



VLSI Research Group

University of Windsor

*Theory and Applications of the
Double-Base Number System*

Re-submitted to IEEE Transactions on Computers

V. S. Dimitrov, G. A. Jullien and W. C. Miller

Theory and Applications of the Double-Base Number System

V. S. Dimitrov, G. A. Jullien and W. C. Miller

VLSI Research Group, University of Windsor
Windsor, Ontario, Canada N9B 3P4

Abstract

In this paper we analyze some of the main properties of a double base number system, using bases 2 and 3; in particular we emphasize the sparseness of the representation. A simple geometric interpretation allows an efficient implementation of the basic arithmetic operations and we introduce an index calculus for logarithmic-like arithmetic with considerable hardware reductions in look-up table size. We discuss the application of this new number system in the area of digital signal processing; we illustrate the discussion with examples of finite impulse response filtering.

Keywords: *Double-Base number system; Index Calculus; Digital Signal Processing*

1 Introduction

In many applications the computational complexity of algorithms crucially depends upon the number of zeros of the input data in the corresponding number system [1]-[5]. Number systems are often chosen to enable a reduction of the complexity of the arithmetic operations; the most popular are, perhaps, signed-digit number systems [5]. An analysis of the expected number of zeros in the representation of arbitrary integers in the binary signed-digit number system shows that on average, for long wordlengths, 33% fewer adders are needed to perform multiplication than binary [6][7]. In these number systems we need, on average, $O(\log N)$ non-zero digits to represent the integer N .

A number system, allowing as digits only 0,1 and requiring $o(\log N)$ nonzero digits, is the double base number system (DBNS), using bases 2 and 3; that is, a representation having the form of eqn. (1).

$$x = \sum_{i,j} d_{i,j} 2^i 3^j \quad (1)$$

Clearly the binary number system is a special case (and valid member) of the above representation. In this paper we will deal with canonic (minimal number of non-zero digits) and near canonic forms for the representation.

The DBNS has an unusually simple 2-D geometric interpretation, suitable for implementation via cellular automata [8] for example, and, in this paper, we introduce an index calculus with which we can perform arithmetic using logarithmic-like computational units.

Arithmetic operations in this number system do not guarantee that the results are obtained in the minimal, or canonic form, and the associated problem of conversion from such a non-canonic form leads to interesting problems in transcendental number theory. The canonic number system, however, appears to provide very fast carry-free addition and is also suitable for multiplication. We illustrate our ideas with applications to computing finite impulse response filter inner products.

2 Properties of the DBNS

Definition 1: The representation of a given integer x into the form:

$$x = \sum_{i,j} d_{i,j} 2^i 3^j, d_{i,j} \in \{0, 1\} \quad (2)$$

will be referred to as a *double-base number system* (DBNS).

The representation of a given integer as a sum of minimal number 2-integers (numbers of the form $2^i 3^j$, smaller than or equal to x) will be referred to as the *canonic double-base number representation* (CDBNR). The main feature of the CDBNR is the unusual sparsity of the representation. It is easy to check, for example, that 23 is the smallest integer requiring three 2-integers. The smallest integer, requiring four 2-integers is 431, five 2-integers are needed to represent 18431 and six 2-integers are needed to represent 3 448 733. Up to this limit we can represent every integer as a sum of at most five 2-integers. The procedure to find a CDBNR of a given very large integer seems to be a very complex task; however, in [9] we have proposed a greedy algorithm with the input as a positive integer x ; and an output of 2-integers, a_i , such that $\sum_i a_i = x$. The algorithm finds the largest 2-integer, w , smaller than or equal to x , and recursively applies the same for $x - w$ until reaching zero. In [9] we prove the following theorem:

Theorem 1: The greedy algorithm terminates after $k = O\left(\frac{\log x}{\log \log x}\right)$ steps.

We also conjecture that the coefficient associated with the complexity order is close to unity (this is based on performing many numerical experiments).

Definition 2: We call the representation obtained by the greedy algorithm a Near-Canonic DBNR (*NCDBNR*).

To confirm the utility of the greedy algorithm, we generated NCDBNRs for 1000 randomly chosen 215-bit integers. We predict (from Theorem 1) that the expected number of non-zero digits is 27.75; the occurrence of the number of 2-integers peaks at about 30, as shown in Figure 1, which successfully demonstrates the efficacy of our algorithm. A variety of computational experiments shows that the largest 2-integer, smaller than x , occurs in at least one of the CDBNRs of x , in about 80% of the cases. This observation along with Theorem 1 allows an

estimate that the greedy algorithm returns a CDBNR with probability $0.8^{\frac{\log x}{\log \log x}}$; fortunately this tends to zero very slowly.

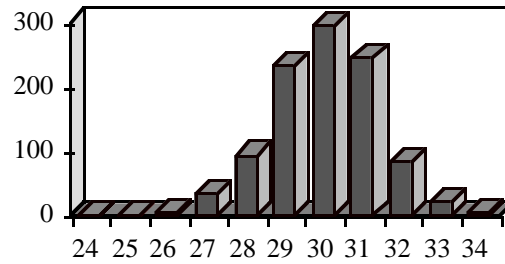


Figure 1. Occurrence of 2-integers for 215-bit numbers

We also find that the greedy algorithm produces a representation requiring, on average, $O\left(\frac{\log x}{\log \log x}\right)$ 2-integers. This NCDBNR provides a sufficiently sparse representation of x to make it very useful.

