Arithmetic: Proposal for a New Standard," *Computer*, Vol. 30, No. 3, pp. 65-73, Mar. 1997.

[13] T. Kurokawa and T. Mizukoshi, "A fast and simple method for curve drawing—a new approach using logarithmic number systems," *J. of Information Processing*, vol. 14, pp. 144–152, 1991.

[14] D. M. Lewis, " An architecture for addition and subtraction of long word length numbers in the logarithmic number system," *IEEE Trans. Comput.*, pp. 1325–1336, 1990.

[15] D. M. Lewis, "Interleaved memory function interpolators with application to an accurate LNS Arithmetic Unit," *IEEE Trans. Comput.*, vol. 43, no. 8, pp 974-982, Aug. 1994.

[16] D. M. Lewis, "114 MFLOPS logarithmic number system arithmetic unit for DSP applications", *International Solid-State Circuits Conference*, pp. 1547–1553, San Francisco, Feb. 1995.

[17] J. D. Marasa and D. W. Matula, "A simulative study of correlated error in various finite-precision arithmetics," *IEEE Trans. Comput.*, vol. C-22, pp. 587–597, June 1973.

[18] R. Mehmke, "Additionslogarithmen für Complexe Grössen," *Zeitschrift für Mathematik und Physik*, vol. 40, pp 15-30, 1895.

[19] J. M. Muller, "Une Mèthodologie du Calcul Hardware des Fonctions Èlèmentaires", *Mathematical Modeling and Numerical Analysis*, Gauthier-Villars, vol. 20, pp. 667-695, Sep. 1985.

[20] J. M. Muller, A. Tisserand and A. Scherbyna, "Semi-Logarithmic Number System," *Proceedings of the 12th Symposium on Computer Arithmetic*, IEEE Computer Society Press, Bath, England, pp. 201-207, July 19-21, 1995.

[21] R. E. Morley, T. J. Sullivan and G. L. Engel, "VLSI based design of a battery-operated hearing aid," *Southcon/90*, pp. 55–59, Orlando, FL, Mar. 20–22, 1990.

[22] Nell, "Ueber die Interpolationsrechnungen bei grosseren Logarithmentafeln", *Zeitschrift fur Vermessungswesen*, vol. 20, pp. 442-446, 1891.

[23] I. Orginos, V. Paliouras and T. Stouraitis, "Novel algorithm for multi-operand logarithmic number system addition and subtraction using polynomial approximation", *International Symposium on Circuits and Systems*, pp. 1992–1995, Seattle, Apr. 30–May 5, 1995.

[24] V. Paliouras and T. Stouraitis, "Novel algorithm for accurate logarithmic number system subtraction", *International Symposium on Circuits and Systems*, pp. 268–271, Atlanta, May 12–15, 1996.

[25] L. Pickett, Private communication, Nov. 17, 1996.

[26] L. Pickett, "Method and Apparatus for exponential/logarithmic computation", U. S. Patent 5,197,024, Mar. 23, 1993.

[27] T. Stouraitis, "Logarithmic number system theory, analysis, and design," Ph. D. Dissertation, University of Florida, Gainesville, 1986.

[28] S. Shanks, "New Soft Co-Processor for Fujitsu's High-Performance Embedded Controllers", *Embedded Control Design*, vol. 1, p. 10, Fall 1994.

[29] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Trans. Comput.*, vol. C-24, pp. 1238–1242, Dec 1975.

[30] E. E. Swartzlander, et. al., "Arithmetic for ultra-high speed tomography," *IEEE Trans. Comput.*, vol. C-29, pp. 341–353, 1980.

[31] E. E. Swartzlander, et. al., "Sign/logarithm arithmetic for FFT Implementation," *IEEE Trans. Comput.*, vol. C-32, pp. 526–534, 1983.

[32] F. J. Taylor, R. Gill, J. Joseph and J. Radke, "A 20 Bit logarithmic number system processor," *IEEE Trans. Comput.*, vol. C-37, pp. 190–199, 1988.

[33] P. R. Turner, "Complex SLI Arithmetic: Representation, Algorithms and Anaylsis," *Proceedings of the 11th Symposium on Computer Arithmetic*, Windsor, Ontario, IEEE Computer Society Press, pp. 18-25, June 29-July 4, 1993.