The Denormal Logarithmic Number System

Mark G. Arnold, Caroline Collange

The Denormal Logarithmic Number System

Mark G. Arnold
XLNS Research
PO Box 605 Laramie WY 82070 USA
markgarnold@xlnsresearch.com

Caroline Collange
INRIA
Centre de recherche Rennes - Bretagne Atlantique
Rennes, France
caroline.collange@inria.fr

***Abstract*—Economical hardware often uses a FiXed-point Number System (FXNS), whose constant absolute precision is acceptable for many signal-processing algorithms. The almost-constant relative precision of the more expensive Floating-Point (FP) number system simplifies design, for example, by eliminating worries about FXNS overflow because the range of FP is much larger than FXNS for the same wordsize; however, primitive FP introduces another problem: underflow. The conventional Signed Logarithmic Number System (SLNS) offers similar range and precision as FP with much better performance (in terms of power, speed and area) for multiplication, division, powers and roots. Moderate-precision addition in SLNS uses table lookup with properties similar to FP (including underflow). This paper proposes a new number system, called the Denormal LNS (DLNS), which is a hybrid of the properties of FXNS and SLNS. The inspiration for DLNS comes from the denormal numbers found in IEEE-754 (that provide better, gradual underflow) and the μ-law often used for speech encoding; the novel DLNS circuit here allows arithmetic to be performed directly on such encoded data. The proposed approach allows customizing the range in which gradual underflow occurs. A wide gradual underflow range acts like FXNS; a narrow one acts like SLNS. Simulation of an FFT application illustrates a moderate gradual underflow decreasing bit-switching activity 15% compared to underflow-free SLNS, at the cost of increasing application error by 30%. DLNS reduces switching activity 5% to 20% more than an abruptly-underflowing SLNS with one-half the error. Synthesis shows the novel circuit primarily consists of traditional SLNS addition and subtraction tables, with additional datapaths that allow the novel ALU to act on conventional SLNS as well as DLNS and mixed data, for a worst-case area overhead of 26%.**

Keywords: Computer Arithmetic, Logarithmic Number Systems (LNS), underflow, denormal.

I. Introduction

Designers of application-specific systems often have knowledge about their numeric requirements, which can be satisfied with more economical arithmetic circuits than found in general-purpose systems. This has given rise to a variety of special-purpose number systems, that have certain advantages in application-specific systems. For example, designers may know that most numbers processed by the application fall within a certain range; neglecting an occasional small number that underflows this range gives only a small error acceptable to the application. This paper considers a new number system, which combines features of several well-known number systems, to give application-specific designers new options for dealing with such situations, particularly in applications like signal-processing.

In computer arithmetic, a *representation* (denoted in uppercase: X) is a finite vector of bits that represents a numeric value in a particular number system. The *value* (lowercase x) is a real number that may be approximated by X. There is a particular real, $\bar{x}$, that X represents exactly. Other values of x in the neighborhood of $\bar{x}$ use the same representation. The resulting error can be measured in bits of absolute error (1), or in bits of relative error (2):

$$e_a = \log_2 |x - \bar{x}| \tag{1}$$

$$e_r = \log_2 \left(\left| \frac{x - \bar{x}}{\bar{x}} \right| \right) = \log_2 |1 - x/\bar{x}|. \tag{2}$$

The simplest number system of this kind is the fixed-point number system, in which X consists of a signed integer X_F that is scaled by 2^F to provide a constant F bits of absolute precision:

$$\bar{x} = X_F \cdot 2^{-F}. \tag{3}$$

Many problems perform better when the relative precision is held constant. Binary floating-point number systems provide nearly constant relative precision by providing

$$\bar{x} = (-1)^{X_S} X_M 2^{X_E} \tag{4}$$

where X is subdivided into three parts: the sign ($X_S \in \{0, 1\}$), the fixed-point mantissa ($1 \leq X_M < 2$) and the integer exponent $X_E = \lfloor \log_2 |x| \rfloor$). The choice of these fields impacts the quality of the floating-point system. Using *hidden-bit normalization*, there can be an assumed 1 in $X_M = 1 + X_F \cdot 2^{-F}$. Because of the finite size of X, there are upper and lower bounds on the exponent, $L \leq X_E < U$, which determines the dynamic range, $2^L \leq \bar{x} < 2^U$.

To overcome incompatibility caused by different manufacturer's arbitrary choices for X_M and X_E, a formal standard for binary floating-point, IEEE-754 [15], was adopted quickly in the 1980s by all manufacturers, and was revised in 2008 [16]. IEEE-754 uses single (32-bit X, $F = 23$, $L = -126$ and $U = 128$) and double (64-bit X, $F = 52$, $L = -1022$ and $U = 1024$) precision, named Binary32 and Binary64, respectively, in the 2008 standard, with hidden-bit normalization. IEEE 754 actually encodes X_E with a biased exponent, but that is irrelevant for the discussion of what values can be represented.

One of the features introduced in IEEE 754, which was controversial at the time, is gradual underflow, sometimes

called *subnormals* or denormals. Prior to IEEE 754, most floating points left a gap between the smallest representable positive number, 2^L, and zero. There would be a similar gap on the negative side. To fill this gap, IEEE 754 defines a special case (signaled here by $X_E = L - 1$) where an unnormalized $X_M = X_F \cdot 2^{-F}$ has the same meaning as a fixed point value between zero and 2^L, in other words:

$$\bar{x} = (-1)^{X_S} X_M 2^{L-1} \tag{5}$$

when $X_E = L - 1$. The value $+0.0$ is then not a unique case, but rather just the nonnegative subnormal with the smallest absolute value, defined by $X_M = 0$ and $X_S = 0$. IEEE 754 requires a distinct representation of -0.0, similarly defined as the subnormal with $X_M = 0$ and $X_S = 1$.