3 Arithmetic Ready Transformations

The mechanism of finding the NCDBNR plays a crucial role in performing basic arithmetic operations. Along with sparseness, we also require that non-zero digits be non-consecutive in our mapping representation; this allows addition to be mapped to a simple boolean operation. Based on this requirement, we provide the following definition:

Definition 3: A DBNR that has no consecutive non-zero digits is defined as an addition ready DBNR (ARDBNR).

3.1 ARDBNR reduction rules

We can use a geometrical interpretation, with orthogonal dimensions for each of the bases, to represent numbers in the DBNR. Non-zero DBNR digits are shown as black squares (active cells). This interpretation allows us to demonstrate simple identities on special combinations of active cells that provide a transformation of a DBNR to an ARDBNR.

For example, Table 1 shows the representation of the identity $2^i 3^j + 2^{i+1} 3^j = 2^i 3^{j+1}$ to remove consecutive active cells lying in one column.

Table 1 Column reduction

	...	3^j	3^{j+1}
...			
2^i			
2^{i+1}			
...			

 \Rightarrow

	...	3^j	3^{j+1}
...			
2^i			
2^{i+1}			
...			

Table 2 demonstrates the application of the identity, $2^i 3^j + 2^{i+1} 3^{j+1} = 2^{i+2} 3^j$, to remove consecutive active cells lying in one row. This procedure is akin to the symbolic substitution process used in optical computing.

Table 2 Row reduction

	...	3^j	3^{j+1}
...			
2^i			
2^{i+1}			
2^{i+2}			
...			

 \Rightarrow

	...	3^j	3^{j+1}
...			
2^i			
2^{i+1}			
2^{i+2}			
...			

3.2 Generalized reduction

We can generalize the reduction problem using the purely exponential Diophantine eqn. (3), where $l < k$.

$$\begin{aligned}
 & 2^{i_1} 3^{j_1} + 2^{i_2} 3^{j_2} + \dots + 2^{i_k} 3^{j_k} \\
 & = 2^{m_1} 3^{n_1} + 2^{m_2} 3^{n_2} + \dots + 2^{m_l} 3^{n_l}
 \end{aligned}
 \tag{3}$$

The problem of solving Diophantine equations such as eqn. (3) has been a subject of investigation over the last two decades [10], although some interesting results were obtained in the 30's and 40's [11].

We need only consider some special cases for k and l . For example using $k=2, l=1$, we can prove the following theorem:

Theorem 2: The Diophantine equation $x + y = z$ where $\text{GCD}(x, y, z) = 1$ and x, y and z are 6-integers (that is x, y, z have the form $2^{x_1}3^{x_2}5^{x_3}7^{x_4}11^{x_5}13^{x_6}$, with $x_i \geq 0$, $i = 1, 2, 3, 4, 5, 6$) has exactly 545 solutions.

Proof: See [12].

□

In our case $x_3 = x_4 = x_5 = x_6 = 0$ and the only solutions of $x + y = z$ are $\{1, 2, 3\}$, $\{1, 3, 4\}$ and $\{1, 8, 9\}$. Therefore these represent the only 3 cases where we can replace two active cells with one.

For $k=3, l=1$ an interesting possibility for reducing the active cells follows from the solution of the Pillai equation [13]:

$$2^a \pm 3^b = 2^c \pm 3^d \quad (4)$$

Pillai was able to solve all of the above 4 equations, excluding the equation $2^a - 2^b = 3^c - 3^d$ on which he conjectured that the only solutions are $(3,1,2,1)$, $(5,3,3,1)$ and $(8,4,5,1)$. The conjecture was proved by Stroeker and Tijdeman [22]. For our purposes only eqn. (5) and (6) are relevant:

$$2^a - 2^b = 3^c + 3^d \quad (5)$$

$$2^a + 2^b = 3^c - 3^d \quad (6)$$

Following [22], the solutions of (4) are $(2,1,0,0)$, $(3,1,1,1)$, $(5,1,1,3)$, $(3,2,1,0)$, $(5,2,0,3)$, $(4,2,1,2)$ and $(8,2,5,3)$, while (5) has only one solution $(0,0,1,0)$.

The total number of solutions of the equation $x + y + z = t$, $\text{GCD}(x, y, z, t) = 1$, in 2-integers is 27 and they are tabulated in Table 3.

A powerful technique in transcendental number theory (studies of equations such as (3)), described by Baker [14], allows one to conclude that eqn. (3) has only a finite number of solutions. Existing methods for bounding the upper limits, however, give very large upper bounds; therefore, in searching for a MDBNR, we are forced to use methods that do not guarantee exact *minimality* but rather provide both near minimality, and algorithmic realization.

Table 3. The Solutions Of the Equation $x + y + z = t$ in 2-integers

1,2,3,6	1,2,6,9	1,2,9,12	1,2,24,27	1,3,4,8	1,3,8,12
1,3,12,16	1,3,32,36	1,4,27,32	1,6,9,16	1,8,9,18	1,8,18,27
1,8,27,36	1,8,72,81	1,9,54,64	1,12,243,256	1,16,64,81	1,27,36,64
1,32,48,81	1,216,512,729	2,3,4,9	2,3,27,32	2,9,16,27	3,4,9,16
3,8,16,27	8,9,64,81				

4 DBNS-Map Addition and Multiplication

4.1 Addition

Let x and y be two integers in the CDBNR. We note that if x and y contain the element $2^i 3^j$, then the element $2^{i+1} 3^j$ does not exist, therefore addition can be computed by simply overlaying the corresponding DBNS maps; there will be no overlapping active cells. In order to prepare for another addition we ideally perform a reduction into minimal form. In practice, however, we only require an ARDBNR and the symbolic substitution methods described in the previous section can be effectively used.

