Theory and Applications for a Double-Base Number System

V.S. Dimitrov, G.A. Jullien, W.C. Miller
VLSI Research Group, University of Windsor, Windsor, ON, Canada N9B 3P4

Abstract

In this paper we present a rigorous theoretical analysis 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. Two potential areas of applications are discussed: applications in digital signal processing for computation of inner products and in cryptography for computation of modular exponentiations.

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, $O(\log N)$ nonzero 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).

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

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 [9] 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 and modular exponentiations.

2. Basic definitions

Following from de-Weger [10], we use the following definitions:

Definition 1: An integer x, is called s-integer if all of its prime divisors are among the first s primes.

Definition 2: Let $G_{2,3}(x)$ be the set of 2-integers, smaller than or equal to x.

The asymptotic behavior of the cardinality of $G_{2,3}(x)$ can be easily estimated from the inequality $2^k 3^m \le x$, that is $k \cdot \ln 2 + m \cdot \ln 3 \le \ln x$. Hence the cardinality of $G_{2,3}(x)$ is equal to the number of nonnegative integer coordinates, satisfying eqn. (1). This number is approximately $\left\lceil \frac{(\log_2 x)^2}{2 \cdot \log_2 3} \right\rceil \approx \frac{1}{3.17}\log_2^2 x$ [11]. Obviously every integer, x, has different representations as a sum of 2-integers. Let us consider a table with $[\log_3 x] + 1$ columns and $[\log_2 x] + 1$ rows, so that in every cell (i,j) the number $2^i 3^j$ is written. An arbitrary integer smaller than or equal to $2^m 3^k$ can be represented as a sum of numbers, which appears in the first $k + [m \cdot \log_2 3] + 1$ rows and

$m + [k \cdot \log_3 2] + 1$ columns. We will refer to the image as a *DBNS*-map.

Definition 3: The representation of an arbitrary integer in the DBNS-map using a minimal number of ones is called the minimal double-base number representation (MDBNR).

Definition 4: We will call a cell *(i,j)* *active* if it takes part in the corresponding DBNS-map.

Thus the *MDBNR* is a *DBNS* -map with the minimal number of active cells.

In the next section we discuss techniques for obtaining near MDBNRs with some results on sparseness.

3. Minimal DBNS and Sparseness

The MDBNR is extremely sparse. It is easy to check, for example, that 23 is the smallest nonnegative integer, requiring 3 ones. The smallest integer, requiring 4 ones in its MDBNR is 431, which is a considerable distance from 23. The smallest integer, requiring 5 ones is 18431 (it has, however, 219 different minimal representations). The smallest integer, requiring 6 ones is 3 448 733.

The extreme sparsity of the DBNS is a good measure for potential implementation of many algorithms. Table 1 shows MDBNRs for two randomly selected small integers.

Table 1 A MDBNR for 79 and 110

	1	3	9	27
1	■			
2		■		
4				
8			■	

79

	1	3	9	27
	■			
				■

110

We now present some important theorems relating to the MDBNR. We first determine the average number of non-zero digits required for a given number; we then show that a greedy algorithm can be employed to find near MDBNRs in polynomial time.

Theorem 1: :Any natural number n can be written as a sum of k terms of the form $2^a 3^b$, where $k = O\left(\frac{\log n}{\log\log n}\right)$.

Proof: The idea is to split the interval $[1,n]$ into two parts; apply a greedy method to the larger interval, and a trivial method to the smaller interval. We start with the trivial observation that we can do the job with $k = O$ (log n), simply by taking the 2-adic and 3-adic expansion. Further, we shall always assume, that n is large enough. Thus any number in the interval $[1,m]$ can be written as a sum of terms $2^a 3^b$ with at most O (log m) terms. Tijdeman [15] has shown that there is an absolute constant $C > 0$ such that there is always a number of the form $2^a 3^b$ between $n - \frac{n}{(\log n)^C}$ and n. Now we put $n_0 = n$, and the Tijdeman result implies that there exists a sequence $n_0 > n_1 > n_2 > \ldots > n_l > n_{l+1}$ such that $n_i = 2^{a_i} 3^{b_i} + n_{i+1}$ and $n_{i+1} < \frac{n_i}{(\log n_i)^C}$ for $i = 0,1,2,\ldots,l$.