In 1975, Swartzlander and Alexopoulos [27] proposed the Signed Logarithm Number System (SLNS), which represents the magnitude of values with their base-b logarithms and a concatenated sign bit. SLNS represents a real number, x, using a sign bit, X_S, and a finite approximation to the logarithm of the absolute value, $X_L = Q(2^F \cdot \log_b |x|)/2^F$, where F is the precision and Q is a quantization function whose output (defining a particular rounding mode) is an integer that fits within the finite word. A given SLNS representation, defined by X_S and X_L, maps into the exact value

$$\bar{x} = (-1)^{X_S} \cdot b^{X_L}. \tag{6}$$

With the typical choice of $b = 2$ and a symmetrical range of exponents ($L \approx -U$), the dynamic range (including non-denormal underflow) is similar to floating point, since $L \leq X_L < U$. SLNS keeps this logarithmic representation during all computation (including addition). When precision requirements are low to moderate and multiplication is more frequent than addition, SLNS is more cost effective than floating point. The simplest definition of SLNS excludes representing an exact zero; a special bit may be included to allow for this at some extra hardware cost.

An isomorphic definition of SLNS [23] uses integer powers of the smallest value greater than 1.0 that is exactly representable, $\beta = b^{2^{-F}}$. With either definition, the relative spacing between SLNS points is β, and with faithful rounding [3] when $|x|$ is larger than $|\bar{x}|$, $|x/\bar{x}| \leq \beta$. The relative precision is $1-\beta$ and from (2), the number of bits of relative precision will be the constant $\log_2(1-\beta) \approx F$.

Multiplication and division are straightforward in SLNS. Since the values are already represented as logarithms, a simple addition or subtraction computes the product or quotient, together with an exclusive OR to find the sign. Although it makes multiplication and division easy, SLNS makes addition and subtraction more difficult than fixed point. The manual algorithm for logarithmic addition was first described by Leonelli and popularized by Gauss in the early nineteenth century [14]. Swartzlander et al. [27], [26] and others [19], [12] reconsidered these algorithms and found them quite attractive in light of the technology available for digital signal processing in the 1970s. Beyond simple table lookup, several implementations [21], [8], [7], [6] have provided SLNS arithmetic with increased performance and reduced implementation cost. In particular SLNS appears to offer reduced power consumption in many applications [25], [23]. Successful applications have included massive scientific simulation [24], Hidden-Markov Models (HMM) [28], and music synthesis [20]. The European Logarithmic Microprocessor (ELM) [9] provides dual SLNS ALUs that implement the Gauss/Leonelli algorithm in 0.18 μm 125MHz hardware. More recently, advances in FPGA [13] and cotransformation [17] implementations of SLNS allow higher-precision applications to be affordable. Logarithmic arithmetic has generalizations in the complex numbers [4] and quaternions [5].

The Gauss/Leonelli addition algorithm requires computing one of the two following functions. When the signs of the numbers to be added are the *same*, the hardware computes

$$s_b(z) = \log_b(1 + b^z).$$

For all possible z, $0 < s_b(z)$. For $z > 0$, $s_b(z) \geq z$. It is not necessary for the hardware to deal with both positive and negative z since

$$s_b(z) = s_b(-z) + z. \tag{7}$$

When the signs of the numbers are *different*, the hardware computes

$$d_b(z) = \log_b \left|1 - b^z\right|. \tag{8}$$

For $z \geq \log_b(2)$, $0 \leq d_b(z) < z$. Analogously to s_b,

$$d_b(z) = d_b(-z) + z. \tag{9}$$

There is a point, $E_0 \approx F$, known as the essential zero, for $z < -E_0$ where $s_b(z) < 2^{-F}$ and $d_b(z) < 2^{-F}$, in other words, the quantized values are zero. From (7) and (9), there is a similar essential-identity property ($s_b(z) \approx d_b(z) \approx z$) for $z > E_0$

Given $\bar{x}$ represented as X_S and X_L, and $\bar{y}$ represented as Y_S and Y_L, there are two cases for SLNS addition. If $X_S = Y_S$,

$$\begin{aligned} \bar{x} + \bar{y} &= (-1)^{R_S} \cdot \left(b^{X_L} + b^{Y_L}\right) \\ &= (-1)^{R_S} \cdot \left(b^{X_L} \cdot (1 + b^{Y_L}/b^{X_L})\right) \\ &= (-1)^{R_S} \cdot b^{R_L}, \end{aligned}$$

where the actual computation performed by the hardware is

$$R_L = X_L + s_b(Y_L - X_L). \tag{10}$$

If $X_S \neq Y_S$, the hardware does a similar computation,

$$T_L = X_L + d_b(Y_L - X_L). \tag{11}$$

(The variables P ... T will be reserved for results in this paper.) The sign of the result (R_S or T_S) is simply the sign of the larger of the input arguments.

An earlier attempt to incorporate denormals into SLNS [2] is quite different than what is proposed in this paper. Arnold et al. [2] treat denormals specially and use over a dozen cases to consider operands and results of different magnitudes. In contrast, the novel representation proposed here may accom-

plish similar gradual underflow using simple algorithms that do not explicitly refer to the magnitude of the operands or results. The simple algorithms proposed here will be much more efficient than those of [2] in a software-based gradual-underflow implementation (for instance, on the the ELM [9], [17], a microprocessor that provides hardware for SLNS-without-denormals). Furthermore, while [2] only applies to denormals patterned after IEEE-754, the novel approach in this paper suggests a range of denormal representations (from one similar to IEEE-754 to a fully-denormal one similar to the μ-law for speech encoding [30]).