Let $I_x(i,j)$ and $I_y(i,j)$ be the DBNS maps of the integers x and y , represented in the ARDBNR. The image $I_z(i,j)$ of the DBNS map of the number $z = x + y$ can be obtained using:

$$I_z(i+1,j) = I_x(i,j) \text{ AND } I_y(i,j) \quad \text{Rule (1)}$$

$$I_z(i,j) = I_x(i,j) \text{ XOR } I_y(i,j) \quad \text{Rule (2)}$$

Note, using the ARDBNR, if $I_x(i,j) = I_y(i,j) = 1$, then $I_x(i+1,j) = I_y(i+1,j) = 0$ and therefore addition can be accomplished using a symbolic substitution technique. To reduce this result it is sufficient to use the following rules (see Tables 1 and 2.)

$$I_z(i,j+1) = I_z(i,j) \text{ AND } I_z(i+1,j) \quad \text{Rule (3)}$$

$$I_z(i+2,j) = I_z(i,j) \text{ AND } I_z(i,j+1) \quad \text{Rule (4)}$$

In the worst case, we have to perform a reduction ('carry' removal) for each row in the 2-D representation. For an equivalent n -bit binary number, there are $\frac{n}{\lg_2 3}$ rows in the representa-

following from the solutions of the Pillai equation [13], eqn. (4). The final map is given in Table 5.

Table 5. Pillai Solution Final Map

	1	3	9	27	81
1					
2					
4					
8					
16					
32					
64					
128					

$$x+y=211=128+81+2$$

4.2 Multiplication:

Let x and y be integers, represented by DBNS maps in the CDBNR. The CDBNR of their product, z , is an n -tuple of the elements $\{2^{i_z}3^{j_z} = 2^{i_x+i_y}3^{j_x+j_y}\}$, where the $\{i_x, j_x\}$ and $\{i_y, j_y\}$ are the 2-integer index locations of the active cells in the CDBNRs of x and y respectively. It is clear that the multiplication process simply corresponds to 2D shifts and DBNS additions, in an equivalent way to that performed using binary arithmetic. The promise here, however, is that the number of operations is considerably reduced based on the sparseness of the representation.

Let us consider the multiplication of the numbers 79 and 107, represented via their DBNS maps. as shown in Table 6.

Table 6. Example of the DBNS Multiplication Process

	1	3	9	27
1				
2				
4				
8				
16				
32				
64				

×

	1	3	9	27
1				
2				
4				
8				
16				
32				
64				

$x = 79$

$x = 107$

The final reduced forms of the product can be found by using 2 specific solutions of the Pillai equation (eqn. (4)).

$$I_z(i,j+4) = I_z(i,j) \text{ AND } I_z(i+4,j) \text{ AND } I_z(i+6,j) \tag{Rule (7)}$$

$$I_z(i,j+3) = I_z(i+1,j) \text{ AND } I_z(i,j+2) \text{ AND } I_z(i+4,j) \tag{Rule (8)}$$

The representation of the multiplication result is shown at the left side of Table 7, and the ARDBNR reduction, using Rule (7) and Rule (8), at the right side of Table 7.

Table 7. The reduction rules applied here are Rule (7) and Rule (8)

	1	3	9	27	81	243	729
1			■				
2	■			■			
4		■					
8					■		
16			■				
32		■					
64			■				
128							
256				■			

=

	1	3	9	27	81	243	729
1							■
2	■				■		
4							
8					■		
16							
32							
64							
128							
256				■			

Efficient implementation of the NCDBNR (or ARDBNR) reduction procedure is an ongoing project within our group. An interesting implementation using analog Cellular Neural Networks has recently been presented [15] using only the column and row reduction rules of Table 1 and Table 2 to obtain ARDBNR forms.

5 DBNS Index Calculus

The n-tuple, $\{2^{i_z}3^{j_z} = 2^{i_x+i_y}3^{j_x+j_y}\}$, introduced in the previous section, immediately leads to an implementation of multiplication using index addition, where the index mapping of x , for example, is simply the n-tuple $\{i_x, j_x\}$. For cases where we are approximating the reals by fixed point numbers, it is possible to find a *single index* CDBNR for any real number with arbitrary precision. This leads us to a multiplication technique only involving a single 2D shift, which corresponds to a pair of index additions. If only one of the numbers to be multiplied is in the single index form, multiplication will only require $O\left(\frac{\log x}{\log \log x}\right)$ addition pairs.

The following theorem proves the single index mapping property.

Theorem 3: Let n and m be integers. The set $A_{n,m} = \{2^n 3^m\}$ is compact over the nonnegative reals; that is, in every interval $[\delta_1 \div \delta_2]$ ($\delta_1 \geq 0, \delta_2 > \delta_1$) at least one number of the form $2^n 3^m$ appears.

Proof: The proof follows from the well-known fact that the set of numbers: $\{n + m \cdot \omega\}$, $n, m \in \mathbb{Z}$, ω – irrational is compact over the reals. In our case we have $\omega = \log_2 3$.