Here we choose $l = l(n)$ so that $n_{l+1} \le f(n) < n_l$ for some function f to be chosen later. It follows that we can now write n as a sum of k terms of the form $2^a 3^b$, where $k = l(n) + O(\log f(n))$. We now have to estimate $l(n)$ in terms of $f(n)$, and choose an optimal $f(n)$. Note that if $i < l$, then $n_i > n_l > f(n)$, hence $n_{i+1} < \frac{n_i}{(\log n_i)^C} < \frac{n_i}{(\log f(n))^C}$ This implies at once that $f(n) < n_l < \frac{n}{(\log f(n))^{lC}}$, and thus we find $l(n) < \frac{\log n - \log f(n)}{C \log\log f(n)}$. We now take $f(n) = \exp\frac{\log n}{\log\log n}$. This function is the largest possible function (apart from constant) to which we can apply our observation above, to show that any number in the interval $[1, f(n)]$ can be written as a sum of $O\left(\frac{\log n}{\log\log n}\right)$ terms of the form $2^a 3^b$. We want to show that with this function f we also have $l(n) = O\left(\frac{\log n}{\log\log n}\right)$, i.e. there is a constant $D > 0$ such that $l(n) < \frac{1}{D}\frac{\log n}{\log\log n}$. Thus it suffices to show that

$$\frac{\log n - \frac{\log n}{\log\log n}}{C \log\frac{\log n}{\log\log n}} < \frac{1}{D}\frac{\log n}{\log\log n}.$$

This inequality can be rewritten as $D\log\log n + C\log\log\log n < C\log\log n + D$ which is true if $D < C$ with n large enough. Such a D exists, and so the proof is complete.

❑

We now need an algorithm which guarantees the minimality of the representation. First we analyze the exhaustive search algorithm, which obviously finds the total set of minimal representations.

Theorem 2: Let k be a fixed integer. The check whether eqn. (1) has a solution in 2-integers needs at most $O((\log n)^{2k})$ operations.

Proof: The exhaustive search over all 2-integers smaller than or equal to n requires $O\begin{pmatrix} c \cdot (\log n)^2 \\ k \end{pmatrix}$ operations [12], that is $O((\log n)^{2k})$ which is a polynomial of log n.

❑

For large values of k this approach seems impractical. Moreover, if k is not fixed then the exhaustive search algorithm requires exponential time.

A natural technique for finding the *MBDNR* appears to be the following greedy algorithm.

```
Input: positive integer x;
Output: 2-integers $a_i$, such that
        $\sum a_i = x$.
  procedure greedy(x)
  if (x > 0) then do
     {
     find the largest 2-integer $w \le x$;
     write(w);
     x:= x - w;
     greedy(x)
     }
  else exit;
```

Unfortunately, this algorithm does not ensure the minimality of the representation. The smallest x, when the greedy algorithm fails, is 41. It is intuitively clear, however, that the algorithm provides close solutions to the *MDBNR*, moreover it is very easy to implement, while the problem of finding the *MDBNR* would seem to be *NP* -hard.

Theorem 3: :The greedy algorithm terminates on average after $O\left(\frac{\log x}{\log\log x}\right)$ steps.

Proof: Omitted for brevity.

❑

Definition 5: 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. Theorem 1 predicts 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 Fig. 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 MDBNRs of x, in about 80% of the cases. This observation along with Theorem 1 allows an estimate that the greedy algorithm returns a MDBNR with probability $0.8^{\frac{\log x}{\log\log x}}$; fortunately this tends to zero very slowly.

