Towards a Quaternion Complex Logarithmic Number System

Mark G. Arnold*, John Cowles†, Vassilis Paliouras‡, Ioannis Kouretas‡
*Lehigh University
marnold@cse.lehigh.edu
†University of Wyoming
cowles@cs.uwyo.edu
‡University of Patras
paliuras@ece.upatras.gr

***Abstract*—The well-known generalization of real to complex arithmetic (two reals) extends further to more obscure quaternion arithmetic (four reals), which has applications in signal processing, aerospace, graphics and virtual reality. Quaternion multiplication implements 3D rotation, but is expensive (usually 16 floating-point multiplications and 12 additions). This paper proposes an alternative quaternion representation using logarithms to reduce multiplication cost. The real Logarithmic Number System (LNS) allows fast and inexpensive multiplication and division in embedded and FPGA-based systems. Recent advances in the Complex LNS (CLNS) [5] have made fast log-polar complex representation affordable. Although the quaternion logarithm function is also well-defined, it is not useful to simplify multiplication (in the same way real and complex logarithms are) because quaternion multiplication is not commutative but quaternion addition is. To overcome this, we propose a novel Quaternion Complex (QCLNS) representation using a pair of CLNS numbers. This representation implements quaternion multiplication using only the theoretical minimum [11], [15] of 8 LNS multipliers (i.e., fixed-point adders) and two CLNS adders. Because CLNS numbers are more compact than ordinary rectangular complex representation, single-precision QCLNS occupies 10.9 percent less memory than conventional quaternion representation. Extrapolating conventional LNS and floating-point synthesis data from Fu et al. [12], QCLNS saves on average 10 percent of FPGA resources for precisions between 13 and 45 bits.**

***Keywords*-Complex number, quaternion, Logarithmic Number System, hardware function evaluation, rotation, FPGA.**

I. Introduction

Both logarithms and quaternions are venerable mathematical concepts; when discovered, each revolutionized and ruled its century's science and engineering, both from a theoretical and practical (manual-computational) standpoint. Seventeenth-century logarithms paved the way for calculus and logarithm-based slide rules gave three centuries of engineers hand-held computational power. Nineteenth-century quaternions opened mathematicians' eyes to the possibility of non-commutative algebras, and allowed easier computation for contemporary problems like Maxwell's equations and the Lorentz transform [19]. In the twentieth century, the computational uses of both logarithms and quaternions were diminished by newer innovations. In the case of logarithms, floating-point computers and calculators eliminated the use of most slide rules; in the case of quaternions, vector and matrix calculation seemed more convenient than old-fashioned quaternions. At the dawn of the twenty-first century, however, the computational use (in digital computers) of both logarithms and quaternions experienced a bit of a revival. The Logarithmic Number System (LNS) [30], described in Section II, offers advantages over floating point for certain embedded and FPGA systems [12]; floating-point quaternions, described in Section III, have proven quite useful in areas such as visualization [13], color-image processing [27], virtual reality [17], bioinformatics (both as a representation of DNA base-pairs [18] and to describe the rotation of DNA molecules [21]), aerospace control systems [16], among other topics. This paper explores the possibility of combining quaternions with LNS. In light of the non-commutative nature of quaternions, a combination of LNS and quaternions that is practical from a computational standpoint requires a novel approach like the one described in this paper.

The recent resurgence of interest in quaternions is mostly due to the fact that quaternions are convenient for 3D-rotation. The need for such rotations occur in a variety of applications, such as graphics (e.g., human limb motion animation [8]), virtual reality (e.g., smooth camera tracking [13]), robotics (e.g., manipulator-sensor positioning [7]) and control systems (e.g., spacecraft altitude [16]). Unlike rotation using Euler angles, quaternion rotation avoids singularities [17]. For example, although gyroscopes cannot completely guarantee to avoid the danger of physical *gimbal lock*, in which two measurements that are supposed to be orthogonal actually are not, modeling with quaternions allows a simulation to avoid virtual gimbal lock [13]. In addition to numerical stability, quaternions (consisting of four elements) are more storage-efficient than equivalent rotation matrices (consisting of nine elements). Because quaternions are a double cover of the rotational matrix group, SO(3), both a quaternion, Q, and its conjugate, Q^*, represent the same rotation [1]. If Q is chosen to rotate half of the desired angle, the product, $Q\mathbf{V}Q^*$, is the 3D-vector output (i.e., quaternion with no real component) for the rotation of the 3D-vector input, $\mathbf{V}$, by the desired angle, θ about a desired axis $\mathbf{U}$

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.14

specified by $Q = \cos(\theta/2) + \sin(\theta/2)\mathbf{U}$. These advantages have motivated designers of graphics libraries, like DirectX [24], to provide built-in quaternion capability.

Section II reviews prior LNS literature, including its generalization (CLNS) for complex values. Section III outlines the basic properties of the quaternions, including the conventional way multiplication is implemented using floating-point arithmetic and the well-established mathematical definition of a quaternion logarithm. Section IV explains why this definition is not suitable as the foundation for a practical quaternion LNS. Section V introduces a novel number system (QCLNS) that overcomes this problem by representing a quaternion using a pair of CLNS representations. Section VI describes a small simulation that suggests that the accuracy of QCLNS multiplication is acceptable. Section VII shows how the QCLNS multiplier hardware may be optimized, and extrapolates proprietary synthesis results for the classical LNS reported in literature [12] to compare the sizes of quaternion multipliers using classical-LNS, floating-point and the proposed-QCLNS approaches. Section VIII concludes that, although preliminary, the data reported here suggest QCLNS may offer some combination of smaller bit-size representations (with smaller busses, memories and associated-power consumption), smaller area and/or more accurate results.

II. Review of Prior Real and Complex LNS

The classical Logarithmic Number System (LNS) represents a non-zero real value A using a quantized base-β logarithm, $a_0 = \log_\beta |A|$ and a sign bit, a_1. In this paper, upper-case variables will be used for the values perceived by the end-user; lower-case variables will be used for the logarithmic representation. In particular, the upper-case "A" will be used for a real value, and the lower-case letter "a" will be used for its LNS representation. A similar upper/lower-case distinction will be used for complex ("$\bar{X}$" and "$\bar{x}$") and quaternion ("Q" and "q") values and representations. The bar ("$\bar{X}$" or "$\bar{x}$") is used in this paper to avoid ambiguity only when dealing with an entire complex value or representation; otherwise subscripted variable names identify their data type as described above (i.e., X_0 without the bar is the real component of the complex value $\bar{X}$; $\bar{X}_0$ is the first element of an array of complex values; its real component is $X_{0,0}$).

The quantization of a_0 is often considered as a fixed-point format for a base-two logarithm with F fractional bits and K integer bits for rough equivalence to $K + F + 1$-bit floating point (e.g., $K = 8$ and $F = 23$ for single precision); however, an equivalent quantization for a_0 is as a $(K + F)$-bit integer using $\beta = 2^{2^{-F}}$. Some applications need an exact representation of zero, which can be indicated by an additional bit, a_z. The value represented by $a = (a_z, a_0, a_1)$ is $a_z \beta^{a_0} (-1)^{a_1}$. For storage efficiency, the zero information could also be encoded in the a_0 field as $(\zeta(a_z, a_0), a_1)$, where $\zeta(a_z, a_0) = 2^{-K-F}$ if a_z is true and otherwise passes a_0 through unmodified; ζ^{-1} must be performed to recover a_z when a value is retrieved from memory into the ALU. Some applications do not require a_z; for simplicity in this paper, we will mostly ignore a_z and ζ in the mathematical description of LNS.

