The use of the multi-dimensional logarithmic number system in DSP applications

V. S. Dimitrov, J. Eskritt, L. Imbert, G. A. Jullien and W.C. Miller

VLSI Research Group, University of Windsor
Windsor, Ontario, Canada N9B 3P4
(519) 253-3000 Ext. 2581
vassil@vlsi.uwindsor.ca

ABSTRACT

A recently introduced double-base number representation has proved to be successful in improving the performance of several algorithms in cryptography and digital signal processing. The index-calculus version of this number system can be regarded as a two-dimensional extension of the classical logarithmic number system. This paper builds on previous special results by generalizing the number system both in multiple dimensions (multiple bases) and by the use of multiple digits. Adopting both generalizations we show that large reductions in hardware complexity are achievable compared to an equivalent precision logarithmic number system.

1. INTRODUCTION

The logarithmic number system (LNS) [1][2][3] is an alternative to the binary representation and it has been a subject of some investigation [4][5], particularly in the field of digital signal processing (DSP) [6][7], where the computation of inner (dot) products is a major computational step. In the LNS, multiplication and division are easy operations, whereas addition and subtraction are difficult operations, traditionally implemented by making use of large ROM arrays [8][9].

Inner products computed in DSP algorithms are often between a predetermined set of coefficients (e.g., FIR filters or discrete transform basis functions) and integer data. For fixed-point binary implementations, the uniform quantization properties[1] are perfectly matched to the mapping of most input data (the mapping of input data for non-linear hearing instrument processing is a counter-example), but often the predetermined coefficients are better suited to a non-uniform quantization mapping. A study of a large number of filter designs reveals a histogram that benefits from the quantization associated with a logarithmic mapping [10].

A logarithmic-like representation, referred to as the index calculus double-base number system (IDBNS), was recently introduced [11] that promises implementation improvements over the LNS while maintaining a logarithmic quantization distribution, and there have been several papers published on special results from this number representation [12]-[17]. In this paper we will generalize the number system and present several new results that demonstrate the efficiencies in using this representation over the classical LNS for typical DSP computations.

The IDBNS is based on a single-digit representation of the form:

$$y = s2^b3^t \tag{1}$$

where $s \in \{-1, 0, 1\}$ and b, t are signed integers. In this case we have the following theorem [12]:

Theorem 1: For every $\varepsilon > 0$ and every non-negative real number x, there exist a pair of integers a and b, such that the following inequality holds:

$$\left|x - 2^a3^b\right| < \varepsilon .$$

❑

We may therefore approximate, to arbitrary precision, every real number with the triple $\{s, b, t\}$. We may look at this representation as a two-dimensional generalization of the binary logarithmic number representation. The important advantage of this generalization is that the binary and ternary indices are operated on independently from each other, with a potential reduction in complexity of the implementation hardware. As an example, a VLSI architecture for inner product computation with the IDBNS, proposed in [12][13], has an area complexity dependent entirely on the dynamic range of the ternary exponents. We can capitalize on this complexity dependency by placing design constraints on the ternary exponent size. For example, if we want to represent digital filter coefficients in the IDBNS, then we can design the coefficients in such a way that the ternary exponent is minimized - an integer programming task [14]. Although this approach is sound, and can produce modest improvements, generalizing the representation to multi-dimensions and/or multiple digits has the potential to bring about very large reductions in hardware complexity of DSP implementations.

In this paper we provide new results for both the quantiza-

[1] A constant error bound over all mapped input values

tion of input data as well as implementation efficiencies using multiple dimensions and digits in this index calculus.

2. MATHEMATICAL PRELIMINARIES

There are some well-established results on *s*-integers that we can build upon. We start with two basic definitions [18].

Definition 1: An *s*-integer is a number which largest prime factor does not exceed the *s*-th prime number.

For example, non-negative powers of two are 1-integers, numbers of the form $2^a 3^b$, a, b - non-negative integers, are 2-integers and so on.

Definition 2: Modified 2-integers are numbers of the form $2^a p^b$, p - odd integer.

Note that we do not impose restriction on the sign of *a* and *b* in Definition 2.

The next definition offers the most general representation scheme we will consider in this paper.

Definition 3: A representation of the real number x in the form:

$$x = \sum_{i=1}^{n} s_i \prod_{j=1}^{b} p_j^{e_j^{(i)}} \qquad (2)$$

where $s_i \in \{-1, 0, 1\}$ and $p_j, e_j^{(i)}$ are integers, is called *a multidimensional n-digit logarithmic* (*MDLNS*) *representation* of *x*. *b* is the number of bases used (at least two, the first one, that is, p_1, will always be assumed to be 2 in this paper).