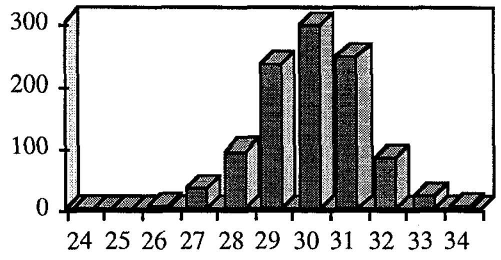


Fig. 1 Occurrence of 2-integers for 215-bit numbers

We also find that the greedy algorithm produces a representation requiring, on average, $O\left(\frac{\log x}{\log\log x}\right)$ 2-integers (from Theorem 3). This NCDBNR provides a sufficiently sparse representation of x to make it very useful.

3.1. An index representation

Since the NCDBNS is very sparse, we can efficiently map the original 2-D representation of Table 1 into an index representation formed by an n-tuple of the 2-integers. As an example, consider the following 215-bit prime number [2].

p_{65}=32769 1329932667 0954996198 8190834461 4131776429 6799294253 9798288533

The greedy algorithm produces 29 2-integers, as below: (78,86 77,82 75,79 42,95 11,109 35,89 128,26 35,78 132,12 56,54 81,34 76,32 76,28 25,51 48,32 52,25 41,26 22,34 8,35 1,32 27,11 4,21 13,10 7,11 15,3 1,9 9,1 0,3 1,0)

4. The NCDBNR Transformation

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. If we once again consider the geometrical interpretation of the numbers in the DBNR, we can find simple identities on special combinations of active cells that allow removal of adjacent active cells. For example, Table 2 uses the identity $2^i3^j + 2^{i+1}3^j = 2^i3^{j+1}$ to remove consecutive cells lying in one column. Table 3 demonstrates the application of the identity, $2^i3^j + 2^i3^{j+1} = 2^{i+2}3^j$, to remove consecutive cells lying in one row. This procedure is akin to the symbolic substitution process used in optical computing. We can generalize this reduction problem using the purely exponential Diophantine eqn. (2), where $l < k$. The problem of solving Diophantine equations such as eqn. (2) has been a subject of investigation over the last two decades [13], although some interesting results were obtained in the 30's and 40's [11].

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

Table 2 Column reduction

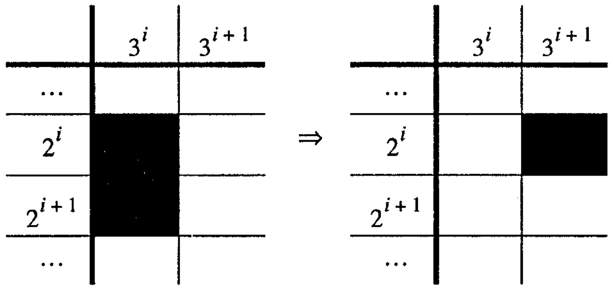
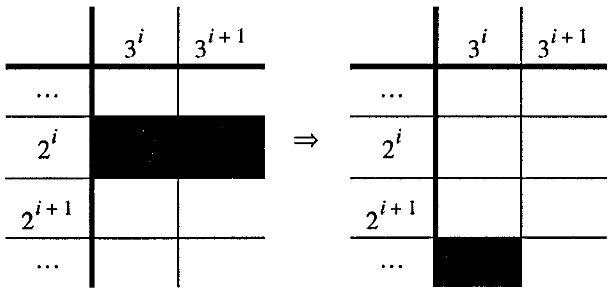


Table 3 Row reduction



We need only consider some special cases for k and l.

4.1. $k = 2, l = 1$

Theorem 4: The Diophantine equation $x + y = z$ where $\mathrm{GCD}(x, y, z) = 1$ 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 [10].

❑

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.

A powerful technique in transcendental number theory (studies of equations such as (2)), described by Baker [14], allows one to conclude that eqn. (2) 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.

5. DBNS-Map Addition and Multiplication

5.1. Addition

Let x and y be two integers in the MDBNR. We note that if x and y contain the element 2^i3^j, then the element $2^{i+1}3^j$ does not exist, therefore addition can computed by simply overlying 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, the approach described in the previous section allows us to find suitable DBNS maps without necessarily requiring a minimal form.