Section II describes the novel DLNS representation and gives options for how addition may be performed. Section III considers simplifications possible when not all operands are given in DLNS. Section IV presents a simple model for DLNS error, and observes this model roughly predicts the errors we observe with actual DLNS arithmetic in simulation of a typical application, the Fast Fourier Transform (FFT). This section also reports DLNS may reduce bit-level switching activity (and therefore power consumption) for the FFT. Section V presents synthesis results for the preferred circuit. Section VI presents conclusions.

II. DLNS Addition

The Denormal Logarithmic Number System (DLNS) uses

$$\bar{x} = (-1)^{X_S} \cdot \left(b^{X_D} - b^J\right), \tag{12}$$

where $J \leq X_D < U$ and $J \leq 0$ is an integer constant for implementation convenience. Notice that, unlike simple SLNS, DLNS does not need a special bit to represent zero exactly, but rather uses $X_D = J$. DLNS has some similarity to redundant LNS [1] and multi-dimensional LNS [22] that involve a definition with addition/subtraction of two exponentials; however unlike those systems, in DLNS one of the exponentials is a constant. The choice of the constant J in (12) is arbitrary; a large negative J restricts the denormal behavior to values close to zero (analogous to IEEE-754); J near 0 makes DLNS like FXNS.

Compared to the symmetrical SLNS representation, where $L \leq X_L < U$, DLNS (with the choice of $J = 0$ in (12)) typically requires one fewer bit than SLNS. DLNS does this at the cost of reducing the relative precision for values near zero. In effect, values near zero are represented with F-bit absolute precision (similar to FXNS); values far from zero are represented with F-bit relative precision (similar to FP and conventional SLNS).

This section describes cases when all the inputs and outputs are in pure-DLNS format. The next section will consider how the cases simplify when some of the inputs are not in pure-DLNS format. The problem of DLNS addition is to find the closest representation to

$$\bar{x} + \bar{y} = (-1)^{X_S} \cdot \left(b^{X_D} - b^J\right) + (-1)^{Y_S} \cdot \left(b^{Y_D} - b^J\right).$$

Just as with conventional SLNS, the hardware has to deal with two cases, a) when the signs of $\bar{x}$ and $\bar{y}$ are the same, and b) when the signs are different (in other words, $X_S = Y_S$ and $X_S \neq Y_S$).

A. Same Signs

Suppose $\bar{x}$ and $\bar{y}$ have the same sign. The sign of the result, $R_S = X_S = Y_S$, will be the same, which allows the sign to be factored out of the computation of the magnitude of the result. There are two alternative ways to derive the computation that the DLNS hardware performs. The first of these performs the addition first, and then converts this back to the DLNS format:

$$\begin{aligned}
\bar{x} + \bar{y} &= (-1)^{R_S} \cdot \left(((b^{X_D} + b^{Y_D}) - b^J) - b^J\right) \\
&= (-1)^{R_S} \cdot \left((b^{X_D}(1 + b^{Y_D}/b^{X_D}) - b^J) - b^J\right) \\
&= (-1)^{R_S} \cdot \left((b^{\log_b(b^{X_D}(1+b^{Y_D}/b^{X_D})} - b^J) - b^J\right) \\
&= (-1)^{R_S} \cdot \left((b^{X_D + s_b(Y_D - X_D)} - b^J) - b^J\right) \\
&= (-1)^{R_S} \cdot \left(b^{R_D} - b^J\right)
\end{aligned}$$

where the actual computation performed by the hardware in this case,

$$\begin{aligned}
R_D &= \log_b(b^{X_D + s_b(Y_D - X_D)} - b^J) \\
&= J + d_b(X_D + s_b(Y_D - X_D) - J),
\end{aligned} \tag{13}$$

uses both s_b and d_b. The commutativity of addition allows interchanging X_D and Y_D and we can make the argument to d_b positive. If $X_D > E_0 + J \approx F + J$, we know (since s_b is always positive) that $X_D + s_b(Y_D - X_D) - J > E_0$ and that the d_b is an essential identity. In that case, this leaves $R_D = J + X_D + s_b(Y_D - X_D) - J = X_D + s_b(Y_D - X_D)$, in other words, the standard SLNS addition algorithm. Just like IEEE-754 (or the messy LNS algorithms in [2] inspired by it), the simple algorithm (13) maintains constant relative precision, except for gradual underflow of "tiny" numbers. The distinction here is that the definition of "tiny" is user configurable with the choice of F and J.

The alternative approach (still for the case when the signs of $\bar{x}$ and $\bar{y}$ are the same) converts one of the representations to SLNS before performing the addition:

$$\begin{aligned}
\bar{x} + \bar{y} &= (-1)^{R_S} \cdot \left((b^{X_D} - b^J) + (b^{Y_D} - b^J)\right) \\
&= (-1)^{R_S} \cdot \left((b^{X_D} + (b^{Y_D} - b^J)) - b^J\right) \\
&= (-1)^{R_S} \cdot \left(b^{R'_D} - b^J\right)
\end{aligned}$$

where the actual computation performed by the hardware in this case is

$$\begin{aligned}
R'_D &= \log_b((b^{X_D} + (b^{Y_D} - b^J)) \\
&= X_D + s_b(J + d_b(Y_D - J) - X_D).
\end{aligned} \tag{14}$$

(14) also uses both s_b and d_b, although in the opposite order from (13). The argument to d_b is positive, unless $Y_D = J$ (which represents $\bar{y} = 0.0$). Since d_b has a singularity, the hardware that computes (14) must return $R'_D = X_D$ in that case. By similar reasoning as with the other alternative, if $Y_D > E_0 + J \approx F + J$, (14) reduces to the standard SLNS addition algorithm. Given that in DLNS, $X_D \geq J$ and $Y_D \geq J$, the two alternatives produce the same result in all

cases, $R_D = R'_D$, assuming that s_b and d_b could be computed precisely.

