

# On Efficient Techniques for Difficult Operations in One and Two-digit DBNS Index Calculus

R. Muscedere, V.S. Dimitrov, G.A. Jullien, W.C. Miller and M. Ahmadi  
VLSI Research Group  
University of Windsor, Ontario Canada N9B 3P4  
musced3@uwindsor.ca

## Abstract

The Double Base Number System (DBNS), using orthogonal bases of 2 and 3, has similar properties to the logarithmic number system (LNS) if an index calculus is used. The DBNS provides more degrees of freedom than the LNS by virtue of both the orthogonal bases and the ability to use multiple digits. As with the LNS, multiplication and division are easy but addition and subtraction are difficult. This paper introduces a technique that uses a map to a DBNS “index sequence domain”, removing the need for the relatively large look-up table solution of the LNS.

## 1. Introduction

Special purpose high performance DSP systems often take advantage of the properties of special number representations. Popular examples are canonic-signed digit representations of filter coefficients for filter implementations with reduced complexity multipliers [1][2]. Examples of other number representations that allow low complexity multiplication operations are the Residue Number System (RNS) [3] and the Logarithmic Number System (LNS) [4]. The LNS uses the logarithm of the number as its representation. The arithmetic now becomes easy for multiplication and division, but addition and subtraction are now difficult. The LNS has been shown to be efficient for pipelined DSP arithmetic units [5].

A number representation that has similar properties to the LNS is the index calculus double-base number system (IDBNS) [6]. The DBNS represents a given integer,  $x$ , with the form:

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

where  $i, j$  are signed integers.

In this paper we will briefly review the IDBNS and then introduce the index sequence domain via a novel technique of inverse conversion to the binary number system. Based on this, we will discuss reduced complexity techniques for the difficult operations of addition and subtraction.

## DBNS index calculus.

The n-tuple,  $\left\{ 2^{i_x} 3^{j_x} = 2^{i_x + i_y} 3^{j_x + j_y} \right\}$ , 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 DBNS representation 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.

## Doing the arithmetic

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  $\epsilon$  is the error allowed, then  $|x - s_x 2^{b_x} 3^{t_x}| < \epsilon$ .

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:

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

$$x/y = ((s_x + s_y) \bmod 2, b_x - b_y, t_x - t_y) \quad (3)$$

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

$$2^a 3^b + 2^c 3^d = 2^a 3^b (1 + 2^{c-a} 3^{d-b}) \quad (4)$$

$$\approx 2^a 3^b \Phi(c-a, d-b)$$

$$2^a 3^b - 2^c 3^d = 2^a 3^b (1 - 2^{c-a} 3^{d-b}) \quad (5)$$

$$\approx 2^a 3^b \Psi(c-a, d-b)$$

We will, of course, precompute and store the functions



tables it is 80):

$$Tbase = (t + tmin) - Tstart + (Quotient - Bstart) \cdot 5 \quad (9)$$

A binary approximation of the DBNS value can now be determined given the *Column*, *Row* and *Tbase*. This method will require only three to four cycles to perform and can be implemented in a feedforward pipelined structure. The value of *Tbase* is first used to address a LUT containing the beginning binary value of the conversion, values of  $3^{Tbase}$ . The range of *Tbase* (*Trange*) will typically be relatively small depending on the application, therefore the LUT will also be small and very manageable (see Table 3).

Table 3: Ternary Base

Ternary Base	Binary Representation
0	1.00000000
1	3.00000000
2	9.00000000
3	27.00000000
4	81.00000000
:	:

The next step of the process uses the *Column* value to further approximate the binary representation. Since each column of the table begins when the indices are a multiple of 8 and -5, the *Column* value is used to determine a multiplying factor (for the  $3^{Tbase}$  beginning value) of  $\left(\frac{256}{243}\right)^{Column}$ .

Again, since *Column* will have a small range, these values can be referenced by a LUT (see Table 4).

Table 4: Column Multiplier

Column	Binary Index	Ternary Index	Binary Representation
0	0	0	1.00000000
1	8	-5	1.05349794
2	16	-10	1.10985791
3	24	-15	1.16923303
4	32	-20	1.23178459
:	:	:	:

The binary values in this table will be between 1 and 3, therefore an optimized multiplier circuit can be used to generate a good binary approximation. The final stage is to multiply by the *Row* factor, which is placed in another LUT. *Row* also has a limited range so the LUT entries will be small and it's content will be of binary representations of 1 to  $2^8 \cdot 3^{-5} = \frac{256}{253} = 1.05349794$  (see Table 5).