The next two definitions are special cases of Definition 3; the representation schemes defined by them will be used extensively in the paper.

Definition 4: An approximation of a real number x as a signed modified 2-integer $s2^a p^b$, is called *a two-dimensional logarithmic representation of x*.

Definition 5: An approximation of a real number x as a sum of signed modified 2-integers $\sum_{i=1}^{n} s_i 2^{a_i} p^{b_i}$ is called *an n-digit two-dimensional logarithmic representation of x.* $n = 2$ will be a special case.

It is important to note that an extension of the classical LNS to a multi-digit (or multi-component) representation *does not provide* any inherent advantages in terms of complexity reduction. We can show this for the special 2-digit 2-dimensional case with the following two theorems.

Theorem 2: Let x be an integer with the following 2-digit 2-base logarithmic approximation $x \approx s_1 2^{a_1} p^{b_1} + s_2 2^{a_2} p^{b_2}$. Then for $\left| x - (s_1 2^{a_1} p^{b_1} + s_2 2^{a_2} p^{b_2}) \right| < 0.5$, the dynamic range of $max(b_1, b_2)$ is $0.5 \cdot log(x) + o(log(x))$.

❑

Arnold at al. were the first to consider a similar representation scheme [4] in the case of classical LNS (we shall call it a 2-component LNS). Although it leads to some reduction of the exponents dynamic range (correspondingly, reduction of the ROM sizes), the number of bits required by the larger exponent to store the integer number x is approximately $log(x)$. The storage reduction in the 2-component LNS (as opposed to the 1-component LNS) comes from the observation that in the 1-component LNS one needs approximately $log(x) + log\,log(x)$ bits to encode *x*.

Theorem 3: Let x be an integer with a 2-component LNS representation $x \approx s_1 2^{l_1} + s_2 2^{l_2}$.

Then for $\left| x - (s_1 2^{l_1} + s_2 2^{l_2}) \right| < 0.5$ the dynamic range of $max(l_1, l_2)$ is $log(x) + O(1)$.

❑

We have omitted the proofs of the above two theorems for brevity.

We can see from Theorems 2 and 3, that the dynamic range of the exponents is reduced by a factor of 2 for the 2DLNS, but for the 2-component LNS system there is no reduction at all. Since we will demonstrate that hardware complexity for the MDLNS is exponentially dependent on the size of the nonbinary-base(s) exponent, we clearly have a potential for quite considerable hardware reduction providing that the following inequality is met: $max(|b_1|, |b_2|) \ll max(|l_1|, |l_2|)$.

3. INPUT DATA MAPPING

3.1 Error-Free Representations

As stated in the introduction, most often in DSP applications the input data has to be converted from analog to a fixed-point binary value with a uniform quantization error bound. Mapping to integers has a quantization error bounded by –0.5 for all converted values. For a classical LNS representation (and also a 1-digit MDLNS representation) we do not have this uniform quantization accuracy so we have to choose a sufficient number of bits so that we will be able to maintain this conversion accuracy for the larger data values. In the multi-digit MDLNS we can mitigate this quantization problem; in fact, we can find certain MDLNS representations that are completely error free!

Consider the case of the two odd prime bases, (3, 5).

A representation of a real number into forms given in definitions 3 to 5 is called *error-free* if there is zero approximation error. The next three theorems and one conjecture provide new results about the error-free two-dimensional logarithmic representation of numbers.

Theorem 4: Every real number x may have at most 91 different error-free 2-digit two-dimensional logarithmic representations.

Proof:

Let us assume that x is represented in the form of *Definition 5*:

$$x = \pm 2^{a} p^{b} \pm 2^{c} p^{d} \quad (3)$$

Clearly, x must be a rational number. Now we multiply the two sides of (1) by $z = 2^{-min(a, c, 0)} p^{-min(b, d, 0)}$. The left and right sides of the new equation will be integers. Divide by the greatest common divisor of $(2^{a - min(a, c, 0)} p^{b - min(b, d, 0)}, 2^{c - min(a, c, 0)} p^{d - min(b, d, 0)})$ Let us denote the left side of the equation obtained as M. We may obtain only two types of equations:

$$M = \pm 1 \pm 2^{e} p^{f} \quad (4)$$

or

$$M = \pm 2^{e} \pm p^{f} \quad (5)$$

eqn. (4) may have at most one solution, due to the fundamental theorem of arithmetic. eqn. (5) can be treated like this. We represent the exponents e and f with respect to modulo 3: $e = 3e_1 + e_2$ and $f = 3f_1 + f_2$, $e_2, f_2 \in \{0, 1, 2\}$. For the nine possible combinations of residues (e_2, f_2) we have nine Diophantine equations of the form:

$$M = \pm A 2^{3e_1} \pm B p^{3f_1} \quad (6)$$

where $A \in \{1, 2, 4\}$ and $B \in \{1, p, p^2\}$. We substitute $X = 2^{e_1}$ and $Y = p^{f_1}$. The final equation we have is:

$$M = c_1 X^3 + c_2 Y^3 \quad (7)$$

where c_1 and c_2 are constants. Tue s theorem [19] about the cubic Diophantine equations states that the equation $c_1 x^3 + c_2 x^2 y + c_3 x y^2 + c_4 x^3 = c_5$ may have at most five different solutions in integers. If M is positive, then the only possible combinations of signs in eqn. (5) are: (+,+), (+,-); if M is negative, the choices are (-,-), (+,-). Therefore, we have 18 equations of type (5), that is, we may have at most 90 possible solutions of eqn. (5). Therefore, the total number of possible error-free 2-digit two-dimensional logarithmic representation of a given real number is bounded from above by 91 (at most one for eqn. (4) and at most 90 for eqn. (5)).

❑

The upper bound proved in Theorem 4 can certainly be improved. We have not found any real number with more than five error-free 2-digit LNS representations; here we report one case having exactly five error-free representations.

Let x=3.25; then x can be represented with no error in a 2-digit 2-D LNS with odd base 3 as follows:

$3.25 = (1, -2, 2, 1, 0, 0)$

$3.25 = (1, 0, 1, 1, -2, 0)$

$3.25 = (1, 2, 0, -1, -2, 1)$

$3.25 = (1, 1, 1, -1, -2, 2)$

$3.25 = (1, 6, 0, -1, -2, 5)$

The point of the theorem is to establish an effectively computable upper bound that could be a starting point for improvements. The example given shows that the lower bound for the maximal number of error-free representation is five.

Theorem 5: The smallest positive integer with no error-free 2-digit two-dimensional LNS representation in the case of odd base three is 103.

Theorem 6: The smaller positive integer with no error-free 2-digit two-dimensional logarithmic representation in the case of odd base five is 43.

We have omitted the proofs for brevity.

The following conjecture is based on extensive numerical calculations.

Conjecture 1: The smallest positive integer with no error-free 3-digit two-dimensional logarithmic representation in the case of odd base three is 4985. Or, in the language of the exponential Diophantine equations, it can be posed as follows - the equations $\pm 2^{a} 3^{b} \pm 2^{c} 3^{d} \pm 2^{e} 3^{f} = 4985$ do not have solutions in integers.

It is important to note that such results will be available (and different) for every particular set of bases that we choose. In this case (that is, a 3-digit two-dimensional logarithmic representation with odd base three) we see that a 12-bit error-free mapping is available; a useful dynamic range for many DSP applications. It should also be noted that the size of the exponents used (a, b, c, d, e, f) is only 3-bit unsigned integers.

3.2 Non Error-Free Representations

Clearly, error-free representations are special cases of the

MDLNS, but the extra degree of freedom provided by the use of multiple digits can mitigate the non-uniform quantization properties of the classical LNS.

To illustrate this, we present numerical results for mapping 10-bit signed binary input data to the 2-digit 2-D LNS where we treat the odd base as a parameter. In order to demonstrate the ability to closely match input data with very small exponents, we have restricted the odd base exponent to 3-bits. We are allowing the binary exponent to be somewhat larger, namely 6-bits, but, as we will see in the next section, this has very little bearing on the overall complexity of the inner product implementation (i.e., the hardware complexity is mainly driven by the dynamic range of the ternary exponents). As stated above, we require quantization errors to be <0.5 in order to match a binary representation. Figure 1 shows the results for parameters in the set {3,5,7,11,13,15,17,47}. The scale for each graph is between 0.5 and 1 for the absolute error. There are two observations: 1) there are very few values in any of the results where the error exceeds 0.5; 2) there appears to be no correlation between the density of errors >0.5 and the value of the odd base.