□

Based on our geometrical interpretation, if we extend the DBNS map in both directions, then every nonnegative real number can be approximated with arbitrary small error using only one active cell. We can show that a single-cell coefficient can be represented by two $(\log_2 n)/2$ -bit numbers so we have not incurred any dynamic range redundancy compared to the binary representation of the number, even though our *DBNS* representation is inherently very redundant (and sparse). The advantage, of course, is the ability to use index calculus on the representation.

5.1 Implementing the index calculus

We represent a number, x , as a triple (s_x, b_x, t_x) , where s_x is the sign bit, and b_x and t_x are integers such that $s_x 2^{b_x} 3^{t_x}$ is an acceptable approximation to x . More precisely, if ε is the error allowed, then $|x - s_x 2^{b_x} 3^{t_x}| < \varepsilon$.

We have a low complexity implementation of multiplication and division, namely, if: $x = (s_x, b_x, t_x)$ and $y = (s_y, b_y, t_y)$, then:

$$x \cdot y = ((s_x + s_y) \bmod 2, b_x + b_y, t_x + t_y) \quad (7)$$

$$x/y = ((s_x + s_y) \bmod 2, b_x - b_y, t_x - t_y) \quad (8)$$

The implementation of addition and subtraction within this index calculus can be performed using the identities:

$$\begin{aligned} 2^a 3^b + 2^c 3^d &= 2^a 3^b (1 + 2^{c-a} 3^{d-b}) \\ &\approx 2^a 3^b \Phi(c-a, d-b) \end{aligned} \quad (9)$$

$$\begin{aligned}
2^a 3^b - 2^c 3^d &= 2^a 3^b (1 - 2^{c-a} 3^{d-b}) \\
&\approx 2^a 3^b \Psi(c-a, d-b)
\end{aligned} \tag{10}$$

We will, of course, precompute and store the functions containing the approximation of:

$$\Phi(x, y) = 1 + 2^x 3^y \approx 2^\alpha 3^\beta \tag{11}$$

$$\Psi(x, y) = 1 - 2^x 3^y \approx 2^\gamma 3^\delta \tag{12}$$

Addition (subtraction) of two numbers is now mapped into the following two steps:

1. Find the corresponding element (α, β) in the table;
2. Add (subtract) (a, b) with (α, β) .

5.2 Comparison with the Log Number System

It is clear that this index calculus shares some similarities with the Logarithmic Number System (LNS) [16]. Both allow the mapping of multiplication and division to addition and subtraction, uses an identity requiring the look-up of a unary function (e.g. eqn. (11) is similar to the LNS unary function $\log_2(1 + 2^{(\beta-\alpha)})$ where the input to the table is $(\beta - \alpha)$, the logarithms of the numbers being added).

There is a fundamental difference in the mapping, however, since the DBNS involves indices over 2 orthogonal dimensions, where the logarithmic number system involves only one ‘index’. In the DBNS, if the error of the computations is fixed to be ϵ , then in approximating some real number, x , we can expect the size of the integers used to be smaller than the corresponding fixed-point number in the LNS [17]. Therefore, the multiplications (divisions) can be performed by two parallel additions (subtractions) of two small numbers.

A disadvantage of the DBNS is that there is no direct way to quickly compare two numbers of the form $2^a 3^b$ and $2^c 3^d$. The simplest way seems to be to compare the numbers $a + b \cdot \log_2 3$ and $c + d \cdot \log_2 3$, which requires one fixed-number multiplication, one addition and one final subtraction to obtain the result.

To make a fair comparison between both number systems, we have to evaluate the size of the integers used in representing a given real number, x , and the size of the look-up tables used. To do this, we use Theorem 4:

Theorem 4: Given α, β reals and $\varepsilon > 0$, then there exist integers p and q , such that:

$$|q\alpha - p - \beta| < \varepsilon \text{ and } |p, q| \leq \sqrt[4]{8\varepsilon}^{-\frac{1}{2}}.$$

Proof: See reference [18].

□

There are many ways to easily find the corresponding p and q using one of the large variety of approximation algorithms. Examples include the continued fraction algorithm [19], LLL reduction [20], Ferguson-Forcade algorithm [21], Szekeres algorithm [22], etc. For our purposes the above theorem is relevant, because it allows us to estimate the size of the integers (b_x, t_x) taking part in the approximation of a real number, x . Let us assume, that x is represented with precision $\varepsilon = 2^{-k}$, then the above theorem tells us that there exist integers p and q requiring at most $(k/2) + 1$ bits and ensuring the approximation with desired accuracy. The theorem also guarantees that the size of the look-up tables will be the same as the size of the look-up tables in the LNS. Table 8 summarizes the comparison analysis.

Table 8. Comparison between the LNS and the DBNS

	Multiplication/ Division	Addition/ subtraction	Comparison
LNS	One n-bit addition (subtraction)	One n-bit addition (subtraction). One table.	one n-bit subtraction
DBNS (using index calculus)	Two n/2-bit parallel additions (subtractions)	Two n/2-bit additions (subtractions) One table.	one n/2-bit multiplication two n/2-bit subtracts.