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

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

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

Note, using the MDBNR, 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 2 and 3.) In the worst case, DBNS addition requires 37% fewer 'carries' (removal of consecutive active cells) than binary addition.

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

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

5.2. Multiplication:

Let x and y be integers, represented by DBNS maps in the MDBNR. The MDBNR of their product, z, is an n-tuple of the elements $\left\{2^{i_z}3^{j_z} = 2^{i_x+i_y}3^{j_x+j_y}\right\}$, where the $\{i_x, j_x\}$ and $\{i_y, j_y\}$ are the 2-integer index locations of the active cells in the MDBNRs 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.

6. DBNS Index Calculus

The n-tuple, $\left\{2^{i_z}3^{j_z} = 2^{i_x+i_y}3^{j_x+j_y}\right\}$, 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* MDBNR 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 5: 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: $\{n + m \cdot \omega\}$, $(n, m \in Z)$, 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.

6.1. 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 $\left|x - s_x 2^{b_x} 3^{t_x}\right| < \varepsilon$.

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:

$$x \cdot y = ((s_x + s_y)\mathrm{mod}2, b_x + b_y, t_x + t_y) \qquad (3)$$

$$x/y = ((s_x + s_y)\mathrm{mod}2, b_x - b_y, t_x - t_y) \qquad (4)$$

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

$$\begin{aligned} 2^a 3^b + 2^c 3^d &= 2^a 3^b (1 + 2^{c-a} 3^{d-b}) \\ &\approx 2^a 3^b \Phi(c-a, d-b) \end{aligned} \qquad (5)$$

$$\begin{aligned} 2^a 3^b - 2^c 3^d &= 2^a 3^b (1 - 2^{c-a} 3^{d-b}) \\ &\approx 2^a 3^b \Psi(c-a, d-b) \end{aligned} \qquad (6)$$

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

$$\Phi(x, y) = 1 + 2^x 3^y \approx 2^\alpha 3^\beta \qquad (7)$$

$$\Psi(x, y) = 1 - 2^x 3^y \approx 2^\gamma 3^\delta \qquad (8)$$

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).

6.2. 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, and addition and subtraction uses an identity requiring the look-up of a unary function (e.g. eqn. (7) 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. Whereas the DBNS mapping is onto, the LNS mapping is not. Although the inherent dynamic range compression of the LNS is often touted as an advantage, it is not clear that this compression outweighs the non-linear nature of the error (digital noise) in a system such as a FIR filter. 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 [21]. 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 the 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 3:

Theorem 3: :Given α, β reals and $\varepsilon > 0$, then there exist integers p and q, such that:

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

Proof: See reference [22].

❑

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, LLL reduction, Ferguson-Forcade algorithm, Szekeres algorithm [15], 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 4 summarizes the comparison analysis

Table 4 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.

7. A FIR Filter Example

A FIR filter implements the linear convolution:

$$y_i = \sum_k x_k \cdot h_{i-k} \qquad (9)$$

A well established technique to reduce the complexity of eqn. (9) is to select filter coefficients with a small number of non-zero binary digits. 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 5.

Table 5 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.

7.1. An index calculus IPSP

A major building block for DSP processors is the inner product step processor (IPSP). The IPSP for the index calculus is shown in Fig. 2.

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. Typically, $b' = b$, the dynamic range of the multiplication. Note that the DBNS substitutes the normally required b-bit input ROM with the much smaller ROM and a barrel shifter. This is a considerable hardware reduction.

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.