B. Different Signs

The other case for DLNS addition we must consider is when $\bar{x}$ and $\bar{y}$ have different signs. The sign of the result, T_S, will be the sign of the larger value, which we will assume is $\bar{x}$, i.e., $T_S = X_S$ and Y_S will be the opposite of T_S. Again, there are two ways to derive the computation carried out by the hardware. We could perform the addition of opposite signs (i.e., subtraction of absolute values) first, and then convert this back to the DLNS format:

$$\begin{aligned}
\bar{x} + \bar{y} &= (-1)^{T_S} \cdot \left((b^{X_D} - b^J) - (b^{Y_D} - b^J)\right) \\
&= (-1)^{T_S} \cdot \left((b^{X_D} - b^{Y_D} + b^J) - b^J\right) \\
&= (-1)^{T_S} \cdot \left((b^{X_D}|1 - b^{Y_D}/b^{X_D}| + b^J) - b^J\right) \\
&= (-1)^{T_S} \cdot \left((b^{\log_b(b^{X_D}|1-b^{Y_D}/b^{X_D}|)} + b^J) - b^J\right) \\
&= (-1)^{T_S} \cdot \left((b^{X_D + d_b(Y_D - X_D)} + b^J) - b^J\right) \\
&= (-1)^{T_S} \cdot \left(b^{T_D} - b^J\right)
\end{aligned}$$

where the actual computation performed by the hardware in this case is

$$\begin{aligned}
T_D &= \log_b(b^{X_D + d_b(Y_D - X_D)} + b^J) \\
&= J + s_b(X_D + d_b(Y_D - X_D) - J).
\end{aligned} \tag{15}$$

If $X_D > E_0 + J \approx F + J$, (15) reduces to the standard SLNS algorithm for absolute subtraction.

The other alternative for differing signs is:

$$\begin{aligned}
\bar{x} + \bar{y} &= (-1)^{T_S} \cdot \left((b^{X_D} - b^J) - (b^{Y_D} - b^J)\right) \\
&= (-1)^{T_S} \cdot \left(|(b^{X_D} + b^J) - b^{Y_D}| - b^J\right) \\
&= (-1)^{T_S} \cdot \left(b^{Y_D)}|1 - b^{(J + s_b(X_D - J))}/b^{Y_D}| - b^J\right) \\
&= (-1)^{T_S} \cdot \left(b^{T'_D} - b^J\right)
\end{aligned}$$

where the actual computation performed by the hardware in this case,

$$T'_D = Y_D + d_b(J + s_b(X_D - J) - Y_D), \tag{16}$$

is similar to R'_D, except the roles of X_D and Y_D as well as s_b and d_b have interchanged. Assuming that s_b and d_b could be computed precisely, $T_D = T'_D$. From the above, there are four alternative combinations (R_D/T_D, R_D/T'_D, R'_D/T_D or R'_D/T'_D) of hardware possible.

III. Mixed DLNS Operations

It is apparent from the previous section that DLNS addition involves conversion of one number (either one of the operands or the result) from DLNS format to the conventional SLNS representation. If one of the operands is already available in SLNS format, the operations may simplify.

A. DLNS plus SLNS Add

Suppose that rather than to start with two given DLNS inputs (X_D and Y_D), the addition hardware inputs are X_D and Y_L, the latter being the conventional SLNS representation of $\bar{y}$. The desired result is then simpler for the $X_S = Y_S$ case,

$$\begin{aligned}
\bar{x} + \bar{y} &= (-1)^{R_S} \cdot \left((b^{X_D} - b^J) + b^{Y_L}\right) \\
&= (-1)^{R_S} \cdot \left(b^{X_D} + b^{Y_L}\right) - b^J \\
&= (-1)^{R_S} \cdot \left(b^{R'_D} - b^J\right),
\end{aligned}$$

as is the actual computation performed by the hardware,

$$R''_D = \log_b(b^{X_D} + b^{Y_L}) = X_D + s_b(Y_L - X_D). \tag{17}$$

More importantly, this (DLNS+SLNS yields DLNS) case is identical to what would have happened for the conventional (SLNS+SLNS yields SLNS) case.

In a similar way, when $X_S \neq Y_S$, the hardware computation for the DLNS+SLNS yields DLNS case is:

$$T''_D = X_D + d_b(Y_L - X_D). \tag{18}$$

This also identical to what would have happened for the conventional (SLNS+SLNS yields SLNS) case when $X_S \neq Y_S$.

B. DLNS by SLNS Multiply

Multiplication of two DLNS values is a difficult operation involving conversion of both operands; it is better if one of the operands can already be in SLNS format. In many signal-processing systems, the multiplier is either constant or is reused many times (and may be brought into a register). As with SLNS, the sign of the product is simply the exclusive OR or the input sign bits. Assuming W_L is the SLNS multiplier, and Y_D is the DLNS multiplicand,

$$\begin{aligned}
|\bar{w} \cdot \bar{y}| &= b^{W_L}\left(b^{Y_D} - b^J\right) \\
&= b^{W_L} b^{J + d_b(Y_D - J)} + b^J - b^J \\
&= \left(b^{W_L + J + d_b(Y_D - J)} + b^J\right) - b^J \\
&= b^{P_D} - b^J
\end{aligned}$$

where the hardware computation,

$$P_D = J + s_b(W_L + d_b(Y_D - J)) \tag{19}$$

seems similar to the computations required for DLNS+DLNS yields DLNS cases described in Section II.