6 A FIR Filter Example

A FIR filter implements the linear convolution:

$$y_i = \sum_k x_k \cdot h_{i-k} \quad (13)$$

A well established technique to reduce the complexity of eqn. (13) is to select filter coefficients with a small number of non-zero binary digits. Canonic-signed-digit (CSD) representations allow a greater choice of coefficients with no increase in the of number of non-zero digits. Such an approach, however, often leads to increases in the filter length to allow a desired spectral envelope to be matched. The DBNS representation typically allows single digit approximations with *much greater coefficient space support* than the CSD approach. There is also the advantage of reduced multiplication complexity, as discussed in the previous section.

We consider the following low-pass filter example to demonstrate the efficacy of the DBNS index mapping.

1. Passband and stopband edge frequencies of 0.021 and 0.07 radians.
2. Passband ripple and stopband attenuation requirements of 0.2dB and 60 dB respectively.

The infinite precision 60-tap filter has a passband ripple of 0.2 dB and stopband attenuation 61.7 dB; the coefficients, along with a 10-bit DBNS mapping, are given in Table 9.

Table 9. DBNS10-bit pairs for filter example

h_i	S_i	a_i	b_i	h_i	S_i	a_i	b_i
0.00378596	1	198	-130	-0.0566649	-1	782	-496
0.00341834	1	98	-67	-0.0410201	-1	100	-66
0.00354156	1	778	-496	-0.01574	-1	-827	518
0.00268679	1	841	-536	0.01928687	1	-58	33
0.00097656	1	-10	0	0.06421424	1	1012	-641
0.00207487	1	-337	207	0.11767056	1	-729	458
0.00720243	1	107	-72	0.17966271	1	-917	577
-0.0141662	-1	780	-496	0.2451056	1	-785	494
-0.0229532	-1	833	-529	0.31444408	1	-810	510
-0.0327162	-1	255	-164	0.38289611	1	211	-134
-0.0429886	-1	845	-536	0.44535403	1	972	-614
-0.0527344	-1	-9	3	0.5015795	1	-654	412
-0.061021	-1	-619	388	0.5468766	1	603	-381
-0.0649574	-1	23	-17	0.57795338	1	-42	26
-0.0645546	-1	-779	489	0.59443865	1	405	-256

The resulting stopband attenuation is 60.1 dB; still within specifications.

6.1 An index calculus IPSP

A major building block for DSP processors is the inner product step processor (IPSP). The basic structure of the index calculus IPSP is shown in Figure 2. As with LNS implementations, the sign of the data and coefficient inputs and the zero data case are handled separately; we have not included this logic block in order to keep the Figure 2 as simple as possible.

The function implemented is $S_i = S_{i-1} + (x_i \times y_i)$ where $x_i \Rightarrow \chi_i$ and $y_i \Rightarrow \zeta_i$. Each of the mappings produces a binary and ternary exponent. These exponents are added to produce the sum exponents. Rather than map back to the DBNS, we convert to a floating point-type representation for the accumulation. Noting that the binary exponents simply represent shifts, we only have to look up the exponent and mantissa for the ternary components. There are 4 binary adders, a $(b/2 + 1) \times b'$ -bit ROM and a barrel shifter. The word length of the ROM (b' -bits) is the concatenation of the mantissa and exponent for the approximation $3^{v_i^{[3]}} \cong y_i^{[e]} \times 2^{y_i^{[e]}}$. Note that the DBNS substitutes the LNS b-bit input ROM with the much smaller address size ROM and a barrel shifter. This may represent a considerable hardware reduction.

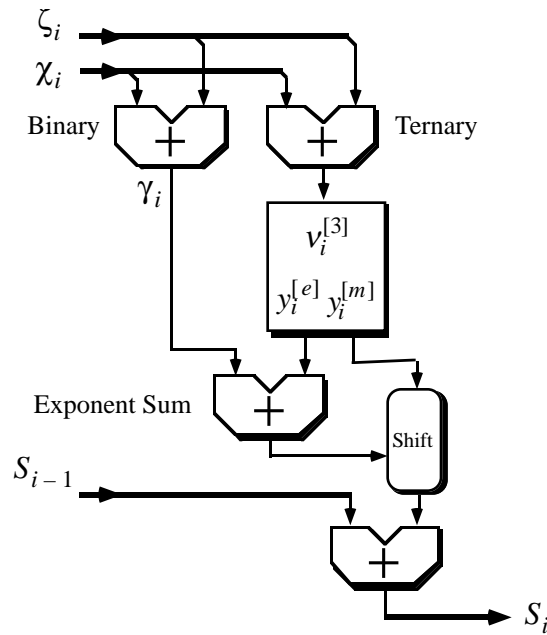


Figure 2. Index Calculus IPSP

The input conversion can be performed with a single table, providing the input data word-width is not too large. This is the same restriction as for LNS implementations, and still yields many practical DSP applications.

A very interesting feature of the IPSP in Figure 2, is that the mantissa is limited in the amount of right or left shift it will undergo in the barrel shifter. This is apriori knowledge since it depends on the precision of the original mapping. Clearly the inputs to the *Exponent Sum* adder, in general, will exceed the shift limit, but the sum output will always be within this limit. We can therefore save on adder hardware by only constructing modulo 2^S adders for both the Binary and Exponent Sum adders, where S is the maximum left or right shift. The exponent part of the ROM output can also be limited to an S -bit (modulo 2^S) word.

6.2 A DBNS FIR filter simulator

We have built a simulation of the IPSP in order to verify its operation, particularly in regard to the word reduction of the ROM and *Binary* and *Exponent Sum* adders. The simulation tests the IPSP in a multi-tap systolic FIR filter configuration. A screen dump of the essential elements of the simulator for a 5-tap filter test is shown in Figure 3.