The advantage of the classical LNS is that roots, powers, division and multiplication are easy, using properties like $\log_\beta |AB| = \log_\beta |A| + \log_\beta |B| = a_0 + b_0$. In fact, it is reasonable to say the classical LNS definition is structured around this multiplicative property, at the expense of other operations, like addition and subtraction. To make classical LNS a general-purpose number system, which performs all arithmetic operations on inputs and outputs in the same LNS format, LNS addition and subtraction algorithms derive from the following property: $A+B = A(1+Z)$ where $Z = B/A$. Classical LNS implements this with a fixed-point adder (for the multiply) and a fixed-point subtractor (for the divide) together with a lookup (for the increment) from one of two tables ($s_\beta(z) = \log_\beta(1 + \beta^z)$ or $d_\beta(z) = \log_\beta |1 - \beta^z|$). When the signs are the **s**ame ($a_1 = b_1$), the result can be computed as $\log_\beta |A + B| = \log_\beta |A| + s_\beta(\log_\beta |B| - \log_\beta |A|) = a_0 + s_\beta(b_0 - a_0)$. When the signs are **d**ifferent ($a_1 \neq b_1$), the result can be computed as $\log_\beta |X + Y| = \log_\beta |A| + d_\beta(\log_\beta |B| - \log_\beta |A|) = a_0 + d_\beta(b_0 - a_0)$. The computation of s_β and especially d_β is difficult to implement in hardware. A large body of LNS literature [32] is devoted to a variety of ways to simplify the computation of s_β and d_β using techniques such as table reduction [30], [29], CORDIC [20] and similar algorithms [6], multipartite [9], interpolation [12], cotransformation higher-order tables [9].

In contrast to the extensive classical LNS literature in which the system is defined around a multiplicative basis, two alternative LNS systems for representing real values have been proposed that include an additive component, whose purpose is to simplify addition and/or subtraction. The first of these is known as the Dual Redundant LNS (DRLNS), whose purpose is to defer the difficult LNS operation of subtraction until the end of the computation [2]. Instead of using a logarithm and a sign bit, DRLNS represents a signed real A with two logarithms: a_P and a_N. Since DRLNS is a redundant system, many possible choices for these logarithms represent the same value $A = \beta^{a_P} - \beta^{a_N}$. Addition in this system is simplified because instead of needing s_β and d_β, DRLNS needs only two of the simplier s_β units. DRLNS Multiplication is more complicated (in a way similar to complex multiplication): $AB = (\beta^{a_P} - \beta^{a_N})(\beta^{b_P} - \beta^{b_N}) = (\beta^{a_P+b_P} + \beta^{a_N+b_N}) - (\beta^{a_P+b_N} + \beta^{a_N+b_P})$. This involves two s_β units, computing $a_P + b_P + s_\beta(a_N + b_N - (a_P + b_P))$ and $a_P + b_N + s_\beta(a_N + b_P - (a_P + b_N))$. An even more unusual number system of this additive kind is the Multi-Dimensional LNS (MDLNS) which uses multiple bases (typically two and β not a power of two), for example $A = \pm 2^{a_{00}} \beta^{a_{01}} \pm 2^{a_{10}} \beta^{a_{11}}$. The addition and multiplication algorithms for MDLNS are

quite involved [25].

The classical multiplicative LNS is not limited only to representing real values. Consider the conventional rectangular definition for complex values, $\bar{X} = X_0 + X_1\mathbf{i}$ and $\bar{Y} = Y_0 + Y_1\mathbf{i}$. Conventional complex multiplication involves four real multiplications and two real additions:

$$\begin{aligned} \bar{X}\bar{Y} &= (X_0 + X_1\mathbf{i})(Y_0 + Y_1\mathbf{i}) \\ &= (X_0Y_0 - X_1Y_1) + (X_0Y_1 + X_1Y_0)\mathbf{i}. \end{aligned} \quad (1)$$

If this were implemented in the fastest way possible using conventional real LNS, (1) would require two sets of s_β/d_β tables, and eight fixed-point adders. Instead, the Complex LNS (CLNS) is a generalization which uses a log-polar format in order to simplify such multiplication [23], [3], [20], [31]. CLNS converts the usual rectangular format of a complex value into $\bar{X} = \beta^{x_0}\text{cis}(x_1)$, where $\text{cis}(x_1) = \cos(\alpha x_1) + \mathbf{i}\sin(\alpha x_1)$ and the scaling factor $\alpha = \pi 2^{-F-2}$ makes the modulo 2^{F+3} action of the signed $(F+3)$-bit binary word for x_1 fit the behavior of the circular functions (i.e., $-\pi \le \alpha x_1 < \pi$). The angle only needs $F+3$ bits compared to $F+K+1$ bits for the log-magnitude because $\text{cis}(x_1) = \text{cis}(x_1 \pm 2\pi/\alpha)$, which will be useful later in this paper. When $\alpha = 1$ and $\beta = e$, the inverse of this is equivalent to the mathematical definition $\bar{x} = \log(\bar{X})$:

$$\begin{aligned} x_0 &= \log\left(\sqrt{X_0^2 + X_1^2}\right) \\ x_1 &= \arctan(X_0, X_1) \end{aligned} \quad (2)$$

where $\arctan(x, y)$ is the four-quadrant function, like `atan2(y,x)` in many programming languages. Note that if x_z is not used (or if it is compressed with $\zeta(x_z, x_0)$), the wordsize is $K-1$ bits smaller for the CLNS representation $(K+2F+3)$ versus the conventional real and imaginary pair $(2K+2F+2)$. This same savings with CLNS holds whether the conventional rectangular complex representation uses LNS or floating point. Instead of conventional rectangular complex multiplication (1), CLNS computes the product using two inexpensive parallel fixed-point adders: $\bar{X}\bar{Y} = \beta^{x_0+y_0}\text{cis}(x_1 + y_1)$. The representation of the conjugate is the conjugate of the representation. Conventional LNS can be considered a subset of CLNS—restricting x_1 and y_1 to a single bit makes the system isomorphic to conventional signed real LNS. Direct addition and subtraction are possible in CLNS using a version of $s_{\alpha,\beta}$ that outputs a complex result given a complex argument $\bar{z} = z_0 + \mathbf{i}z_1$:

$$\begin{aligned} s_{\alpha,\beta}(\bar{z}) &= \log_\beta(1 + \beta^{z_0}\text{cis}(z_1)) \quad (3) \\ &= \log_\beta(1 + \cos(\alpha z_1)\beta^{z_0} + \mathbf{i}\sin(\alpha z_1)\beta^{z_0}) \\ &= \frac{\log_\beta(1 + 2\cos(\alpha z_1)\beta^{z_0} + \beta^{2z_0})}{2} \quad (4) \\ &\quad + \mathbf{i}\frac{\arctan(1 + \cos(\alpha z_1)\beta^{z_0}, \sin(\alpha z_1)\beta^{z_0})}{\alpha}, \end{aligned}$$

Since the representation of the negative is formed by adding $\alpha\pi\mathbf{i}$, a separate $d_{\alpha,\beta}$ is not required. Unlike the real s_β, which is only a one-dimensional table (function of z_0), the complex $s_{\alpha,\beta}$ has much larger storage requirements since it is a two-dimensional table (function of z_0 and z_1). Methods to reduce this table size include cotransformation [3], CORDIC [20] and content-addressable memory [31]. More recently a novel, much more economical, algorithm for CLNS has been proposed that uses a conventional real LNS ALU (that outputs real s_β and d_β) to compute the complex $s_{\alpha,\beta}$ function using modest additional hardware for the log-trig functions. [4], [5]. This approach was implemented by a downloadable state-of-the-art function-approximation tool called FloPoCo [10], which produced synthesizable VHDL. This paper will build upon this CLNS foundation in Section V.

	i	**j**	**k**
i	-1	**k**	-**j**
j	-**k**	-1	**i**
k	**j**	-**i**	-1

Table I
SPECIAL QUATERNION PRODUCTS (LEFT COLUMN IS LEFT OPERAND; TOP ROW IS RIGHT OPERAND).