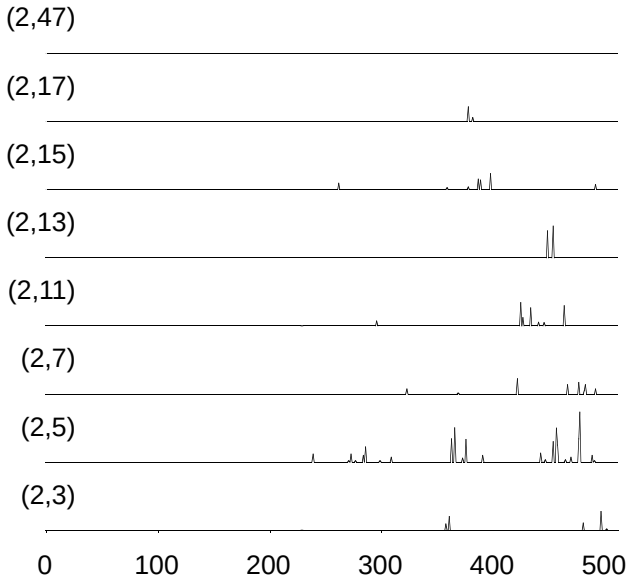

Figure 1. Absolute error (>0.5) for different odd bases

For an odd-base of 47 we find no errors that exceed 0.5, whereas for *integer* bases below this value we find some errors.

To compare these results with an implementation using a classical LNS representation, we need to determine the number of bits of the logarithm to produce an absolute error of <0.5. A previous study has found that we require $n + \lceil log_2 n \rceil$ bits for the logarithm in order to achieve this accuracy for an n-bit positive number [20]. If we assume that the hardware complexity of the classical LNS representation is driven by the number of bits in the logarithm, then we can see a potential for reduction in the implementation complexity of the 2-digit 2-D LNS versus the classical LNS.

3.3 Data Conversion and Approximations to Unity

A fundamental difference between the classical LNS and multi-dimensional LNS is the possibility to find non-trivial approximations of unity in the MDLNS. These can be used to constrain the dynamic range of exponents during general computations. Unity approximants also play a major role in the conversion process, as discussed below.

The conversion process from a binary to MDLNS representation, in its simplest form, is a look-up table. This is certainly an efficient hardware solution for fixed point mapping up to about 13 bits. For larger dynamic ranges we have to find alternate solutions. Two recent papers on the conversion process [16][17] have identified a form of successive approximations technique that has been demonstrated to be efficient for large numbers of bits (>20). A fundamental concept in this process is the identification of close approximations to unity. From Theorem 1, we know that one can be approximated with arbitrary precision as a 2-integer. In fact, *both* bases can be changed and the theorem will still remain valid. Here we expand the discussion of these approximants within the MDLNS, and introduce new results.

As an example with 8-bit exponents, consider the generation of a sequence of successive values of possible 1-digit MDLNS (2-integer) values. Table 1 shows a small subset of values (around unity) obtained from such a sequence.

The fourth and fifth columns show the difference between successive binary and ternary exponents within the sequence. In this small subset of the complete sequence we observe that the differences are limited to only 3 sets of 2-integers with indices $(233, -147)$, $(149, -94)$ and $(-84, 53)$. Each of the 2-integers represents a close approximation to one, multiplication by which generates the next value in the sequence. In Table 2 we have rearranged the sequence from 2-integer values of 1/3 to 1 sorted by increasing values of the binary index. We observe that there is a complete set of binary exponents in the range [-128,128) and the ternary exponents form a complete sub-range [-81,81). The ternary exponent sub-range arises from the fact that $81 \approx 128/(log_2 3)$. If we consider the sub-range from 1/6 to 1/3 we will also have a complete binary exponent sequence and a continuous descending sequence of ternary exponents, except that the ternary exponents will be decremented by one. In fact there will be $256 - 160 = 94$ such sequences covering a range $3^{94} > 2^{148}$ which is useful dynamic range

for many DSP applications.

Table 1: Near Unity Approximants

b	t	2^b3^t	$b_n - b_{n-1}$	$t_n - t_{n-1}$
122	-77	0.971231719	233	-147
38	-24	0.973261899	-84	53
-46	29	0.975296322	-84	53
103	-65	0.984482491	149	-94
19	-12	0.986540369	-84	53
-65	41	0.988602548	-84	53
84	-53	0.997914046	149	-94
0	0	1	-84	53
-84	53	1.002090314	-84	53
65	-41	1.011528852	149	-94
-19	12	1.013643265	-84	53
-103	65	1.015762098	-84	53
46	-29	1.025329408	149	-94
-38	24	1.027472668	-84	53
-122	77	1.029620409	-84	53