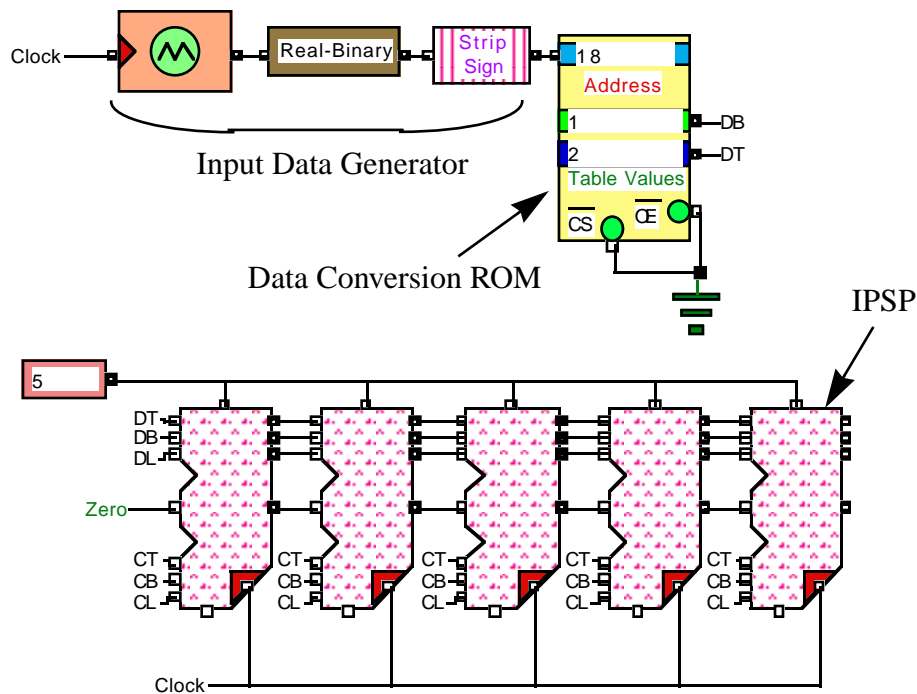


Figure 3. Screen dump of the DBNS 5-tap filter simulation

The *Data Conversion ROM* tables were generated from a custom integer programming package that selects binary and ternary exponents, a and b , based on minimizing the error $\varepsilon = |x - 2^a 3^b|$. The first 10 entries of this table for integer values of x , with 10-bits of precision, is shown in Table 10. These are the full values, not the modulo 2^5 reduced values. Note that only one data conversion ROM is required for the entire IPSP filter chain.

Table 10. First 10 entries of the Data Conversion ROM

x	a (DB)	b (DT)
1	0	0
2	1	0
3	0	1
4	2	0
5	-69	45
6	1	1
7	109	-67
8	3	0
9	0	2
10	-68	45

The top input to each IPSP is the number of bits used in the *Binary* and *Exponent Sum* adders of Figure 2. The inputs on the left are the index and sign data for the input data {DT,DB,DL} and the coefficient {CT,CB,CL} together with the binary accumulating sum. The leftmost IPSP receives a zero input for this sum. In this example all of the coefficients are set to the same value. Latches in the IPSP allow the data to ‘slide’ over the coefficient in this systolic implementation.

A screen dump of the essential elements of the IPSP is shown in Figure 4. For simplicity the entire data and coefficient words are input to the sign/zero logic detection blocks and the corrected data is output to the final adder from the *Fix* block. The *Binary* and *Exponent Sum* adders are set to 5-bits which provides a maximum shift of ± 16 places in the barrel shifter; this exceeds the actual requirement of ± 10 places. In this particular snap-shot, the coefficient is 8 (CB=3, CT=0) and the data is 23 (with an optimal 10-bit mapping of DB=144, DT=-88). The result of the multiplication operation is $8 \times 23 = 184$ which is correctly produced at the input to the *Binary Accumulator*. The ROM looks up the pre-stored value $3^{-88} \cong 2974 \times 2^{-151}$ and the mantissa is loaded into the barrel shifter. The binary exponent output of the ROM is only required to 5-bits

$$-151 \pmod{32} = 9$$

as well as the DB input to the binary exponent adder.

$$144 \pmod{32} = 16$$

The shift input to the barrel shifter is -4 and this value is correctly obtained providing that it is contained within the number of bits of the reduced binary adders.