III. QUATERNIONS

Although less well-known than the one-dimensional real and two-dimensional complex number systems, the four-dimensional quaternion system finds many applications in areas like signal processing [27], avionics/aerospace [17] and virtual reality [13]. One advantage of a quaternion is that a single quaternion, consisting of four real numbers, encodes a 3D rotation that would otherwise require multiplication by a matrix containing nine numbers. Hamilton famously discovered quaternions while walking near a bridge in Dublin in 1843. After years of trying, he realized that three dimensions are not enough to define the next larger arithmetic system beyond the complex number system; four dimensions (scalar together with the unit vectors **i**, **j** and **k** of three-dimensional space) are required to obtain a system with useful properties. His fundamental insight is that the products of quaternion basis vectors are related to each other as described in Table 1. There are two outstanding facts that this table shows. First, each of the independent basis vectors is equally viable to be considered as "imaginary", as captured by the famous equation Hamilton carved on the bridge: $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$. Second, quaternion multiplication is not, in general, commutative since $\mathbf{ij} \neq \mathbf{ji}$, etc. The value of a quaternion, Q, is composed of its scalar (Q_0) and vector (Q_1, Q_2 and Q_3) components:

$$Q = Q_0 + Q_1\mathbf{i} + Q_2\mathbf{j} + Q_3\mathbf{k} \quad (5)$$

$$Q = Q_{00} + Q_{01}\mathbf{i} + Q_{10}\mathbf{j} + Q_{11}\mathbf{k} \quad (6)$$

Single-digit decimal and two-bit binary subscripts are both used in this paper to refer to the same element, e.g., $Q_{10} =$

Q_2.

Addition and subtraction of quaternion values is implemented in the obvious way of adding and subtracting corresponding elements, and is therefore a commutative operation. The conjugate, $Q^* = Q_0 - Q_1\mathbf{i} - Q_2\mathbf{j} - Q_3\mathbf{k}$, changes the signs of all elements (i.e., $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$) except the scalar. As we would expect, the norm, $|Q|$, is a non-negative real computed as the square root of the sum of the squares of the four elements. In fact, the quaternion system has all the field properties that real and complex numbers have, except that multiplication is not commutative. Mathematically, the quaternion system is considered to be a non-commutative division ring. (Every non-zero quaternion, Q, has a reciprocal, $1/Q = Q^*/|Q|^2$.) Given the rectangular values of two quaternions, P and Q, the resulting value, R, from multiplication of P by Q is given by:

$$\begin{aligned} R_0 &= P_0Q_0 - P_1Q_1 - P_2Q_2 - P_3Q_3 \\ R_1 &= P_0Q_1 + P_1Q_0 + P_2Q_3 - P_3Q_2 \\ R_2 &= P_0Q_2 - P_1Q_3 + P_2Q_0 + P_3Q_1 \\ R_3 &= P_0Q_3 + P_1Q_2 - P_2Q_1 + P_3Q_0 \end{aligned} \tag{7}$$

Just as with the complex numbers, the norm of the product, $|PQ|$, equals the product of the norms, $|P||Q|$—the rather involved details in Table 1 and (7) are required to achieve this property. From this we can see that although $PQ \neq QP$, $|PQ| = |QP|$ because the norm is a real number where the commutative property applies. Just as with the real and complex numbers $\log|PQ| = \log|P| + \log|Q|$; however again this is just a real number that describes the magnitude of the quaternion. The question here becomes whether there can be a suitable logarithmic representation for the complete quaternion.

In fact, the natural logarithm, $q = \log_e(Q)$ of a quaternion value, Q, is defined mathematically [28], [13]:

$$\begin{aligned} q_0 &= \log_e|Q| \\ q_1 &= u_1\theta = \frac{Q_1}{\sqrt{Q_1^2+Q_2^2+Q_3^2}}\theta \\ q_2 &= u_2\theta = \frac{Q_2}{\sqrt{Q_1^2+Q_2^2+Q_3^2}}\theta \\ q_3 &= u_3\theta = \frac{Q_3}{\sqrt{Q_1^2+Q_2^2+Q_3^2}}\theta \end{aligned} \tag{8}$$

where the angle $\theta = \arccos(Q_0/|Q|)$ is encoded on top of the 3D-unit vector, $(u_1, u_2, u_3) = \mathbf{u} = \mathbf{Q}/|\mathbf{Q}|$. Boldface variables (either upper- or lower-case) in this paper are used for 3D vectors, which are the non-scalar part of a similarly-named quaternion, i.e., in general, $u = u_0 + \mathbf{u}$; however in this case $u_0 = 0$. Some of the well-known properties useful for LNS do hold with this definition, such as $\log(Q^a) = a\log(Q)$, where a is real. (The cases of squaring, $a = 2$, and square root, $a = 0.5$, are the ones most often implemented in LNS; Spherical quaternion interpolation often uses this with a scalar between zero and one [13].) We will refer to $q = \log(Q)$ as a *single-angle representation* of the quaternion since only θ is involved. This angle can be recovered from $\mathbf{q}$ by a suitably-defined quaternion $\exp(q)$ function described in the next section because $\mathbf{u}$ is defined be a unit vector: the magnitude $|\mathbf{q}| = \sqrt{q_1^2+q_2^2+q_3^2}$ is θ. This definition of the logarithm is a natural generalization of the complex logarithm—the case when $Q_2 = Q_3 = 0$ gives $q_0 = \log_e(\sqrt{Q_0^2+Q_1^2})$ and $q_1 = \arccos(Q_0/\sqrt{Q_0^2+Q_1^2}) = \arctan(Q_0, Q_1)$, which is identical to (2). This offers the hope the classical LNS definition which is quite successful for CLNS can be generalized to quaternions; however, there is a problem with the classical LNS for quaternions, as described in the next section.

IV. Problem with Classical LNS

For simplicity of discussion in this section, assume base-e (rather than arbitrary base-β) logarithms because the problem illustrated in this section does not depend on the base and the mathematical definition of base-e is simplest.

The central advantage of the classical definition for real and complex logarithmic number systems is the ease with which multiplication can be implemented. Given positive reals, A and B, represented by $a_0 = \log(A)$ and $b_0 = \log(B)$, it is well-known that logarithmic representation converts multiplication into addition: $AB = e^{a_0}e^{b_0} = e^{a_0+b_0} = e^{b_0+a_0} = BA$ or $\log(AB) = \log(BA) = a_0 + b_0 = b_0 + a_0$. A similar property holds for complex values, $\bar{X}$ and $\bar{Y}$ represented by (x_0, x_1) and (y_0, y_1): $\bar{X}\bar{Y} = e^{x_0+y_0}\text{cis}(x_1 + y_1) = e^{y_0+x_0}\text{cis}(y_1 + x_1) = \bar{Y}\bar{X}$. This translation of real and complex multiplication into addition works because real and complex systems both have commutative addition and commutative multiplication.