Table 2: Partial sequence sorted by binary exponent

b	t	2^b3^t
-128	80	0.43437111
-127	80	0.86874222
-126	79	0.57916148
-125	78	0.386107653
-124	78	0.772215307
-123	77	0.514810204
-122	76	0.343206803
…	…	…
-3	1	0.375
-2	1	0.75
-1	0	0.5
0	-1	0.333333333
0	0	1
1	-1	0.666666667
2	-2	0.444444444
3	-2	0.888888889
4	-3	0.592592593
…	…	…
122	-77	0.971231719
123	-78	0.647487813
124	-79	0.431658542
125	-79	0.863317084
126	-80	0.575544723
127	-81	0.383696482

An interesting result from an exhaustive search of such subsequences with exponent sizes between 5 and 16 bits, is that there are only 3 different approximants of unity used to generate each sequence. We are working on the theoretical explanation of this observed result.

The usefulness of the existence of good approximations of one, for general computations within dynamic constraints on the exponents, can be seen from the following example:

Example 1: Calculate x^2 by using 9-bit fixed-point arithmetic, where x=(180,-115) in 2-D LNS with odd base 3. The actual value of x is $0.207231\ldots$. Clearly, $x^2 = (360, -230)$, which would cause overflow in 9-bit arithmetic. But if we multiply in advance by a (properly selected) good approximation of one, then the result obtained will have much smaller binary and ternary exponents; consequently, there will not be any risk of overflow. In our case, by multiplying x times $(-84, 53)$ we obtain $(96, -62)$ and now the squaring can be achieved in 9-bit arithmetic without overflow.

More to the point, if at any stage of the computational process one obtains a pair of large exponents, they can be reduced to within the required exponent dynamic range by multiplying the number obtained by a good approximation of one.

Such a feature does not exist in either the binary or logarithmic number system and may be used in fault-tolerant computing systems. One may try to find much better approximations by changing the number of bases. We report a curious numerical result with 8-dimensional LNS having an extremely good approximation of one with very small exponents:

Example 2: Consider an 8-D LNS with the set of bases (2,3,5,7,11,13,23,41).

The vector (13,-3,-3,-7,4,1,-1,1) provides an extremely good approximation of one. To wit:

$$1 < 2^{13}3^{-3}5^{-3}7^{-7}11^{4}13^{1}23^{-1}41^{1} < 1 + 10^{-10}$$

Such an example demonstrates that there are some multidimensional LNS that can be used to handle very complex fault-tolerant computational problems. The investigation of these possibilities is somewhat outside the scope of this paper.

4. HARDWARE COMPLEXITY

In order to provide complexity results for the MDLNS inner product computation unit, we expand on the inner product processor architecture initially developed for the 1-Digit 2-D LNS [12]. The processor can be used in a systolic array for 1-D convolution.

4.1 Single Digit Computational Unit

Figure 2 shows the structure of the proposed single-digit computation unit (CU). Since we do not require to retain the MDLNS representation of the accumulated output, and also since the CU is feedforward, we can use the MDLNS domain for the coefficient multiplication and a binary representation for the accumulated output.

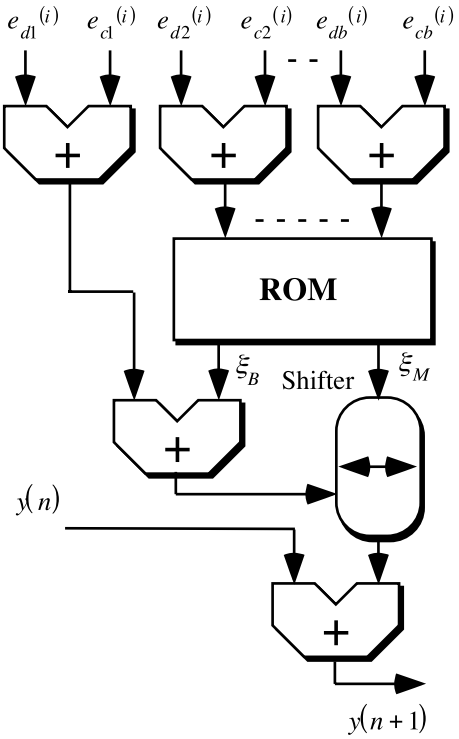

Figure 2. 1-digit MDLNS inner product computation unit

The multiplication is performed by parallel small adders for each of the data and coefficient base exponents. The addition output for each of the $b-1$ odd bases is concatenated into an address for a lookup table (ROM). This table produces an equivalent floating-point value for the product of the odd bases raised to the exponent sum, as shown below:

$$\prod_{j=2}^{b} p_j^{(e_{dj}^{(i)} + e_{cj}^{(i)})} \approx 2^{\xi_B} \cdot \xi_M \tag{8}$$

We note that since the size of the exponents of each odd base in an MDLNS representation (where there are at least 2-digits and 2 bases) can be very small (<4 bits), then the maximum address input to the ROM is given by $4 \cdot (b-1)$ bits. This is an 8-bit address table for a 3-D LNS. The shifter in the floating point to binary (2 s complement) conversion part of the unit also handles the sign of the product (not shown in Figure 2 for brevity).