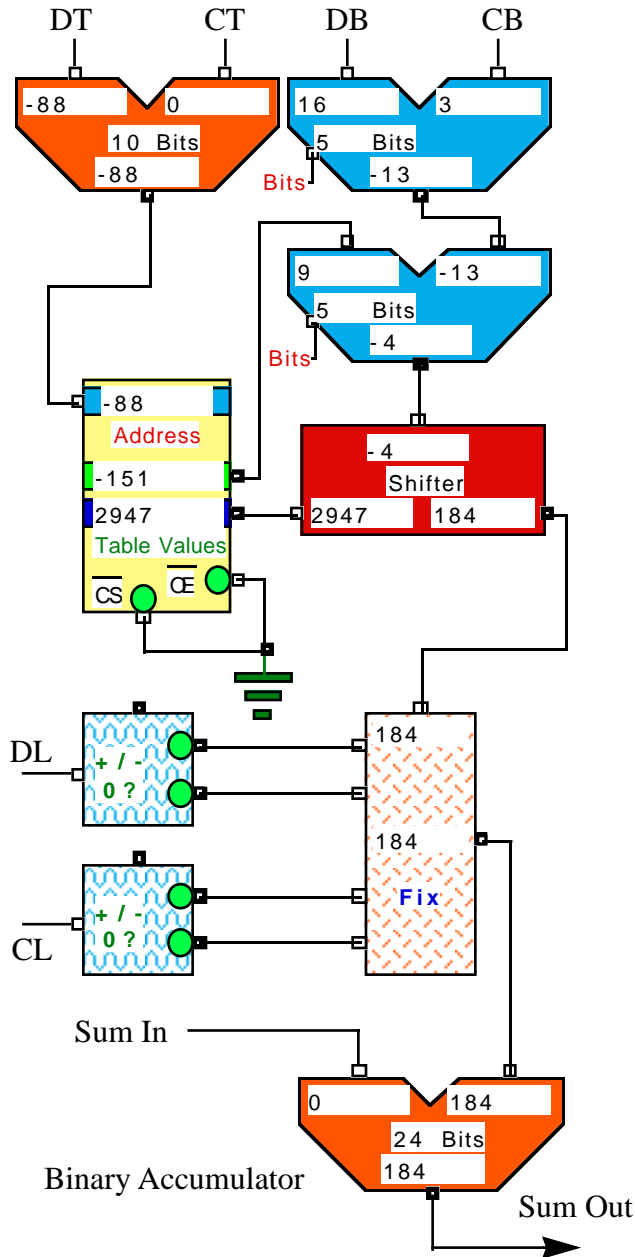


Figure 4. Screen dump of the IPSP Simulator

7 Conclusions

In this paper we have presented the theory of a recently introduced double-base number system as a technique for representing numbers that allows potentially low complexity arithmetic operations using a variety of implementation media.

We have explored an implementation procedure for arithmetic operations using symbolic substitution. For addition in the *DBNS* we simply have to ensure that there no consecutive active cells lying in any column on the *DBNS*-map. For multiplication, using 2-D shifts and additions, however, we seek representations with minimal active cells located large 2-D Hamming distances from each other. Symbolic substitution provides the ability to seek such representation, and we have presented a set of rules for the substitutions that produce near canonic forms for the output representation.

We have also introduced an index calculus which provides the same structure as the Logarithmic Number System (LNS) with a considerable reduction in hardware. An implementation advantage over the LNS is that the index additions and subtractions are reduced in complexity, because the binary and ternary operations are completely independent. The use of this index calculus has been demonstrated using an example of systolic FIR filtering.

8 References

- [1] A. Borodin and P. Towari, "On the decidability of sparse univariate polynomial interpolation", *Computational Complexity*, vol.1, 1991, pp.67-90
- [2] P. Montgomery, "A survey of modern integer factorization algorithms", *CWI Quarterly*, 7, 4, 1994, pp. 337-366
- [3] M.D. Ercegovac, T. Lang, J.G. Nash, and L.P. Chow, "An area-time efficient binary divider", *IEEE Int.Conf on Comp. Design*, Rye Brook, NY, Oct.1987, pp.645-648
- [4] P. Kornerup, "Computer arithmetic: exploiting redundancy in number representations", *ASAP95*, Strasbourg.
- [5] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic", *IRE Trans. Electronic Computers*, vol.10, 1961, pp. 389-400
- [6] H. Garner, "Number systems and arithmetic", *Adv. in Comp.*, vol.6, 1965, pp. 131-194
- [7] G.W. Reitwiesner, "Binary Arithmetic", *Adv. in Comp.*, vol.1, 1960, pp.231-308
- [8] E. Swartzlander, "Digital optical computing", *Applied Optics*, vol.25, 1986, pp. 3021-3032
- [9] V.S. Dimitrov, G.A. Jullien, W.C. Miller, "An algorithm for modular exponentiation",

- Information Processing Letters (66)3 (1998) pp. 155-159.
- [10] T.N. Shorey and R. Tijdeman, "Exponential Diophantine equations", 1986, Cambridge University Press
 - [11] G. Hardy, "Ramanujan", 1940, Cambridge Univ. Press
 - [12] B.M.M. de-Weger, "Algorithms for Diophantine equations", CWI Tracts-Amsterdam, vol.65, 1989
 - [13] S.S. Pillai, "On the equation $2^a - 2^b = 3^c - 3^d$ ", Bulletin of the Calcutta Mathematical Society, vol.37, 1945, pp. 15-20
 - [14] A. Baker, "The theory of linear forms in logarithms", in Transcendental theory-Advances and Applications, A.Baker (ed.), Academic Press, pp.1-27, 1987
 - [15] Saeid Sadeghi-Emamchaie, G.A. Jullien, V. Dimitrov and W.C. Miller, "Digital Arithmetic Using Analog Arrays", Proc. 8th Great Lakes Symposium on VLSI, Lafayette, Louisiana, February 1998 pp. 202-207.
 - [16] E.E. Swartzlander, D.V. Chandra, H.T. Nagel, and S.A. Starks, "Sign/logarithmic for FFT implementation", IEEE Trans. on Computers, vol.C-32, pp. 526-534, 1983
 - [17] D. Lewis, "An accurate LNS arithmetic unit using interleaved memory function interpolator", Proc. of ARITH-11, Windsor, 1993 pp. 2-9.
 - [18] V.S. Dimitrov and T.V. Cooklev, "Two algorithms for modular exponentiation using nonstandard arithmetic", IEICE Trans. on Fundamentals, 1995, pp. 82-87
 - [19] A.J. Brentjes, Multi-dimensional continued fraction algorithms, Mathematical Centre Tracts, Amsterdam, vol.145, 1981
 - [20] C.-Y. Chen, C.-C. Chang and W.-P. Yang, "Hybrid method for modular exponentiation with precomputations", IEE Electronics Letters, vol.32, 6, 1996, pp. 540-541
 - [21] H.R.P. Ferguson and R.W. Forcade, Generalization of the Euclidean algorithm for real numbers for all dimensions higher than two, Bulletin of the American Mathematical Society, vol.1, No. 1979, pp. 912-914
 - [22] R.J. Stroeker and R. Tijdeman, "Diophantine equations", in Computational methods in number theory, (ed.H.Lenstra and R.Tijdeman), Math.Centre Tracts-Amsterdam, vol.155, pp. 321-369, 1987