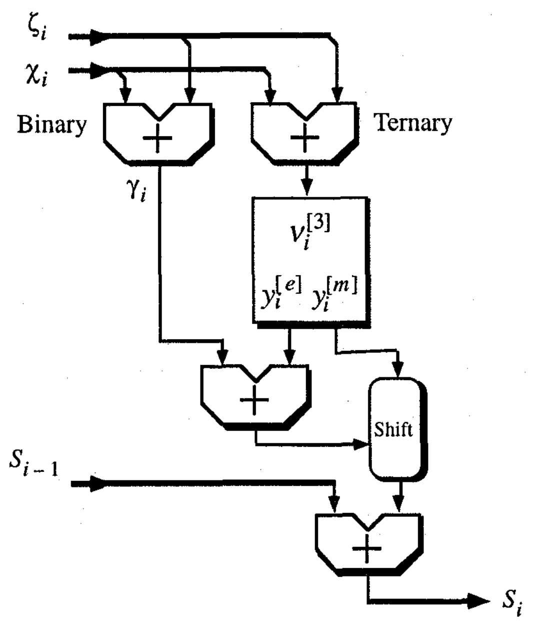


Fig. 2 Index Calculus IPSP

8. Applications to Modular Exponentiation

The application we now present concerns a question of crucial importance in modern cryptography. The problem can be stated as follows: given positive integers A,n and M, compute $C = A^n \pmod M$.

Recently, several papers have appeared which investigate the problem of reducing the time needed to perform a modular exponentiation operation when precomputation is allowed. A typical example is a discrete-log based cryptosystem where we can precompute some powers of A so that the evaluation of $C = A^n \pmod M$, requires a minimal number of modular multiplications (MMs). However, in typical situations, A,n and M are 512 or 1024-bit integers; therefore the precomputation of all powers is obviously impossible. Some of the previously investigated algorithms on this subject [8][19] are based on the assumption that the number of precomputed powers of A is $O(\log n)$ and rely on a proper representation of n; the number of modular multiplications, however, is still $O(\log n)$. The use of the DBNS and a look-up table of $O((\log n)^2)$ size allows us to obtain an asymptotically faster algorithm, as shown below:

Step 1: Precompute $A^{2^i3^j}$ $(modM)$ for all i,j such that $2^i3^j \le n$ and store the values obtained in a look-up table.

Step 2: Find the DBNR of n via the greedy algorithm; that is $n = \sum_{i=1}^{k} 2^{a_i}3^{b_i}$. From Theorem 3 it follows that $k = O\left(\frac{\log n}{\log\log n}\right)$.

Step 3: Multiply (modulo M) the corresponding elements $(a_1,b_1), (a_2,b_2),\ldots,(a_k,b_k)$ from the look-up table.

Because Step 1 is computed off-line, the complexity of this algorithm (from Steps 2 and 3), is $O(k) = O\left(\frac{\log n}{\log\log n}\right)$.

3.1. Comparison with published algorithms

To make a fair comparison among the existing techniques, let us consider that the exponent, n, is a 512-bit integer (the most popular case in practice). The application of the algorithm, based on a hybrid binary-ternary number system [8], requires, on average, 173 modular multiplications. The algorithm based on a ternary-quintary number system [20] requires, on average, 166 MMs. Our new algorithm requires, on average, 57 MMs, but it also requires 83370 stored values. If this storage requirements is too severe for a particular application, the algorithm can be easily modified in order to reduce the memory requirements.

To be more precise, let us assume that the exponent n is written ('folded') into the form: $n = 2^{256}n_1 + n_2$, where n_1 and n_2 are 256-bit integers. Then the computation of $C = A^n(\mathrm{mod}M)$ can be transformed into $C = 2^{2^{256}n_1 + n_2}(\mathrm{mod}M) = A_1^{n_1}(\mathrm{mod}M)A^{n_2}(\mathrm{mod}M)$, where $A_1 = A^{2^{256}}(\mathrm{mod}M)$. A_1 can be precomputed in advance, so we are in position to precompute two look-up tables containing the values of $A^D(\mathrm{mod}M)$ and $A_1^D(\mathrm{mod}M)$, for every D of the form 2^a3^b, smaller than or equal to 2^{256}. Now the total size of the look-up tables is reduced to $2 \cdot \frac{256^2}{3.17} \approx 41347$ values. The average number of MMs is increased to $2 \cdot \frac{256}{\log_2 256} + 1 = 65$; therefore, for the price of eight MMs we reduce the size of the look-up table by a factor of two. Separating the exponent into more parts (for convenience their number should be selected as a power of two) we can obtain further reductions of the total look-up table size. Table 6 gives the expected number of MMs and the required number of precomputed values for our new approach and two other published techniques [19][20].