The final cycle, if applicable, will adjust the sign of the binary approximation. Since the sign of the DBNS representation is a separate variable and does not depend on the indices a simple conversion to 2's complement if the sign is negative will suffice.

Table 5: Row Multiplier

Row	Binary Index	Ternary Index	Binary Representation
0	0	0	1.00000000
1	-84	53	1.00209031
2	-168	106	1.00418500
3	65	-41	1.01152885
4	-19	12	1.01364326
5	-103	65	1.01576210
6	-187	118	1.01788536
7	46	-29	1.02532941
8	-38	24	1.02747267
9	-122	77	1.02962041
10	111	-70	1.03715028
11	27	-17	1.03931825
12	-57	36	1.04149075
13	-141	89	1.04366779
14	92	-58	1.05130039

This method can be extended to multi-digit DBNS by cascading additional convertors for each digit and summing the final binary representations.

Now that an alternative method for conversion from DBNS representation to binary has been devised, the same approach here can be used to convert a DBNS value into the ISD.

### Extension to ISD

The ISD is similar to a typical order function; however, there are gaps from the overflow due to the index representation range (*B* and *T*). These gaps in the ISD are found between two adjacent numbers in the sequence.

Assuming the *Rows*, *Columns* and *Tbase* have been determined, a simple series of fixed multiplications of these values can derive the ISD representation of the DBNS value.

$$ISD(x) = Tbase(x) \cdot Tmax + Column(x) \cdot Cmax + Row(x) \quad (10)$$

where *Tmax* is the number of DBNS values, including gaps, in a binary group; *Cmax* is the number of entries in one column of the table. For  $B = T = 8$ ,  $Cmax = 15$ ,  $Tmax = 317$  ( $Cmax \cdot 21 + \text{remainder of the ISD table which is } 1$ ). Since the ranges of *Tbase* and *Column* will be small, LUTs can replace the multiplication functions for a more efficient implementation.

The inverse ISD function can be found by reversing the process. Given an ISD value, each operand of the ISD function can be broken down. First the *Tbase* can be obtained by dividing by *Tmax* and finding the quotient; however since *Tmax* is constant, a series of comparisons in hardware implementation will suffice for this. The same process can be done to determine the *Column* by dividing by *Cmax* and *Row* will simply be the remainder of the process. Once all these values are determined two LUTs containing the binary and ternary adjustments for the *Column* and *Row* can be referenced and the *Tbase* can be added to the final ternary index.

Further reference to the forward and reverse ISD pro-

cess will be as  $x = ord(b, t)$  and  $[b, t] = ord^{-1}(x)$  respectively.

### Magnitude comparison

Magnitude comparisons of DBNS values can be performed by comparing the ISD values of the particular digits. If the  $ord(b1, t1) > ord(b2, t2)$  then  $2^{b1} \cdot 3^{t1} > 2^{b2} \cdot 3^{t2}$  and vice-versa.

### 3. Addition and Subtraction

Consider the following example:

$$2^1 \cdot 3^2 + 2^2 \cdot 3^3 = 117$$

$$2^1 \cdot 3^3 + 2^2 \cdot 3^4 = 351 = 117 \cdot 3$$

We can see the ternary exponents on the LHS of the second expression have been increased and therefore the result of the addition will also be increased. What is interesting to note here is that the *Row* and *Column* values derived from the solutions are equal. Based on this we will assume that addition and subtraction can be dual operations in the DBNS.

#### Addition

Given  $x_1$  and  $x_2$  (both single digit, positive DBNS values) our goal is to find  $x_3$  and  $x_4$  such that  $x_0 = x_1 + x_2 = x_3 + x_4 + \epsilon$ . There can be many approximations for  $x_0$ , one of which is optimal (where  $\epsilon$  is closest to 0). Determining the optimal choice is not possible in our target hardware, neither is it necessary. We shall choose a more manageable approach. We will instead make the assumption that  $x_3 \gg x_4$ , in fact we will find the closest approximation of  $x_3$  to  $x_0$  and  $x_4$  will be the remaining difference.