For large dimensional LNS, we can also consider the use of unity approximants to reduce the output of each odd-base adder to the number of bits of the input exponents (or even less if we are willing to accept the increased mapping error). This reduction process stores a small number of unity approximants that can be added in parallel to the output of the odd-base adders. The reduced input to the ROM is selected from these parallel results. The ROM input address size is now reduced by $(b-1)$ bits. This is a subject of current research work and will not be explored further here. For the complexity analysis in Section 4.3, we will assume the structure of Figure 2.

4.2 n-digit Computational Unit

The n-digit computational unit is a simple parallel extension of the 1-digit unit. Each of the units computes the binary output for one of the digit combinations. As an example, consider multiplying an accumulating sequence, y, with a coefficient, x, $z = x \cdot y$, where:

$$y = \sum_{i=1}^{2} s_i^{[y]} \prod_{j=1}^{2} p_j^{e_j^{[y](i)}} \; ; \; x = \sum_{i=1}^{2} s_i^{[x]} \prod_{j=1}^{2} p_j^{e_j^{[x](i)}} .$$

We can perform this with 4 parallel 1-digit units, where the (u,v) unit computes:

$$z_{u,v} = s_u^{[y]} \cdot s_v^{[x]} \prod_{j=1}^{2} p_j^{(e_j^{[y](u)} + e_j^{[x](v)})} \tag{9}$$

Clearly there are n^2 such units in an n-digit MDLNS.

The parallel outputs are summed at the end of the systolic arrays using an adder tree.

4.3 MDLNS Complexity Comparison

In terms of complexity comparison with the classical LNS, we need to compare each system based on the same quantization properties.

This will require a knowledge of the change in the size of the odd-base exponents as a function of dimensionality and number of digits. At the moment this appears to be a rather intractable task and so we have to resort to exhaustive search analysis. Here we report the results from 10 and 12-bit unsigned binary dynamic range (these are taken from typical video filter requirements). From Section 3.2, we obtain the number of bits for the classical LNS as $n + \lceil log_2 n \rceil$. These results are shown in Table 3. Here we have assumed that the LNS hardware uses the same ROM/shifter architecture as the 1-Digit 2D LNS [20]. Our assumption in Table 3 is that the ROM size dominates the hardware complexity. For the classical LNS and 1-digit 2D LNS, this is a valid assumption. For the 2-digit LNS the ROM size is very small and the other components in the architecture will undoubtedly be important in the overall complexity.

Table 3: Classical LNS and 2D LNS Comparison

Binary Dynamic Range	LNS	1-digit 2D LNS	2-digit 2D LNS
10-bits	1K	1K (10-bit odd-base exponent)	64 words (4-bit ternary exponent)
12-bits	4K	4K (12-bit odd-base exponent)	128 words (5-bit ternary exponent)

Even so, it is clear that there is a very substantial reduction in the hardware complexity for the 2-digit case. We are currently performing a comparison analysis for a much larger set of digits and dimensions. Our initial findings point to the number of digits as being the primary contributor to the hardware savings.

5. FILTER DESIGN

As an example of the applicability of the MDLNS representation to DSP applications, we have designed a variety of digital filters using the architecture proposed in Section 4.1. Our designs have used both 1 and 2-digit 2D LNS representations with different odd bases.

The design technique is remarkably simple, even though on the surface it appears a very complex integer programming problem. Figure 3 shows the envelope of the distribution of filter coefficients from a study of 200 different designs using 9-bit quantization on the coefficients [21].

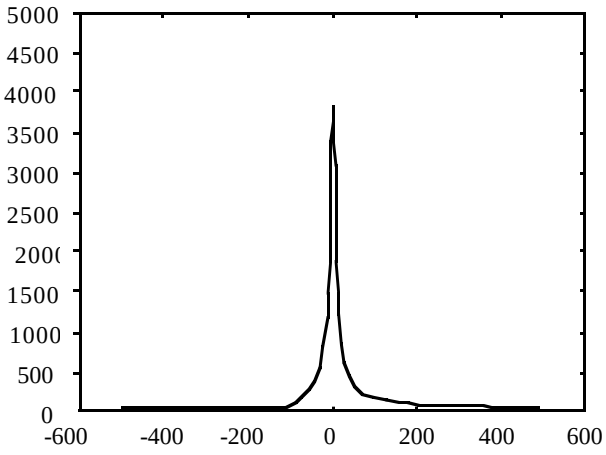

Figure 3. Filter Coefficient Distribution

Such a coefficient distribution is better represented by a logarithmic-like number system, such as the MDLNS, rather than a classical positional number representation (such as binary).