C. A combined DLNS/SLNS ALU

The similarity of (19) to the computations in Section II suggests that a single ALU design could have the ability to perform pure-DLNS addition/subtraction, DLNS-by-SLNS multiplication, as well as pure-SLNS addition/subtraction. Trying to combine all of these into a single circuit will suggest that some of the alternatives described in Section II are less efficient than others. For example, when merging R and T' into a single circuit, it is not possible to implement (19) easily with that circuit. The R/T and R'/T' combinations have an undesirable structure (s_b and d_b units whose inputs and outputs are connected to multiplexors with the complication that one input of each input multiplexor is connected to the output multiplexor). This statically appears to be a feedback path

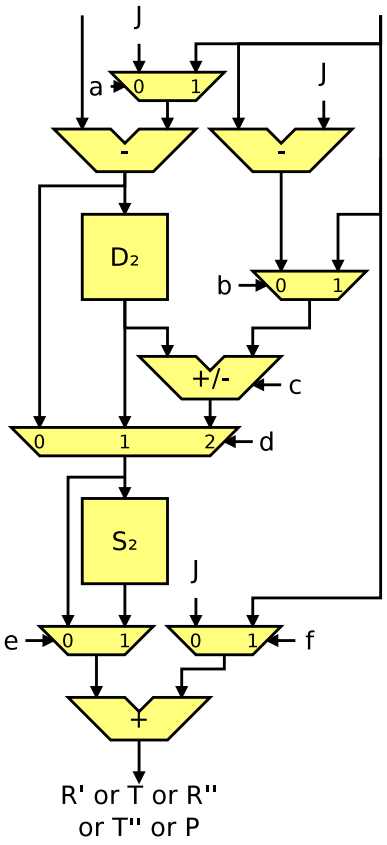

Fig. 1. DLNS ALU.

TABLE I
VALUE OF CONTROL SIGNALS AS A FUNCTION OF THE DESIRED OUTPUT. "X" STANDS FOR ANY VALUE (*don't care*).

Signal	R'_D	T_D	R''_D	T''_D	P_D
a	0	1	1	1	0
b	0	0	X	X	1
c	-	+	X	X	+
d	2	2	0	1	2
e	1	1	1	0	1
f	1	0	1	1	0

requiring a register, although dynamically it resolves to be combinatorial logic (rather like the behavior of an end-around-carry adder). While these R/T or R'/T' circuit combinations could work, the false path will complicate use of synthesis tools. This leaves the preferred combination of R' from (14) and T from (15), which is implemented by the circuit in Figure 1. Table I gives the select inputs to the multiplexors that allow this one circuit to compute R', T, P, R'' and T''.

D. DLNS by SLNS Multiply/Accumulate

The three-operand multiply-accumulate operation, $w \cdot y + x$, is common in many applications. In signal processing, it frequently occurs in situations where the same w is used with different values of x and y, suggesting w could be stored in SLNS format, with x and y in DLNS format. In this case, treating multiply-accumulate as an atomic operation (rather than as a multiply followed by an addition) allows considerable simplification:

$$\begin{aligned} |\bar{w} \cdot \bar{y} + \bar{x}| &= b^{W_L} \cdot \left(b^{Y_D} - b^J\right) + \left(b^{X_D} - b^J\right) \\ &= \left(b^{J+W_L+d_b(Y_D-J)} + b^{X_D}\right) - b^J. \end{aligned}$$

As with pure-DLNS addition, there are two cases, depending on signs. If the sign of $\bar{w} \cdot \bar{y}$ is the same as the sign of $\bar{x}$, the result is

$$\begin{aligned} |\bar{w} \cdot \bar{y} + \bar{x}| &= b^{X_D} \cdot \left(b^{J+W_L+d_b(Y_D-J)}/b^{X_D} + 1\right) - b^J \\ &= b^{P_D} - b^J, \end{aligned}$$

where the hardware computation is

$$P'_D = X_D + s_b(J + W_L + d_b(Y_D - J) - X_D). \tag{20}$$

If the sign of $\bar{w} \cdot \bar{y}$ is different than the sign of $\bar{x}$, the hardware computation is

$$Q_D = X_D + d_b(J + W_L + d_b(Y_D - J) - X_D). \tag{21}$$

IV. ANALYSIS AND SIMULATION

Unlike SLNS, the relative precision in DLNS varies with the magnitude of the value being represented in relation to the designer's choice of b^J. Given one exactly-represented-DLNS point, $|\bar{x}|$, the internal value processed by logarithmic hardware would look like $|\bar{x}| + b^J$. Such internal hardware is subject to the same relative spacing as conventional F-bit SLNS, and so the value of the next larger exactly-represented-DLNS point is $\beta(|\bar{x}| + b^J) - b^J$. From this we see the absolute spacing of the adjacent points is $(\beta - 1)(|\bar{x}| + b^J)$ and for $|\bar{x}| \geq b^J$ the relative spacing is

$$\frac{(\beta - 1)(|\bar{x}| + b^J)}{|\bar{x}|}. \tag{22}$$

For $|\bar{x}| < b^J$, DLNS naturally underflows to the representation $|\bar{x}| = 0.0$, and hence (22) is undefined.