The situation with a quaternion system is different: addition is still commutative, but multiplication is not. To illustrate the difficulty this creates for a classical logarithmic representation, consider how to compute the product of two quaternions (7) from their logarithmic representations. Given a base-e logarithmic quaternion representation, p, the exponential function, $P = \exp(p)$, which is the inverse of (8) that recovers the value of the quaternion, P, is defined [13] as:

$$\begin{aligned} P_0 &= e^{p_0}p_c \\ P_1 &= e^{p_0}p_1p_s \\ P_2 &= e^{p_0}p_2p_s \\ P_3 &= e^{p_0}p_3p_s, \end{aligned} \tag{9}$$

where $p_c = \cos(|\mathbf{p}|)$, $p_s = \sin(|\mathbf{p}|)/|\mathbf{p}|$ and $\mathbf{p}$ is the 3-vector consisting of p_1, p_2 and p_3.

Assuming $Q = \exp(q)$ is similarly defined using q_c and q_s, the value of the quaternion product $R = PQ$ from (7)

can be computed as:

$$\begin{aligned} R_0 &= e^{p_0+q_0}(p_c q_c - p_1 p_s q_1 q_s - p_2 p_s q_2 q_s - p_3 p_s q_3 q_s) \\ R_1 &= e^{p_0+q_0}(p_c q_1 q_s + p_1 p_s q_c + p_2 p_s q_3 q_s - p_3 p_s q_2 q_s) \\ R_2 &= e^{p_0+q_0} \qquad (10) \\ &\quad \cdot (p_c q_2 q_s - p_1 p_s q_3 q_s + p_2 p_s q_c + p_3 p_s q_1 q_s) \\ R_3 &= e^{p_0+q_0}(p_c q_3 q_s + p_1 p_s q_2 q_s - p_2 p_s q_1 q_s + p_3 p_s q_c). \end{aligned}$$

This can be simplified and rewritten in vector notation:

$$R = e^{p_0+q_0}(p_c q_c - p_s q_s \mathbf{p}\cdot\mathbf{q} + p_c q_s \mathbf{q} + p_s q_c \mathbf{p} + p_s q_s \mathbf{p}\times\mathbf{q}). \qquad (11)$$

Taking the base-e quaternion logarithm, $r = \log(R)$ defined with auxiliary variables θ and $\mathbf{v}$ gives:

$$\begin{aligned} r_0 &= p_0 + q_0 \qquad (12) \\ \theta &= \arccos(p_c q_c - p_s q_s \mathbf{p}\cdot\mathbf{q}) \qquad (13) \\ \mathbf{v} &= p_c q_s \mathbf{q} + p_s q_c \mathbf{p} + p_s q_s \mathbf{p}\times\mathbf{q} \qquad (14) \\ \mathbf{r} &= \big(\mathbf{v}/|\mathbf{v}|\big)\theta. \qquad (15) \end{aligned}$$

Substituting for all auxiliary variables yields an expression which seems much more complicated than the multiplication it was intended to replace:

$$\begin{aligned} r_0 &= p_0 + q_0 \qquad (16) \\ \mathbf{r} &= \Big(\cos(|\mathbf{p}|)\sin(|\mathbf{q}|)\mathbf{q}/|\mathbf{q}| \qquad (17) \\ &\quad + \sin(|\mathbf{p}|)\cos(|\mathbf{q}|)\mathbf{p}/|\mathbf{p}| \\ &\quad + \sin(|\mathbf{p}|)\sin(|\mathbf{q}|)(\mathbf{p}\times\mathbf{q})/(|\mathbf{p}||\mathbf{q}|)\Big) \cdot \frac{\mathbf{r}'}{\sqrt{\mathbf{r}''}} \end{aligned}$$

where $\mathbf{r}'$ is

$$\arccos((\cos(|\mathbf{p}|)\cos(|\mathbf{q}|) - \sin(|\mathbf{p}|)\sin(|\mathbf{q}|)/(|\mathbf{p}||\mathbf{q}|)\mathbf{p}\cdot\mathbf{q}))$$

and $\mathbf{r}''$ is

$$\cos^2(|\mathbf{p}|)\sin^2(|\mathbf{q}|) + \sin^2(|\mathbf{p}|)\cos^2(|\mathbf{q}|) + \sin^2(|\mathbf{p}|)\sin^2(|\mathbf{q}|).$$

Although further simplification may be possible, it seems unlikely that the expression for $\mathbf{r}$ can be as simple as the well-known rule which holds for real and complex values: the logarithm of a product is the sum of the logarithms. We conclude that a logarithmic number system derived from this definition is not likely to be practical.

V. Novel Quaternion Complex LNS Representation

An alternative way, known as the *Cayley-Dickson construction* [26], [28], to define a quaternion is to start with a pair of complex values, $\bar{Q}_0 = Q_{00} + Q_{01}\mathbf{i}$ and $\bar{Q}_1 = Q_{10} + Q_{11}\mathbf{i}$, each of which contains half of the information for the quaternion. (For simplicity, we will consider the elements as a linear array with binary subscripts as mentioned earlier, rather as a two-by-two matrix, i.e., $Q_{1,0} = Q_{10}$.) While such Cayley-Dickson representations have been used to construct multipliers for quaternions in rectangular format [26] and used to propose an alternative single-angle polar representation (with a complex angle in [28] rather than the real angles used in this paper), to our knowledge no one has combined Cayley-Dickson with LNS.

Consider how a quaternion can be constructed from a pair of complex values. It is analogous to the well-known approach that one uses to construct a complex value from a pair of reals. The distinction is that we need a different, independent basis vector ($\mathbf{j}$) in this construction:

$$\begin{aligned} Q &= \bar{Q}_0 + \bar{Q}_1\mathbf{j} \qquad (18) \\ &= (Q_{00} + Q_{01}\mathbf{i}) + (Q_{10} + Q_{11}\mathbf{i})\mathbf{j} \qquad (19) \\ &= Q_{00} + Q_{01}\mathbf{i} + Q_{10}\mathbf{j} + Q_{11}\mathbf{k} \qquad (20) \end{aligned}$$

so that, as given in Table 1, $\mathbf{ij} = \mathbf{k}$ gives the fourth element of the rectangular quaternion representation. To form the quaternion conjugate of Q in this representation requires a complex conjugate, $\bar{Q}_0^*$, and a complex negative, $-\bar{Q}_1$. To form the negative simply requires the negative of both parts: $-\bar{Q}_0$ and $-\bar{Q}_1$. Assuming the norms of the two complex numbers are available, computing the norm of the quaternion involves only a single extra addition: $|Q| = \sqrt{|\bar{Q}_0|^2 + |\bar{Q}_1|^2}$. (The square and square-root operations will be easy to implement in LNS.)