Our computer experiments show us that the time needed for step 2 (the greedy algorithm) is much less than the execution time for step 3. For example, for a 512-bit exponent the execution time for the greedy algorithm is equal to about 4 modular multiplications; thus this step can be reasonably neglected from the total run-time analysis.

Table 6 Comparison for MMs using precomputations

Algorithms	Number of MMs	Storage
Algorithm1 [20]	166	36027
Algorithm2 [8]	173	83374
DBNS (unfolded exponent)	57	83374
DBNS (2-folded exponent)	65	41347
DBNS (4-folded exponent)	76	20673
DBNS (8-folded exponent)	92	10336
DBNS (16-folded exponent)	117	5168
DBNS (32-folded exponent)	159	2584

4. Conclusions

In this paper we have presented the theory and two applications of a double-base number system. We have also introduced an index calculus which provides the same structure as the LNS without the logarithmic mapping.

We have directed our attention to two possible areas of application: digital signal processing and cryptography. In the former, we use the index calculus. An implementation advantage over the logarithmic number systems is that the index additions and subtractions are reduced in complexity, because the binary and ternary operations are completely independent. In cryptography applications, the DBNS provides an efficient number representation because of its unusual sparsity. For every cryptosystem, based on modular exponentiation with a fixed-base, our proposed algorithm is an efficient alternative to the existing algorithms.

5. 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] Ercegovac,M.D., Lang, T., Nash,J.G., and Chow,L.P. "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", Advances in Computers, vol.6,1965,pp.131-194

[7] G.W.Reitwiesner, "Binary Arithmetic", Advances in Computers, vol.1, 1960,pp.231-308

[8] V.S.Dimitrov and T.V.Cooklev, "Two algorithms for modular exponentiation using nonstandard arithmetic", IEICE Trans. on Fundamentals, 1995, pp.82-87

[9] E.Swartzlander, "Digital optical computing", Applied Optics, vol.25,1986,pp.3021-3032

[10] B.M.M.de-Weger, "Algorithms for Diophantine equations", CWI Tracts-Amsterdam, vol.65, 1989

[11] G.Hardy, "Ramanujan", 1940, Cambridge Univ. Press

[12] R.P.Stanley, "Enumerative Combinatorics", vol.I,Wadsworth & Brooks, 1986.

[13] T.N.Shorey and R.Tijdeman, "Exponential Diophantine equations", 1986,Cambridge University Press

[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] 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

[16] E.E.Swartzlander,Jr. D.V.SChandra, H.T.Nagel,Jr. and S.A.Starks, "Sign/logarithmic for FFT implementation", IEEE Trans. on Computers, vol.C-32, pp.526-534, 1983

[17] A.Brauer, "On addition chains", Bulletin of the American Mathematical Society, vol.45,1939, pp.736-739

[18] P.Erdos, "Remarks on number theory III: on addition chains", Acta Arithmetica, vol.6, 1960, pp.77-81

[19] E.F.Brickell,D.M.Gordon,K.S.McCurley and D.B.Willson, "Fast exponentiation with precomputation", Proc.Eurocrypt'92,Lecture Notes in Computer Science,vol.658,pp.200-207, Springer,1993

[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] D. Lewis, "An accurate LNS arithmetic unit using interleaved memory function interpolator", Proc. of ARITH-11, Windsor, 1993 pp. 2-9.

[22] L.Lovasz, "An algorithmic theory of numbers, graphs and convexity", SIAM Regional Conference Series in Applied Mathematics, vol.50, 1986

6. 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.