The Fast Fourier Transform (FFT) is a common signal-processing algorithm, often implemented with both fixed- and floating-point arithmetic. It has also been extensively studied in the context of SLNS [26], [18], [3], [13]. We implemented an FFT using actual SLNS (with a wide enough dynamic range that underflow does not occur) and our proposed DLNS $b = 2$ arithmetics. Figure 2 shows the RMS error for a 64-point radix-two FFT whose input is a real-valued 25% duty-cycle square wave plus complex white noise. (We obtained similar figures for larger size FFTs.) This code was simulated 100 times with different pseudo-random noise. Using the same initial random data, the simulation computes several results: a double precision result which, for practical purposes, is regarded as "exact"; DLNS results for $8 \leq F \leq 13$ and $-20 \leq J \leq 0$; and SLNS results for $8 \leq F \leq 13$, shown in the last column. For J near 0, the RMS appears to depend only on the choice of J. When $J < -E_0$, the RMS for DLNS appears asymptotic to the RMS for the F-bit underflow-free

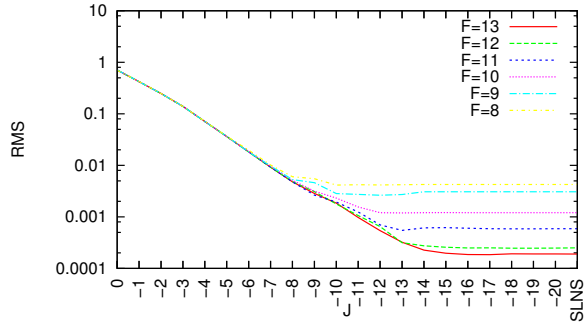

Fig. 2. 64-point FFT RMS using actual DLNS and SLNS arithmetic.

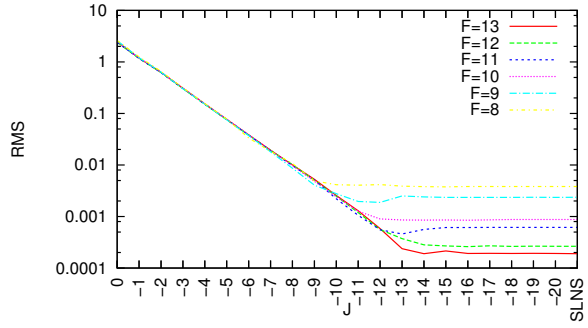

Fig. 3. 64-point FFT RMS using abrupt-underflow SLNS arithmetic.

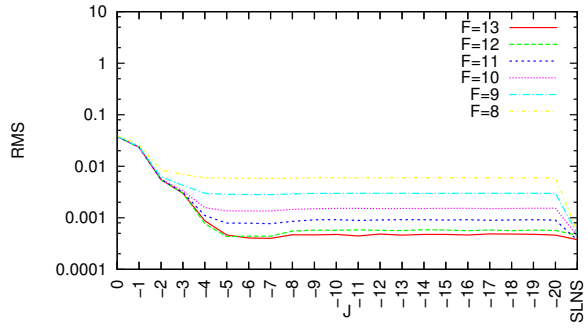

Fig. 4. 64-point FFT RMS using random error model (22).

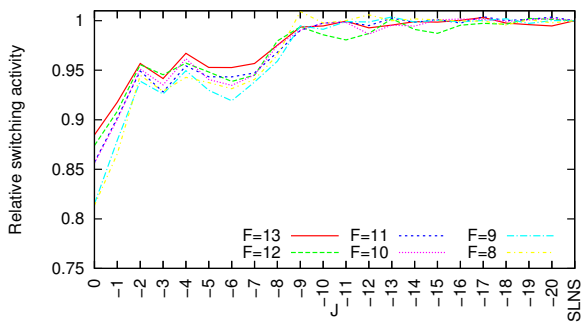

Fig. 5. DLNS FFT Switching Activity (% SLNS) using Two's Complement X_D.

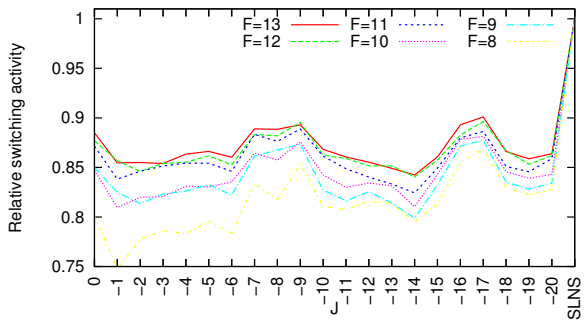

Fig. 6. DLNS FFT Switching Activity (% SLNS) using Offset X_D.

SLNS.

For comparision, instead of a simple underflow-free SLNS, we modeled an SLNS which abruptly underflows at b^{-J}. Figure 3 shows the RMS error for the same FFT simulation using this abrupt-underflow SLNS. The shape of the curves in Figures 2 and 3 are similar, reaching similar asymptotes; however, for J near zero, DLNS is two to three times more accurate.

We also modeled the DLNS error mechanism more abstractly by injecting noise into each double-precision-FFT step from a random distribution whose width is given by (22). Although Figure 4 is noisy and overestimates the error, it appears similar to the actual simulation results in Figure 2, suggesting (22) is a reasonable model for DLNS behavior.

In some applications, Paliouras and Stouraitis [25] have shown SLNS reduces dynamic power consumption of memory accesses because of decreased switching activity on the memory bus resulting from the compression inherent in the logarithmic representation. To see whether DLNS has similar advantages, we measured switching activity during the memory access pattern of our FFT simulation using actual DLNS arithmetic, and also, for comparision, using SLNS arithmetic. The data are plotted in Figure 5 as a percentage of SLNS switching activity. As is most natural, Figure 5 uses two's complement integers to represent the DLNS X_D and SLNS X_L. This means negative X_D represents absolute values less than $1 - b^J$; for $J = 0$, $X_D \geq 0$, which is significant since one of the major causes of increased switching activity is alternating between positive and negative two's complement values in memory. Values of J near zero offer up to 15% reduction in switching activity; $J = -F$ yields a 3% reduction in switching activity. As J moves further away from zero, the switching activity becomes similar to SLNS.