Given two quaternions, each represented with a pair of complex values, Q as in (18) and similarly $P = \bar{P}_0 + \bar{P}_1\mathbf{j}$, the result, R, of multiplying P by Q can be described as a set of complex operations:

$$\begin{aligned} \bar{R}_0 &= \bar{P}_0\bar{Q}_0 - \bar{P}_1\bar{Q}_1^* \qquad (21) \\ \bar{R}_1 &= \bar{P}_0\bar{Q}_1 + \bar{P}_1\bar{Q}_0^* \qquad (22) \end{aligned}$$

Except for the presence of the conjugate operations, this algorithm is the familiar rectangular multiplication algorithm. If one substituted reals for the complex variables, (21) and (22) would reduce to (1) because the conjugate of a real is the identity function. With complex variables, the conjugate operations in (21) and (22) succinctly explain why quaternion multiplication is non-commutative, even though it can be constructed from commutative complex multiplies and adds[1].

It is easy, although tedious, to verify that (21) and (22) are equivalent to (7). (21) and (22) are not as well-known in quaternion literature as (7). This is understandable, since in the conventional rectangular representation of complex and quaternion values, both (7) as well as (21) and (22) require twelve floating-point additions and sixteen floating-point multiplications—there is no distinction between the two approaches when one works with rectangular representations.

[1]Although not a concern here, a similar construction can extend the quaternions into the next larger system, the octonions; however, due to the non-commutativity of quaternion multiplication, the order of the terms must be slightly different than (21) and (22).

The novel concept in this paper, which we will refer to as the Quaternion Complex LNS (QCLNS), is to replace the rectangular representation for the complex variables:

$$\begin{aligned} \bar{P}_0 &= P_{00} + P_{01}\mathbf{i} \\ \bar{P}_1 &= P_{10} + P_{11}\mathbf{i} \\ \bar{Q}_0 &= Q_{00} + Q_{01}\mathbf{i} \\ \bar{Q}_1 &= Q_{10} + Q_{11}\mathbf{i} \end{aligned}$$

with the CLNS representation for the same values:

$$\begin{aligned} \bar{P}_0 &= \beta^{p_{00}}\text{cis}(p_{01}) \\ \bar{P}_1 &= \beta^{p_{10}}\text{cis}(p_{11}) \\ \bar{Q}_0 &= \beta^{q_{00}}\text{cis}(q_{01}) \\ \bar{Q}_1 &= \beta^{q_{10}}\text{cis}(q_{11}). \end{aligned} \tag{23}$$

In other words, $P = \beta^{p_{00}}\text{cis}(p_{01}) + \beta^{p_{10}}\text{cis}(p_{11})\mathbf{j}$ and $Q = \beta^{q_{00}}\text{cis}(q_{01}) + \beta^{q_{10}}\text{cis}(q_{11})\mathbf{j}$. In an analogous way to how DRLNS and MDLNS use an additive definition to obtain the final real value, QCLNS uses an additive definition to obtain the final quaternion value. In contrast to the classical quaternion logarithm, we will refer to this QCLNS approach as a *two-angle* representation because q_{01} and q_{11} are independent angles that together help describe the quaternion, Q. Because these angles can be stored in fewer bits than a rectangular value, there is a reasonable memory savings by using QCLNS to represent a quaternion instead of using four real values. Assuming ζ compression, the QCLNS representation needs $2K + 4F + 6$ bits versus the $4K+4F+4$ bits needed for the conventional rectangular representation. For single precision ($F = 23$,$K = 8$), this means the difference between 114 bits versus 128 bits, which is a 10.9 percent savings.

Forming quaternion negatives and conjugates in QCLNS is trivial. To compute the LNS representation of the quaternion norm involves only a single s_β unit: $\log_\beta |Q| = q_{00} + s_\beta(2(q_{10} - q_{00}))/2$. The interesting case is multiplication. Using QCLNS, the four complex multiplies in (21) and (22) reduce to eight fixed-point adders (i.e., real LNS multipliers) instead of sixteen floating-point multipliers with the conventional approach. It has been proven in the conventional context that the minimum number of floating-point multipliers needed for a general quaternion multiplier is eight [15], [11]. Although the context here is slightly different, the proposed approach achieves this minimum of eight multipliers (implemented as trivial fixed-point adders likely to be much cheaper than floating-point multipliers.)

Of course, this representation has a significant cost for quaternion multiplication: (21) has a complex subtract and (22) has a complex add. Assuming this significant hardware is available, the QCLNS ALU has just the right amount of hardware (two complex adds) to perform both QCLNS addition and multiplication (in contrast to the unbalanced conventional situation of twelve floating-point adds for quaternion multiply but only four floating-point adds for quaternion addition.) The problem is that CLNS addition and subtraction traditionally have been very expensive table-lookup operations; the recent advances in LNS [12] and CLNS [4] ALU design offers the hope that QCLNS may be affordable. Section VII provides area estimations to address this issue, but first we need to consider whether the QCLNS representation is accurate enough to make it worthwhile to implement.

VI. Accuracy of QCLNS

Real additive-log representations, like DRLNS, are known to have multiplication accuracy problems in the general context [2]. To determine whether QCLNS avoids these difficulties, we conducted a simulation that compares when single-precision multiplies are carried out using QCLNS and floating-point. This simulation suggests that even though QCLNS is an additive-log system, the independent basis for the complex and quaternion systems from which it is defined avoids the condition-number problems reported [2] for the dependent-basis DRLNS.

We took all combinations of integer values from -3 to 3 for each of the four dimensions of a conventional quaternion (7^4 unique quaternion values). The inclusion of 3, which does not have an exact base-two logarithmic representation, gives the simulation some opportunity to uncover difficult QCLNS cases (such as QCLNS's asymmetrical treatment of **i**, **j** and **k**). Since integers have an exact floating-point representation unlikely to occur often in real-world quaternion applications, we added a small amount of uniform noise to all non-zero elements. (Exact zeros, represented by the special flags ignored in earlier sections, are important for representing pure scalar and pure basis vectors; our simulation tested all combinations of these.) All possible 7^8 products of the input set were computed in double-precision floating point. The input values were also converted into single-precision (128-bit quaternion) floating-point and similar-precision ζ-compressed QCLNS (114 bit-quaternion) representations. Errors for the two single-precision products are computed assuming the double-precision result may serve as a reference for error computation. The observed error for QCLNS is 16 percent better ($4.5 \cdot 10^{-6}$) than the error for floating-point ($5.3 \cdot 10^{-6}$), even though QCLNS uses fewer bits than the floating-point implementation. Although any feasible 8-dimensional simulation with realistic F is too small to give a conclusive result, this test raises our hope that QCLNS does not have large systematic errors of the kind in [2].