Working in the ISD we can find  $y1 = ord(x1)$  and  $y2 = ord(x2)$ . We will assume that  $x1 > x2$ , therefore  $y1 > y2$ . If this is not the case we can easily exchange the two values of  $x1, y1$  and  $x2, y2$ . This can be determined by comparing  $y1$  and  $y2$ .

Since we assumed addition can be a dual operation, we can find  $\Delta y = y1 - y2$ . Our solution for DBNS addition can be shown by using two specific functions: *MajorAdd(y)* and *MinorAdd(y)*.

$$x3 = ord^{-1}(y1 + MajorAdd(\Delta y))$$

$$x4 = ord^{-1}(y1 - MinorAdd(\Delta y))$$

For the case of  $x3$  the correction factor is added to  $y1$

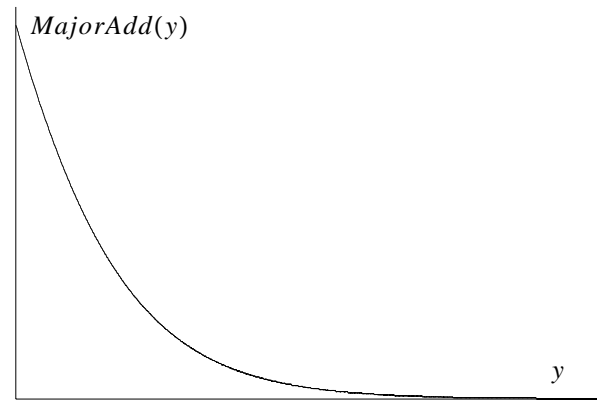
since  $x3$  is expected to be larger than  $x1$ . Conversely, for the case of  $x4$  the correction factor will be subtracted from  $y1$  since  $x1$  is certainly expected to be larger than  $x4$ .

To increase the accuracy of the approximation for  $x0$ ,  $x4$  can be negative for some cases. This would require extra information in both tables to support this.

Experimentally for various values of  $B$  and  $T$  it has been found that if  $\Delta y > 7 \cdot Tmax$  then  $x1 \gg x2$  and  $x2$  will have no effect on  $x1$  for DBNS addition. This is due to the fact that the value of  $x2$  is smaller than the incremental changes at  $x1$  and the other values around it.

The trend of the *MajorAdd* function is shown in Figure 1.

Fig. 1 Trend of the *Major Add* function



The addition of two single-digit DBNS values also offers a method to generate a greedy two-digit DBNS value [6].

#### 3.1. Subtraction

A similar approach can be derived for DBNS subtraction.

Given  $x1$  and  $x2$  (both single digit, positive DBNS values) our goal is to find  $x3$  and  $x4$  such that  $x0 = x1 - x2 = x3 + x4 + \epsilon$ . Again there can be many approximations for  $x0$ , however we shall choose the more manageable approach by assuming that  $x3 \gg x4$ , where  $x3$  is very close to  $x0$ .

Working in the ISD we will again find  $y1 = ord(x1)$  and  $y2 = ord(x2)$  where  $x1 > x2$  and therefore  $y1 > y2$ .

Similarly we find  $\Delta y = y1 - y2$ . Our solution for DBNS subtraction can be shown by using another two specific functions: *MajorSub(y)* and *MinorSub(y)*.

$$x3 = ord^{-1}(y1 - MajorSub(\Delta y))$$

$$x4 = ord^{-1}(y1 - MinorSub(\Delta y))$$

For both of these cases we know that  $x_3$  and  $x_4$  will be smaller than  $x_1$  so the correction factors are subtracted from  $y_1$ .

### 3.2. Addition and Subtraction of signed DBNS

Addition and Subtraction of signed DBNS digits can be easily performed by determining the appropriate action based on the signs of  $x_1$  &  $x_2$  and their order ( $x_1 > x_2$ ). After the selected operation is complete the final signs for  $x_3$  and  $x_4$  may have to be adjusted.