An alternative to two's complement negative X_D is to use an offset (by J) representation for X_D, analogous to how IEEE-754 exponents are encoded. Figure 6 shows this offers switching reduction over a wide range of J. It reaches 15% reduction for $J = -8, F = 8$ and nearly 25% reduction for $J = 1, F = 8$. It is also possible to use offset representation for abrupt-underflow SLNS. Figure 7 shows this offers less switching reduction (around 10%) than DLNS, and, as described earlier, this comes at a cost of greater RMS error than DLNS.

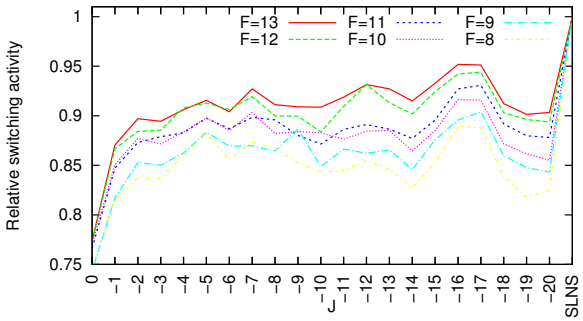

Fig. 7. Abrupt-Underflow SLNS FFT Switching Activity (% SLNS) using Offset X_L.

To measure software-implementation cost, a 32-bit ($F = 23$) C++ implementation of abrupt-underflow LNS (using interpolation and cotransformation with range and precision comparable to IEEE-754 single precision) was extended to DLNS using R and T. A simple computation (Taylor series for $e \cdot 2^k$, where $-45 \leq k \leq -25$) was benchmarked on a 1.3GHz Core 2 Duo, using g++ and Microsoft compilers. DLNS only adds around 16% overhead with purely normal data, because these pass through the extra s_b or d_b as essential identities. As k moves into the denormal range, the speed of DLNS can be as much as 2.5 times slower than LNS. This benchmark also allows comparison of errors between FP, LNS and DLNS. As k decreases, FP and DLNS errors are very similar (gradually increasing). In contrast, LNS experiences large errors when 2^k underflows.

V. Synthesis

We implemented two versions of the proposed DLNS/SLNS ALU designs inside the FloPoCo arithmetic core generator framework. FloPoCo [10] is a software tool that automatically generates arithmetic cores in synthesizable VHDL. It includes support for SLNS arithmetic. Our first ALU implementation computes T, R', R''_D, T''_D and P_D, and our second implementation additionally supports multiply-accumulate operations P'_D and Q_D. We leverage the implementations of s_b and d_b that FloPoCo provides for SLNS. The implementation of s_b is based on an optimized polynomial evaluator [11] and d_b is evaluated using co-transformation [4].

We synthesized both units for a Xilinx Virtex-4 LX-25 FPGA using the Xilinx ISE 12.3 synthesis toolchain. Table II shows the area in FPGA slices and DSP blocks and the combinatorial latency in nanoseconds, for various precisions. These results are compared with the resources taken by s_b and d_b alone, a valid point of comparison for typical applications where the signs of numbers are not known. As can be seen, s_b and d_b account for most of the area and delay. The overheads added by the combined ALU and multiply-accumulate ALU over a conventional SLNS ALU are respectively 26% and 43% in the worst case (for $F = 10$).

VI. Conclusions

This paper has introduced the Denormal Logarithmic Number System (DLNS), which is a hybrid of the properties of FXNS and SLNS. The proposed algorithms are characterized in terms of base (typically $b = 2$), precision (F) and a new design parameter, J, which allows customizing the range in which gradual underflow occurs. $J = 0$ gives a wide gradual underflow range that makes DLNS act like FXNS (and like the μ law which inspired DLNS); $J < -F$ gives a narrow gradual underflow range that makes DLNS act like SLNS (and like the IEEE-754 standard which also inspired DLNS). Simulation of an FFT application illustrates $J \approx 0$ decreases bit-switching activity 15% with a two's complement encoding and nearly 25% with an offset representation; however, this causes significant increase in RMS error. A choice of $J = -F$ provides a balanced design point, decreasing bit-switching activity by 15% with an offset representation at the cost of a 30% increase in RMS error. DLNS reduces switching activity 5% to 20% more than an abruptly-underflowing SLNS with around one-half the RMS error. The majority of the area of the synthesized DLNS circuit is for traditional SLNS addition and subtraction tables; only a small area is used for the novel datapaths that allow the ALU to act on conventional SLNS as well as DLNS and mixed data.

References

[1] M. G. Arnold, T. A. Bailey, J. R. Cowles and J. J. Cupal, "Redundant Logarithmic Arithmetic," *IEEE Trans. Comput.*, vol. 39, pp. 1077–1086, Aug. 1990.

[2] 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.

[3] M. Arnold and C. Walter, "Unrestricted Faithful Rounding is Good Enough for Some LNS Applications," *15th Intl. Symp. Computer Arithmetic*, Vail, Colorado, pp. 237-245, 11-13 June 2001.

[4] M. Arnold and C. Collange, "A Dual-Purpose Real/Complex Logarithmic Number System ALU," *19th Intl. Symp. Computer Arithmetic*, Portland, OR, pp. 15-24, 8-10 June 2009.

[5] M. Arnold, et al. "Towards a Quaternion Complex Logarithm Number System," *20th Intl. Symp. Computer Arithmetic*, Tuebingen, Germany, pp. 33-42, 25-27 July 2011.