Taking advantage of this distribution match we have developed a design procedure for a 2-digit MDLNS using a standard greedy algorithm [10][12].

Results for a 53-tap low-pass filter design are shown in Figure 4. We have included 3 plots: the original double-precision Remez exchange design; the best case quantized result (2-digit with odd-base 3); the worst-case result (2-digit with odd-base 47). Table 4 shows the stop-band attenuation results for all of the filter designs.

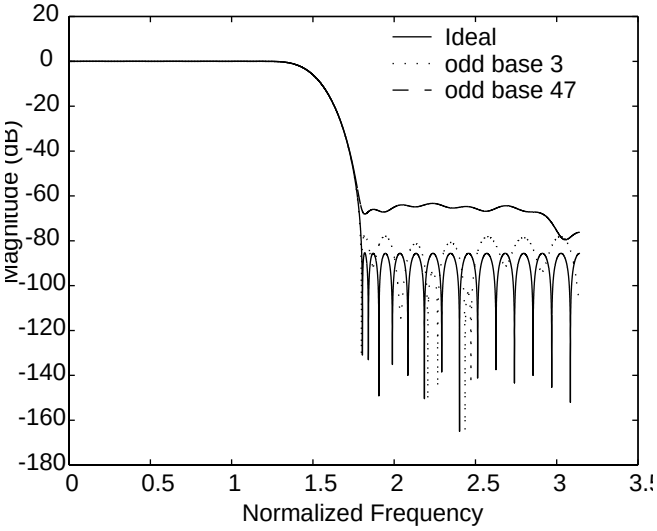

Figure 4. Filter design results for a 2-digit MDLNS

Table 4: Stop-band attenuation results

odd-base	Ideal	3	5	7	11	47
stop-band attenuation (dB)	85.5	78	70	64.5	71.3	63.3

A 2-digit 2D LNS architecture for a 15-tap filter has been fabricated and successfully tested. The filter data uses the full 2-digit representation, but the filter coefficients were designed using a 1-digit MDLNS (a hybrid representation). This requires 2 inner product computational units per coefficient. The layout is shown on the left of Figure 5 and the two parallel arrays are clearly visible. A full 2-digit MDLNS design of a 53-tap filter (not yet fabricated) is shown on the right hand side. There is a factor of 8 in the complexity reduction, which clearly shows the benefit of using a full multiple digit MDLNS.

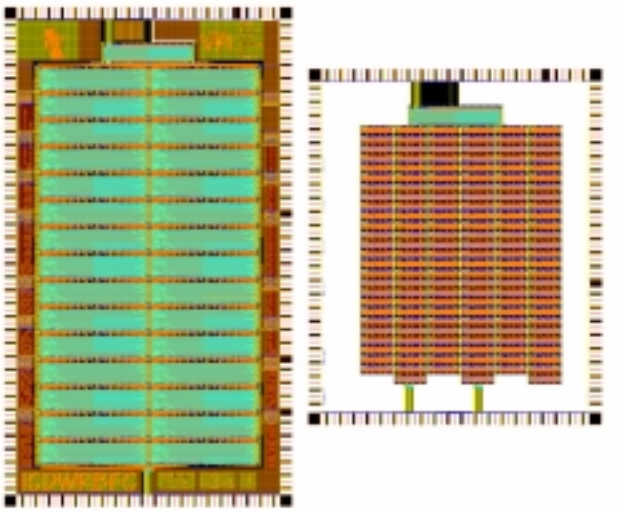

Figure 5. Hybrid and 2-D filter layout comparison

6. CONCLUSIONS

In this paper we have demonstrated that a multi-digit multi-dimensional logarithmic number system has considerable implementation advantages over the classical LNS for many DSP applications. This paper has generalized previous studies of a double-base number system, and we have illustrated some of the advantages of the extra degrees of freedom in both numbers of digits and the dimensionality of the representation. We have demonstrated, using FIR filter VLSI layouts, the complexity reduction obtainable by using a multiple digit MDLNS representation. A more formal comparison between the MDLNS and the classical LNS is currently under investigation in our laboratory.