## 4. Examples

### 4.1. DBNS to binary

Given  $b = 70$  and  $t = -42$ , (the proper binary representation for this is 10.789641086) the *Quotient* ( $70 + 128$ , padded to eliminate signed arithmetic) is 24 and the *Remainder* is 6. By looking in row (*Remainder*) 6 of the search table we can determine that second segment contains the proper information. Here,  $Row = 7$ ,  $Bstart = 21$ ,  $Col = 0$  and  $Tstart = 51$ .  $Row$  is given from the table;  $Column$  is  $24 - 21 + 0$  which is 3. The  $Tbase$  is  $(-42+80) - 51 + (24-21) \cdot 5$  which is 2. Referring to Table 3,  $x = 9$  for  $Tbase = 2$ . Next we multiply  $x$  by the  $Column$  (which is 3) reference in Table 4:

$$x = x \cdot \left(\frac{256}{243}\right)^3 = 9 \cdot 1.16923303 = 10.52309727.$$
 Our

final step is to multiply  $x$  by the  $Row$  (which is 7) reference in Table 5:  $x = 10.52309727 \cdot 1.02532941 = 10.789641$ . The solution is clearly very close to the expected value.

### 4.2. DBNS Addition

Given:

$$x_1 = 2^{46} \cdot 3^{-18} \approx 181634 \quad x_2 = 2^{-110} \cdot 3^{80} \approx 113868$$

we find:

$$y_1 = ord(46, -18) = 3494$$

$$y_2 = ord(110, -80) = 3360$$

Since  $y_1 > y_2$ ,  $\Delta y = y_1 - y_2 = 3494 - 3360 = 134$ .

The result of  $MajorAdd(134)$  is 141, which is added to  $y_1$  equaling 3635. The function  $ord^{-1}(3635)$  returns [69,55] which is  $2^{-69} \cdot 3^{55} = 295528$ . The difference between the estimated sum and the actual sum is 26; this value will then be approximated by  $x_4$ . The result of  $MinorAdd(134)$  is -2547 indicating the sign on  $x_4$  will be negative. We subtract this from  $y_1$  and obtain 947. Since:

$$ord^{-1}(947) = [19, -9] \text{ represents } 2^{19} \cdot 3^{-9} = 26.6.$$

therefore our approximation is  $295528 - 26.6 = 295501.4$  which is very close to the actual sum.

## 5. Conclusions

We have discussed the difficult operations of conversion, addition and subtraction in the Index Calculus Double-Base Number System. For the new conversion process presented, three smaller Look-up Tables (LUTs) have replaced a single larger one but with more computation cycles and operations. However, for large values of ternary index dynamic range, this new process will be more efficient in terms of hardware size.

The new techniques for addition and subtraction provide considerable hardware reduction compared to equivalent dynamic range LNS implementations for the same operations. The reductions stem from the much reduced size of the LUTs in the IDBNS implementation.

## 6. Acknowledgment

The authors would like to thank the Natural Sciences and Engineering Council (NSERC) of Canada, the Micronet Network of Centres of Excellence and Gennum Corporation for their financial support and the Canadian Microelectronics Corporation (CMC) for their equipment and software loan and fabrication services.

## 7. References

- [1] T. Arslan, D.H. Horrocks, "A Genetic Algorithm for the Design of Finite Word Length Arbitrary Response Cascaded IIR Digital Filters", Genetic Algorithm in Engineering Systems: Innovations and Applications, no.414, pp.276-281, Sept. 1995.
- [2] F. Ashrafzadeh, B. Nowrouzian, "Crossover and Mutation in Genetic Algorithms Employing Canonical Signed-Digit Number System", Proceeding of the 4th Midwest Symposium on Circuits and Systems, pp.702-705, Aug. 1997.
- [3] M.A.Soderstrand, W.K.Jenkins, G.A.Jullien and F.J.Taylor, Residue number system arithmetic: modern applications in digital signal processing, IEEE Press, 1986
- [4] D.M.Lewis, "An Architecture for Addition and Subtraction of Long Word Length Numbers in the LNS", in IEEE Trans. Computers, Nov. 1990, pp 1326-1336.
- [5] D.M.Lewis, "Interleaved memory function interpolators with application to an accurate logarithmic number system arithmetic unit", IEEE Trans. on Computers, vol.43, 1994, pp.974-983.
- [6] V. S. Dimitrov, G.A. Jullien and W.C. Miller, 1999, "Theory and Applications of the Double-Base Number System", IEEE Trans. Computers, Vol. 48, 10, pp. 1098-1106