VII. Hardware Optimization and Area Estimation

Although an obvious implementation of (21) and (22) would use eight fixed-point adders to implement the four complex multiplies together with two independent CLNS adder circuits of the kind described in [4], there are further

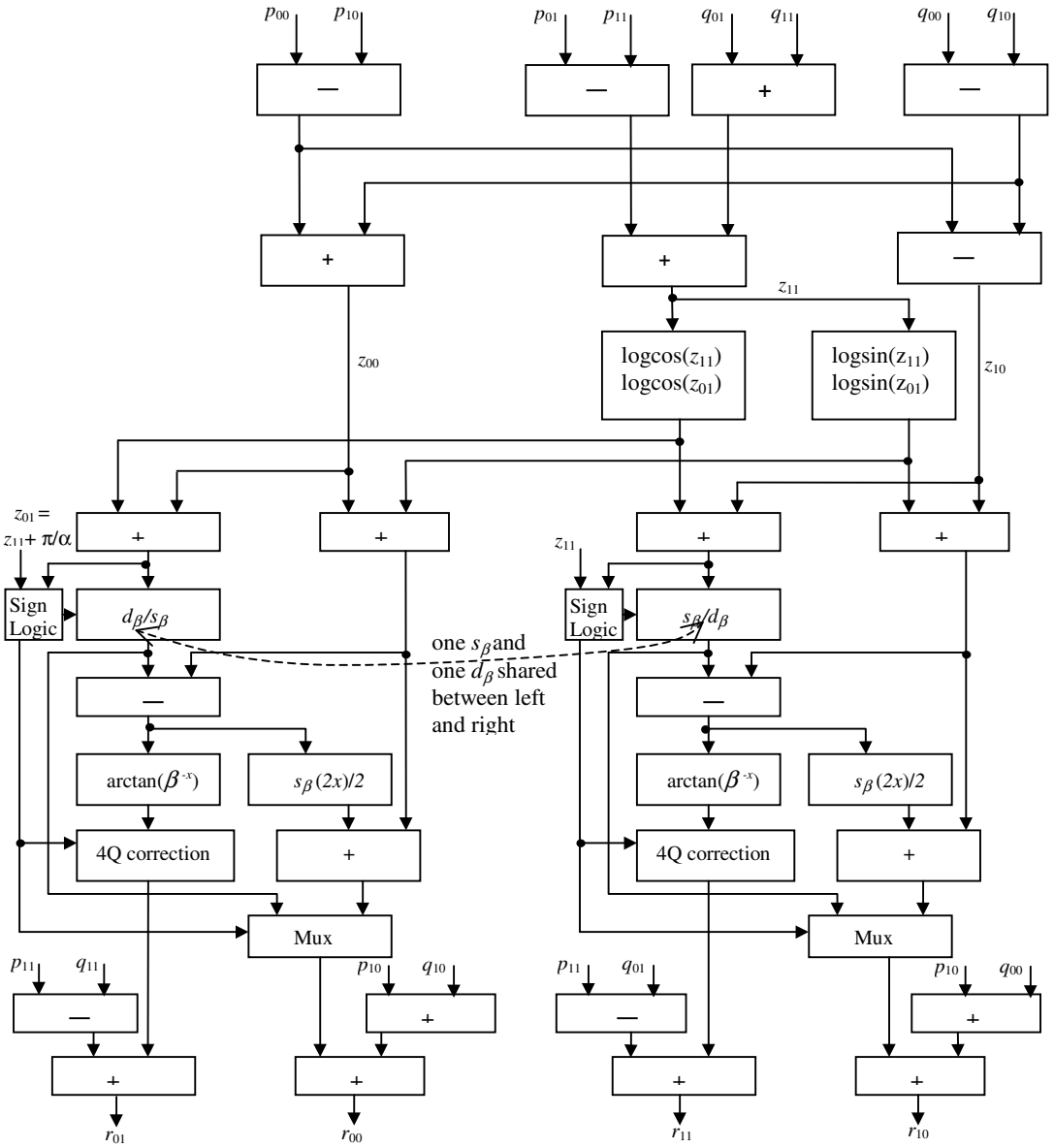

Figure 1. Proposed QCLNS multiplier optimized to share log-trig and s_β/d_β tables.

optimizations possible that allow hardware reduction. In particular, the two CLNS adders share common subexpressions. By substituting (23) into (22), we have:

$$\begin{aligned} \bar{R}_1 &= \beta^{p_{00}}\text{cis}(p_{01})\beta^{q_{10}}\text{cis}(q_{11}) \qquad (24)\\ &\quad + \beta^{p_{10}}\text{cis}(p_{11})\beta^{q_{00}}\text{cis}(q_{01})^* \\ &= \beta^{p_{00}+q_{10}}\text{cis}(p_{01}+q_{11}) \qquad (25)\\ &\quad + \beta^{p_{10}+q_{00}}\text{cis}(p_{11}-q_{01}). \end{aligned}$$

To perform this complex addition using CLNS, we first need to form a complex value, $\bar{Z}_1$; then obtain $\bar{Z}_1+1$ using $s_\beta(\bar{z}_1)$ and finally multiply the result by the addend to form the sum. The value needed to start this process is

$$\begin{aligned} \bar{Z}_1 &= \frac{\beta^{p_{00}+q_{10}}\text{cis}(p_{01}+q_{11})}{\beta^{p_{10}+q_{00}}\text{cis}(p_{11}-q_{01})} \qquad (26)\\ &= \beta^{p_{00}+q_{10}-p_{10}-q_{00}}\text{cis}(p_{01}+q_{11}-p_{11}+q_{01}), \end{aligned}$$

and its associated CLNS representation is:

$$\begin{aligned} z_{10} &= p_{00}+q_{10}-p_{10}-q_{00} \qquad (27)\\ &= (p_{00}-p_{10})-(q_{00}-q_{10}) \\ z_{11} &= p_{01}+q_{11}-p_{11}+q_{01} \qquad (28)\\ &= (p_{01}-p_{11})+(q_{11}+q_{01}). \end{aligned}$$

In a similar way, substituting (23) into (21) and noting $e^{\pi \mathbf{i}} = -1$ gives

$$\begin{aligned} \bar{R}_0 &= \beta^{p_{00}}\text{cis}(p_{01})\beta^{q_{00}}\text{cis}(q_{01}) \quad (29)\\ &\quad - \beta^{p_{10}}\text{cis}(p_{11})\beta^{q_{10}}\text{cis}(q_{11})^* \\ &= \beta^{p_{00}+q_{00}}\text{cis}(p_{01}+q_{01}) \quad (30)\\ &\quad + \text{cis}(\pi/\alpha)\beta^{p_{10}+q_{10}}\text{cis}(p_{11}-q_{11}) \\ &= \beta^{p_{00}+q_{00}}\text{cis}(p_{01}+q_{01}) \quad (31)\\ &\quad + \beta^{p_{10}+q_{10}}\text{cis}(\pi/\alpha + p_{11}-q_{11}). \end{aligned}$$

The value that starts the process is

$$\bar{Z}_0 = \beta^{p_{00}+q_{00}-p_{10}-q_{10}}\text{cis}(\pi/\alpha + p_{01}+q_{01}-p_{11}+q_{11}), \quad (32)$$

and its associated CLNS representation is:

$$\begin{aligned} z_{00} &= p_{00}+q_{00}-p_{10}-q_{10} \quad (33)\\ &= (p_{00}-p_{10})+(q_{00}-q_{10}) \\ z_{01} &= \pi/\alpha + p_{01}+q_{11}-p_{11}+q_{01} \quad (34)\\ &= \pi/\alpha + (p_{01}-p_{11})+(q_{11}+q_{01}). \end{aligned}$$

When defined this way, z_{00} and z_{10} are formed by the sum and difference of $(p_{00}-p_{10})$ and $(q_{00}-q_{10})$. Also, (28) and (34) always differ by exactly π/α, i.e., $z_{01} = \pi/\alpha + z_{11}$. Using the algorithm described in [4], the first step in each approximation unit is to form the logarithms of the absolute values of the sine and cosine of that unit's input angle. Since $\sin(\alpha z_{11}) = -\sin(\alpha z_{11}+\pi)$ and $\cos(\alpha z_{11}) = -\cos(\alpha z_{11}+\pi)$, the starting logsin and logcos are identical. It is not necessary to provide each approximation unit with separate trigonmetric hardware—a significant savings because of the singularity of logsin. One trig unit may be shared between (21) and (22), as shown in Figure 1. The second step in the algorithm described in [4] is to choose whether to perform a real-valued s_β or d_β based on whether the angle is inside or outside the range $-\pi/2$ to $\pi/2$. The definitions of z_{11} and z_{01} guarantee that one unit will use s_β whilst the other will use d_β, or vice versa. A trivial set of muxes (not shown in Figure 1 for clarity) allows the s_β and d_β tables to be shared between the two units.