7. REFERENCES

[1] I.Koren, Computer Arithmetic algorithms, Englewood Cliffs, NJ: Prentice Hall, 1993

[2] E.E. Swartzlander and A.G. Alexopoulos, The Sign/ Logarithm Number System, *IEEE Trans. Computers*, vol. 42, pp 1238-1242, 1975

[3] F.J. Taylor, R. Gill, J. Joseph and J. Radke, A 20 Bit Logarithmic Number System Processor, *IEEE Trans. Computers*, vol. 37, pp 190-200, 1988

[4] M.G. Arnold, T.A. Bailey, J.R. Cowles and J.J. Cupal, Redundant Logarithmic Arithmetic, *IEEE Trans Computers*, vol. 39, No 8, pp 1077-1086, 1990

[5] J.-M. Muller, A. Scherbyna and A. Tisserand, Semi-Logarithmic Number Systems, *IEEE Trans. Computers*, vol. 47, No 2, pp 145-151, 1998

[6] N.G. Kingsbury and P.J.W Rayner, Digital Filtering Using Logarithmic Arithmetic, *Electronics Letters*, vol 7, pp 56-58, 1971

[7] D.M. Lewis, 114 MFLOPS Logarithmic Number System Arithmetic Unit for DSP Applications, *IEEE J. Solid-State Circuits*, vol. 30, pp 1547-1553, 1995

[8] D.M.Lewis, Interleaved memory function interpolators with application to an accurate LNS arithmetic unit, *IEEE Trans. Computers*, Vol. 43, No. 8, pp.974-982, 1994.

[9] J.N. Coleman, E.I. Chester, C.I. Softley and J. Kaldec, Arithmetic on the European Logarithmic Microprocessor, *IEEE Trans. Computers*, vol 49, No 7, pp 702-715, 2000

[10] J. Eskritt, R. Muscedere, G.A.Jullien, V.S.Dimitrov and W.C.Miller, A 2-digit DBNS filter architecture, *IEEE Workshop on Signal Processing*, Louisiana, Oct. 2000

[11] V.S.Dimitrov, S.Sadeghi-Emamchaie, G.A.Jullien and W.C.Miller, A near canonic double-base number system with applications in DSP, *SPIE Conference on Signal Processing Algorithms*, vol. 2846, pp.14-25. 1996

[12] V.S.Dimitrov, G.A.Jullien and W.C.Miller, Theory and applications of the double-base number system, *IEEE Trans. on Computers*, vol. 48, No. 10, pp. 1098-1106, Oct. 1999

[13] S.Sadeghi-Emamchaie, G.A.Jullien, V.S.Dimitrov and W.C.Miller, Digital arithmetic using cellular neural networks, *Journal of Circuits, Systems and Computers*, No. 8, vol. 6, pp. 515-535, Dec. 1998

[14] G. A. Jullien, V. S. Dimitrov, B. Li, W. C. Miller, A. Lee, and M. Ahmadi, 1999, A Hybrid DBNS Processor for DSP Computation, *Proc. Int. IEEE Symp. Circuits and Systems*, Orlando, FL.

[15] V.S.Dimitrov, G.A.Jullien and W.C.Miller, An algorithm for modular exponentiation, *Information Processing Letters*, vol. 36, No. 5, pp. 155-159, May 1998

[16] R. Muscedere, G.A. Jullien, V. Dimitrov, W.C. Miller, 2000, Non-linear signal processing using index calculus DBNS arithmetic , *Proc. of the 2000 SPIE conf. on Advanced Algorithms and Architectures in Signal Processing*, San Diego, August 2000

[17] R.Muscedere, G.A.Jullien, V.S.Dimitrov and W.C.Miller, On efficient techniques for difficult operations in one and two-digit DBNS index calculus, *Asilomar Conf. on Sig., Syst. and Comp.*, Oct. 2000

[18] B.M.M deWeger, Algorithms for Diophantine equations, CWI Tracts, Amsterdam, vol. 65, 1989

[19] Computational methods in number theory, eds. H.W.Lenstra and R.Tijdeman, Mathematical Centre Tracts, vol. 155, 1987

[20] D. Lewis, An Architecture for Addition and Subtraction of Long Word Length Numbers in the Logarithmic Number System, *IEEE Trans. Computers*, Vol. 39, No. 11, pp. 1325-1336, 1990

[21] M. Shahkarami, Exploiting Redundancy in Modulus Replication Inner Product Processors, Ph.D. Thesis, University of Windsor, 1999.