9 Acknowledgments

The authors would like to acknowledge financial assistance from the Natural Sciences and Engineering Research Council of Canada, the Micronet Network of Centres of Excellence and workstations and software from the Canadian Microelectronics Corporation. We also acknowledge the many valuable comments from the anonymous reviewers.

Comments to Referees

The authors would like to thank the referees for their detailed comments about this manuscript. Based on these comments, the following changes have been made:

Referee A

1. We have not found a technique to obtain a minimal representation and it would appear to be a very difficult number theoretic problem.
2. The comment about using symbolic substitution rules to find the NCDBNR (defined as obtained from the greedy algorithm) is extremely interesting. In fact, it is not clear that there is a symbolic substitution algorithm that guarantees producing the same near canonic representation as the greedy algorithm. In the revised manuscript we have identified an Addition Ready DBNR (ARDBNR) which is found by applying the first 2 substitution rules until no consecutive non-zero digits are obtained.
3. We have changed MDBNR to CDBNR (we had used an earlier definition by mistake).
4. We have provided an explanation for the statement that there are 37% fewer carries in DBNS addition than binary addition.
5. We have corrected the error in line 17 (it is actually $I_z(i,j+1) = I_z(i,j) \text{ AND } I_z(i+1,j)$). We have also changed the ternary index in Table 1 and Table 2 from i to j to agree with the variables used for indices in the equations.

Referee B

1. (i) The data conversion ROM, even though potentially larger than the ternary to binary conversion ROMs in the IPSP, is only required at the front end of the IPSP chain. For a large filter (e.g. a 54-tap low-pass interpolation filter for multi-rate systems) the data conversion ROM is probably no more than 4% of the total ROM area for the filter. We have added a statement to indicate that only one such ROM is required. We have tempered our previous statement that there will be a considerable hardware reduction to the statement “This may represent a considerable hardware reduction”.
- (ii) This statement is true. We have added a sentence to reflect this observation.
- (iii) We are currently testing a 0.35μ CMOS chip that contains all of the elements required in the IPSP. Using dynamic circuitry we find a 10-bit address X 17-bit word (12+5)ROM requires 840μ X 260μ in area and consumes 7.8mW/100MHz. A 12-bit to 12-bit barrel shifter with ± 10 position shift requires 126μ X 140μ and consumes less than 0.6mW/100MHz. This total combination is no more in area and much less in power than a ROM with an extra address-bit and a 12-bit output. If we save 2-bits off the address in comparison with a LNS IPSP, then we have saved 50% of the area. We have not included such comparisons in the revised manuscript because they are technology, circuit style, and

application dependent.

(iv) As we mentioned in (i) the Data Conversion ROM is only required at the beginning of the IPSP chain and is almost negligible for a filter with a large number of taps. The ternary lookup ROM actually contributes most of the area to each IPSP and at least 50% of the power consumption. Therefore, for a large IPSP chain, the size of the ROM in the IPSP IS a dominant consumer of area and power. This leads us to our conclusion that a DBNS index calculus approach may offer savings over a LNS approach when considering savings in ROM size.

(v) We are building integer processors therefore it would be entirely unfair to compare our approach with a floating point MAC architecture (we would win hands down!) In terms of comparisons with classical integer binary structures for MACs, all of our experiments show at least a 50% reduction in power and about 30% in area over, say, a PTL 12X12X16 MAC using a parallel multiplier architecture with Booth recoding and carry skip bottom adder. This is a comparison in the same technology with fairly aggressive binary arithmetic design practices. Architectures, such as LNS, DBNS, Distributed Arithmetic etc. are not candidates for the ALU of a general purpose DSP processor. Our target is special DSP ASICs (e.g. special video interpolation filters, MPEG DCT processors etc.) where maximizing performance (speed/power/area) is of utmost importance, and ease of design, for example using code written in 'C', is not an issue. Certainly DSP processors that include special purpose sub-blocks such as MPEG decoders, may be candidates for special arithmetic structures.

2. Some of the theoretical analysis concerning the use of the DBNS in modular exponentiation has been published in [9]. We have referred to this paper in the revised manuscript.

We have included some details of the simulator in order to provide a demonstration of the operation of the index calculus DBNS MAC.

3. We have added a definition of the geometric representation at the beginning of Section 3 and have provided an expansion of 'CSD' in Section 6.