QCLNS is built on top of CLNS, which itself is built on top of conventional LNS. The QCLNS multiplier involves several such levels of abstraction; at each level many design decisions are possible. In order to estimate the area required for QCLNS, we rely upon the published results in [4] and [12]. Further, we assume a more efficient LNS trigonmetric [22] algorithm ($\log|\sin(x)| = \log|\sin(x)/x| + \log|x|$) than was used in [4]. The original CLNS synthesis results [4] assume all hardware in the FPGA is implemented with logic (CLBs); the synthesis results from Fu et al. [12] use the block RAMs and multipliers available in large FPGAs, and are the best available synthesis benchmarks for conventional LNS to the best of our knowledge. Unfortunately, the synthesis results in [12] use a proprietary tool, ASC, which is not available to be modified for QCLNS. Therefore, we rely primarily on the published synthesis results from [12]; however some observations must be made from the results in [4] to extend the results of Fu et al. into the more complicated situation with QCLNS. First, the $\arctan(\beta^x)$ function and the s_β function alone are nearly identical mathematically, and therefore need nearly the same area. Second, the logcos function takes about one-half the area of the s_β function alone. Third, for small F, the FloPoCo-generated tables in [4] are more efficient than the minimum-512-element block-RAM tables used in [12]; however for the larger F of interest in quaternion applications, [12] uses less. Fourth, the slices in [4] grow monotonically as F increases, whereas the slices in [12] reflect a complicated heuristic for dealing with the minimum-512-element size (making the paradox in a few instances that $F+2$-bit precision takes less area than F-bit precision, for example $F = 31$ is worse than $F = 33$).

Based on all these assumptions, Figure 1 needs two actual s_β-only tables and two similarly-sized $\arctan(\beta^x)$ tables plus three (the extra table due to $\log(x)$ needed in [22]) log-trig tables (each about half the size of s_β) for a total of 5.5 s_β-only-sized tables. In addition, Figure 1 requires one unit that shares s_β and d_β. Also, there are about twenty fixed-point adders (approximately the complexity of one floating-point multiplier—underestimating for $F < 20$ cases and overestimating for the $F > 20$ cases we are more interested in). By comparison, conventional rectangular-quaternion multiplication requires 12 floating-point (or LNS) adders and 16 floating-point (or LNS) multipliers. (Although there are efficient quaternion multiplication algorithms [11], [15], [26] that reduce the number of floating-point multipliers, such algorithms tend to increase the number of adders which cost more in FPGA slices than the floating-point multipliers they replace.[12]) Figure 2 plots the equivalent slices with these assumptions. The rectangular LNS implementation is much more expensive than the FP or QCLNS alternatives. As a function of F, the size difference between QCLNS and FP varies quite significantly (reflecting the approximate nature of this estimation and the 512-element paradox of the data from [12]); however, the proposed QCLNS implementation seems to use on average 10 percent less area than floating point (the best case is a savings of 32 percent for $F = 25$; the worst case of $F = 31$ (where QCLNS appears worse than FP) can be explained by the 512-element paradox).

VIII. Conclusion

We have shown that the two most natural ways that one might attempt to use LNS for quaternion multiplication are not cost effective: first, the quaternion logarithm function does not help to simplify multiplication because quaternion multiplication is not commutative but quaternion addition is; second, using LNS for the twelve add/subs involved in the rectangular definition of quaternion multiplication is much

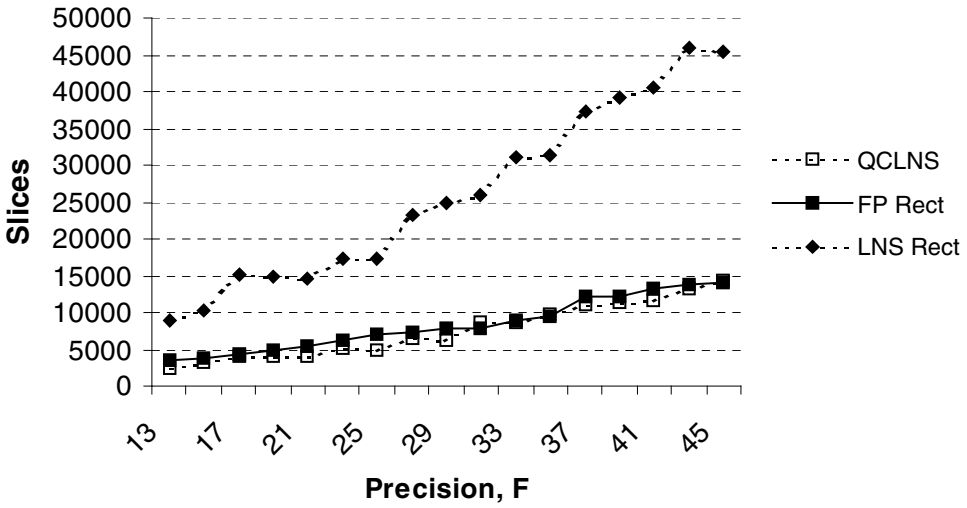

Figure 2. Xilinx FPGA slices required for QCLNS versus rectangular-LNS and -FP multiplication.

more expensive than using floating point. To overcome this, we have outlined a new representation, QCLNS, for the quaternion, Q, using a pair of complex numbers, $\bar{Q}_0$ and $\bar{Q}_1$, in a Cayley-Dickson construction, $Q = \bar{Q}_0 + \bar{Q}_1\mathbf{j}$, with the novel point that each complex number is represented in CLNS log-polar form. A straightforward implementation with this representation needs four CLNS multipliers and two CLNS adders (the equivalent of seven LNS add-onlys and two LNS add/subs), but because the CLNS adders have common subexpressions, the hardware can be optimized down to the equivalent of about 5.5 LNS add-onlys and one LNS add/sub (shared between the left and right paths).

Actually implementing, synthesizing and testing the proposed ALU will be quite involved because of the three levels of abstraction (LNS, CLNS and QCLNS), and the need to develop them from scratch. (The LNS tool in [12] capable of realistic F is proprietary, and its paradoxical cases suggest it could be improved; yet it is difficult to use the available CLNS tool in [4] much beyond $F = 20$.) Determining the delays for floating point and QCLNS is difficult without synthesizing actual circuits. (Fu et al. [12] does not report as much detail on delay as on area, and the delay for the glue logic in Figure 1 is difficult to estimate without synthesizing the circuit.) We have presented three pieces of evidence that such future research might be worth the investment required. First, we have shown QCLNS occupies less memory (10.9 percent in the case of $F = 23$ single precision) than conventional quaternion representation. Second, a small simulation (about five million integer-plus-noise quaternion products) suggested that ideal $F = 23$ QCLNS might be up to 16 percent more accurate than conventional quaternion multiplication; however, some of this may be lost depending on the details in a synthesized ALU. Third, by extrapolating from LNS FPGA results reported by Fu et al. [12], we conclude: a) with some certainty, the proposed QCLNS approach is not significantly more expensive than the conventional floating-point approach, and b) with less certainty, $13 \leq F \leq 45$ QCLNS might be, on average, 10 percent more area efficient than floating point. Although these preliminary estimates may not exactly predict the performance of an actual synthesized circuit, they suggest QCLNS may offer some combination of smaller bit-size representations (with smaller busses, memories and associated-power consumption), smaller area and/or more accurate results.