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

[7] E. I. Chester and J. N. Coleman, "Matrix Engine for Signal Processing Applications Using the Logarithmic Number System," *Proceedings of the IEEE Intl. Conf. on Application-Specific Systems, Architectures and Processors*, San Jose, California, pp. 315-324, 17-19 July 2002.

[8] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlac, "Arithmetic on the European Logarithmic Microprocessor," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 702-715, July 2000.

[9] J. N. Coleman, C. I. Softley, J. Kadlec, R. Matousek, M. Tichy, Z. Pohl, A. Hermanek, and N. F. Benschop, "The European Logarithmic Microprocessor", IEEE Transactions on Computers, pp. 532-546, 2008.

[10] F. de Dinechin, "The Arithmetic Operators You Will Never See in a Microprocessor", *20th Intl. Symp. Computer Arithmetic*, Tuebingen, Germany, pp, 189-190, July 2011.

[11] F. de Dinechin, M. Joldes and B. Pasca, "Automatic Generation of Polynomial-Based Hardware Architectures for Function Evaluation", *Application-specific Systems, Architectures and Processors*, IEEE, 2010.

[12] A.D. Edgar and S.C. Lee, "FOCUS Microcomputer Number System," *Commun. of the ACM*, vol. 22, p. 166-167, 1979.

[13] 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.

TABLE II

AREA AND COMBINATORIAL DELAY OF SYNTHESIZED OPERATORS. AREA IS GIVEN IN GENERAL-PURPOSE SLICES AND DSP48 SLICES. THE AREA AND DELAY OF THE s_b AND d_b PARTS SYNTHESIZED SEPARATELY ARE SHOWN FOR REFERENCE. FOR ALL ENTRIES, $J = 3$.

F	Combined			s_b and d_b			Multiply-accumulate			d_b and s_b-d_b		
	Slices	DSPs	ns	Slices	DSPs	ns	Slices	DSPs	ns	Slices	DSPs	ns
8	895	0	35.7	739	0	30.9	1184	0	42.90	892	0	38.6
9	1105	0	39.3	857	0	31	1414	0	46.70	1086	0	40.4
10	1177	3	42.3	936	2	32.8	1694	3	50.72	1182	2	40
11	1462	3	43.0	1226	2	34.9	1975	3	52.72	1686	2	42.2
12	1665	6	46.4	1479	6	37.8	2602	6	54.18	1824	6	46.2
13	2063	6	46.8	1740	6	39.5	3279	6	56.53	2366	6	53.4
14	2356	6	44.6	2158	6	37.8	4209	6	57.69	3004	6	47.4

[14] K. F. Gauss, *Werke*, vol. 8, pp. 121-128, 1900.

[15] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, IEEE, 1985.

[16] IEEE Standard for Floating-Point Arithmetic, ANSI/IEEE Std 754-2008, IEEE, 2008.

[17] R. C. Ismail and J. N. Coleman, "ROM-less LNS", *20th Intl. Symp. Computer Arithmetic*, Tuebingen, Germany, pp. 43-51, July 2011.

[18] S. J. Kidd, "Implementation of the Sign-Logarithm Arithmetic FFT,"*Royal Signals and Radar Establishment Memorandum 3644*, Malvern, 1983.

[19] N. G. Kingsbury and P. J. W. Rayner, "Digital Filtering Using Logarithmic Arithmetic," *Electron. Lett.*, vol. 7, no. 2, pp. 56-58, Jan 28, 1971.

[20] M. Kahrs and K. Branderburg, Editors., *Applications of Digital Signal Processing to Audio and Acoustics*, Kluwer Academic Publ., Norwell, Massachusetts, p. 224, 1998.

[21] D. M. Lewis, "114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications," *Intl. Solid-State Circuits Conf.*, San Francisco, pp. 1547-1553, Feb. 1995.

[22] V. S. Dimitrov, J. Eskritt, L. Imbert, G. A. Jullien and W. C. Miller, "The Use of The Multi-Dimensional Logarithmic Number System in DSP Applications," *15th Intl. Symp. Computer Arithmetic*, Vail, Colorado, pp. 247-254, 11-13 June 2001.

[23] I. Kouretas, Ch. Basetas and V. Paliouras, "Low-Power Logarithmic Number System Addition and Subtraction and their Impact on Digital Filters," *IEEE Trans. Comput.*, 29 May 2011, IEEE Computer Society Digital Library, http://doi.ieeecomputersocety.org/10.1109/TC.2012.111

[24] Junichiro Makino and Makoto Taiji, *Scientific Simulations with Special-Purpose Computers—the GRAPE Systems*, John Wiley and Sons, Chichester, England, 1998.

[25] V. Paliouras and T. Stouraitis, "Low Power Properties of the Logarithmic Number System," *Proceedings of the 15th IEEE Symp. on Computer Arithmetic*, Vail, Colorado, pp. 229-236, 11–13 June 2001.

[26] E. E. Swartzlander, D. Chandra, T. Nagle, and S. A. Starks, "Sign/logarithm Arithmetic for FFT Implementation," *IEEE Trans. Comput.*, vol. C-32, pp. 526-534, 1983.

[27] E. E. Swartzlander and A. G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Trans. Comput.*, vol. C-24, pp. 1238–1242, December 1975.

[28] S. Young, et al., *The HTK Book (for HTK Version 3.1)*, Cambridge University Engineering Department, England, Dec. 2001. `http://htk.eng.cam.ac.uk`

[29] `www.xlnsresearch.com` has an extensive bibliography of LNS-related articles.

[30] "Pulse Code Modulation (PCM) of Voice Frequencies", International Telecomunications Union, 1988. `http://www.itu.int/rec/T-REC-G.711/en`