These features could benefit embedded systems that make heavy use of quaterions, such as gaming and navigation. Such potential users might have concerns about the asymmetric treatment of **i**, **j** and **k** that could cause subtle numerical instabilities. Implementers of very critical systems that could benefit from QCLNS (e.g., spacecrafts) are unlikely to trust such an exotic system on the basis of the small simulation reported here. Given the high dimensionality of QCLNS, exhaustive testing is impossible for non-trivial bit widths. Addressing such concerns in detail is beyond the scope of this paper, but future research should consider rigorous error analysis of QCLNS compared to floating point.

REFERENCES

[1] S. L. Altmann, *Rotations, Quaternions and Double Groups,* Dover, Mineola, NY, 1986.

[2] M. G. Arnold, T. A. Bailey, J. R. Cowles and J. J. Cupal,"Redundant Logarithmic Arithmetic," *IEEE Transactions on Computers*, vol. 39, pp. 1077–1886, August 1990.

[3] M. G. Arnold, T. A. Bailey, J. R. Cowles and M. D. Winkel, "Arithmetic Co-transformations in the Real and Complex Logarithmic Number Systems," *IEEE Transactions on Computers*, vol. 47, no. 7, pp. 777–786, July 1998.

[4] M. G. Arnold and S. Collange, "A Dual-Purpose Real/Complex Logarithmic Number System ALU," *19th IEEE Symposium on Computer Arithmetic*, Portland, Oregon, pp. 15–24, June 2009.

[5] M. G. Arnold and S. Collange, "A Real/Complex Logarithmic Number System ALU," *IEEE Transactions on Computers*, vol. 60, no. 2, pp. 202–213, Feb 2011.

[6] C. Chen and C. H. Yang, "Pipelined Computation of Very Large Word-Length LNS Addition/Subtraction with Polynomial Hardware Cost," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 716–726, July 2000.

[7] Jack C. K. Chou and M. Kamel, "Finding the Position and Orientation of a Sensor on a Robot Manipulator Using Quaternions," *The International Journal of Robotics,* vol. 10, no. 3, pp. 240–254, June 1991.

[8] M. Da Silva, Y. Abe and J. Popovic, "Simulation of Human Motion Data using Short-Horizon Model-Predictive Control," *Computer Graphics Forum,* vol. 27, pp. 371–380, 2008.

[9] J. Detrey and F. de Dinechin, "A Tool For Unbiased Comparison Between Logarithmic and Floating-Point Arithmetic," *Journal of VLSI Signal Processing*, Springer, vol. 49, no. 1, pp. 161–175, 2007.

[10] F. de Dinechin, B. Pasca, O. Creţ and R. Tudoran, "An FPGA-specific Approach to Floating-Point Accumulation and Sum-of-Products," *Field-Programmable Technology*, IEEE, pp. 33–40, 2008.

[11] H. F. de Groote, "On the Complexity of Quaternion Multiplication," *Information Processing Letters,* vol 3, no. 6, pp. 177–179, July 1975.

[12] H. Fu, O. Mencer and W. Luk, "FPGA Designs with Optimized Logarithmic Arithmetic," *IEEE Transactions on Computers*, vol. 59, no. 7, pp.1000–1006, July 2010.

[13] A. J. Hanson, *Visualizing Quaternions,* Elsevier, Amsterdam, 2006.

[14] M. Haselman, M. Beauchamp, K. Underwood and K Hemmert, "A Comparison of Floating Point and Logarithmic Number Systems for FPGAs," *FCCM* pp. 181–190, 2005.

[15] T. D. Howell and J. C. Lafon, "The Complexity of the Quaternion Product," Cornell University, Ithaca, NY, TR 75-245, June 1975.

[16] L. Jinsong, R. Fullmer and Y. Q. Chen, "Time-optimal magnetic attitude control for small spacecraft," *43rd IEEE Conference on Decision and Control,* vol. 1, pp. 255–260, Dec. 2004.

[17] J. B. Kuipers, *Quaternion and Rotation Sequences,* Princeton University Press, 1999.

[18] H. K. Kwan and S. B. Arniker, "Numerical representation of DNA sequences," *IEEE International Conference on Electro/Information Technology,* Windsor, ON, pp. 307–310, June 2009.

[19] J. Lambeck, "If Hamilton Had Prevailed: Quaternions in Physics," *The Mathematical Intellegencer,* vol. 17, no. 4, pp 7–15, 1995.

[20] D. M. Lewis, "Complex Logarithmic Number System Arithmetic Using High Radix Redundant CORDIC Algorithms," *14th IEEE Symposium on Computer Arithmetic*, Adelaide, Australia, pp. 194–203, April 1999.

[21] B. Macadangdang and C. Dwyer, "DNA Self-Assembly of Nanoelectronic Devices: The Nanodynamics Simulator," `http://beta.ece.duke.edu/files/ece/Macadangdang_Rpt2007.pdf`, 2007.

[22] E. Malamas, V. Paliouras and T. Stouraitis, "Efficient Algorithms and VLSI Architectures for Trigonometric Functions in the Logarithmic Number System Based on the Subtraction Function," *Proceedings of the Third IEEE International Conference on Electronics, Circuits and Systems*, pp. 964–967, Rodos, Greece, 13-16 October 1996.

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

[24] Microsoft Corp., "Programming Guide for Direct 3D," `http://msdn.microsoft.com/en-us/library/bb206327(VS.85).aspx`

[25] R. Muscedere, V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "Efficient Conversion from Binary to Multi-Digit Multidimensional Logarithmic Number Systems Using Arrays of Range Addressable Look-Up Tables," *Application-specific Systems, Architectures and Processors*, San Jose, pp. 130–138, July 2002.

[26] M. Parfieniuk and A. Petrovsky, "Quaternion Multiplier Inspired by the Lifting Implementation of Plane Rotations," *IEEE Transactions on Circuit and Systems I: Regular Papers*, vol. 57, no. 10, pp. 2708–2717, Oct. 2010.

[27] S.-C. Pei, J.-J. Ding and J.-H. Chang, "Efficient Implementation of Quaternion Fourier Transform, Convolution, and Correlation by 2-D Complex FFT," *IEEE Transactions on Signal Processing,* vol. 49, no. 11, pp. 2783–2797, Nov. 2001.

[28] S. J. Sangwine and N. Le Bihan, "Quaternion Polar Representation with a Complex Modulus and Complex Argument Inspired by the Cayley-Dickson Form," *Advanced Applied Clifford Algebra,* vol. 20, pp. 111–120, 2010.

[29] T. Stouraitis, *Logarithmic Number System Theory, Analysis, and Design*, Ph.D. Dissertation, University of Florida, Gainesville, 1986.

[30] E. E. Swartzlander and A. G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Transactions on Computers*, vol. C-24, pp. 1238–1242, Dec. 1975.

[31] P. Vouzis and M. G. Arnold, "A Parallel Search Algorithm for CLNS Addition Optimization," *IEEE International Symposium on Circuits and Systems*, Kos, Greece, pp. 2417–2420, May 2006.

[32] `http://www.xlnsresearch.com`.