ZOT-BINARY: A NEW NUMBERING SYSTEM WITH AN APPLICATION ON BIG-INTEGER MULTIPLICATION

[1]SHAHRAM JAHANI, [2]AZMAN SAMSUDIN

[1,2] School of Computer Sciences, Universiti Sains Malaysia,Pulau Penang, Malaysia

E-mail: [1]shahramjahani@gmail.com, [2]azman@cs.usm.my

ABSTRACT

In this paper we present a new numbering system with an efficient application on Big-Integer multiplication. The paper starts with an introduction to a new redundant positional numbering system known as “Big-Digit Numbering System” (*BDNS*). With *BDNS*, a new non-redundant positional numbering system known as *ZOT-Binary* is proposed. *ZOT-Binary* has a low Hamming weight with an average of 23.8% nonzero symbols, and therefore is highly suitable for Big-Integer calculation, especially for Big-Integer multiplication. To harvest such benefit from the *ZOT-Binary* representation, a new Big-Integer multiplication algorithm, *ZOT-CM*, which is based on the *Classical* multiplication algorithm, is proposed. Our result shows that when compared with the *Classical* multiplication algorithm, *ZOT-CM* is about 12 times faster for multiplying 128 bits numbers and at least 16 times faster for multiplying numbers that are bigger than 32,000 bits long. Our result also shows that *ZOT-CM* is about 20 to 3 times faster than *Karatsuba* multiplication algorithm, for multiplying numbers that are ranging from 128 bits to 32,000 bits long. From the findings, it is clear that *ZOT-CM* is a valuable addition to Big-Integer multiplication algorithm, and it is also believed that *ZOT-Binary* representation can benefit many other Big-Integer calculations.

Keywords: *Numbering system, Big-Integer multiplication, Cryptography, Hamming weight.*

1. INTRODUCTION

Numbering system has always been important in the history of human civilization, and the increasing of number crunching applications signifies numbering system as a crucial necessity more than in the past. To improve arithmetic operations, researchers [1-4] have looked into alternative numbering systems. For example, by proposing signed-binary numbers [1, 3, 4] instead of the standard binary numbers has decreased the number of partial product in *Classical* multiplication algorithm [5] and therefore increased overall algorithm efficiency. Multi-base numbering systems [2, 6, 7] are other similar examples.

The main goal of this paper is to increase the efficiency of Big-Integer multiplication operation which has many important applications, such as cryptography and other scientific calculations. To achieve this goal, modification of known multiplication algorithm running on top of a new numbering system has been identified as the approach. The most popular algorithms for Big-Integers multiplication are *Classical*, *Karatsuba* [8], *Toom-Cook* [9, 10] and *Shonang-Strassen* [11], in which the first two algorithms are more common than the others. Although the complexity of *Classical* multiplication algorithm, $O(n^2)$, is higher than the complexity of other algorithms, Xianjin and Longshu [12] have shown that the *Classical* multiplication algorithm is efficient for multiplying numbers that are less than 255 digits long. On top of its efficiency, it has also been shown that *Classical* multiplication algorithm is efficient in memory utilization. *Karatsuba* multiplication algorithm has a better complexity, $O(n^{1.58})$, compared to *Classical* multiplication algorithm. However *Karatsuba* multiplication algorithm overhead in lower range numbers (less than 255 digits) has caused implementers [12] to combine both algorithms, *Classical* and *Karatsuba*, to achieve better overall algorithm efficiency. In this paper we propose a new numbering system that can help improve the efficiency of *Classical* multiplication algorithm and consequently increase the range of its functionality above the 255 digits threshold.

We review most of the related works on the positional numbering system of radix 2 in Section 2.1. We describe the *Classical* multiplication algorithm in Section 2.2. Section 3 is dedicated to the proposed numbering systems, *BDNS*, and *ZOT-Binary*. In Section 3.4 the proposed multiplication algorithms *mbCM* and *ZOT-CM* are described. We present the experimental results and comparison of the proposed multiplication algorithm with existing methods in Section 4, before concluding in Section 5.

2. NUMBERING SYSTEMS AND ARITHMETIC OPERATIONS

Section 2.1 reviews positional numbering system since it has a significant role in arithmetic operations. Following that, Section 2.2 reviews the *Classical* multiplication algorithm which is fundamental to the work of this paper.

2.1. Positional Numbering System

In positional numbering system, an integer in radix-*r* (or base *r*) can be written as:

$$(a_n \dots a_1 a_0)_r = a_n r^n + \cdots + a_1 r^1 + a_0 \qquad (1)$$

If $0 \le a_i < r$, then this representation is unique. We call a_i as *digit*, and its related set, such as $\{0,1,\dots,\mathrm{r}-1\}$, as *digit set*. Decimal numbers with *r = 10* and $a_i \in \{0,1,\dots,9\}$, and binary numbers with *r = 2* and $a_i \in \{0,1\}$, are the two most known numbering systems. Ternary (*r = 3*), quaternary (*r = 4*), octal (*r = 8*) and hexadecimal (*r = 16*) are the other examples of fixed-radix positional numbering system [5].

2.1.1. Fixed-radix numbering system

Equation (1) represents the fixed-radix numbering system. The weight of each digit (r^i for a_i) in the representation is obtained by multiplying a fixed value (*r*) by the previous digit's weight (r^{i-1}). In the following, our discussion focus on the radix-*2* number systems with different *digit set*. We use *S* to represent the *digit set*.

2.1.1.1. Binary number

The birth of radix-*2* arithmetic is usually attributed to G. W. Leibniz[13], while the binary notation has appeared in 1605 in some unpublished manuscripts of Thomas Harriot [5]. *Digit set* for the binary system is $S = \{0,1\}$.

2.1.1.2. Signed binary (*SB*) number

In *Signed Binary* representation, which is sometimes known as *Ternary Numbering System* [14], the *digit set* is $S = \{0, \pm 1\}$. The redundancy in the numbering system is the result of using 3 symbols in radix-*2*. *Booth* [1], *NAF* [3] and *MOF* [4] algorithms are the examples of *SB* numbering systems that take advantage from this redundancy to decrease the number of operations in the multiplication [1, 15-18], exponentiation [19, 20] and scalar multiplication [1, 3, 4, 17, 18] computations.

Booth [1] in 1959 proposed an algorithm to speed up the *Classical* multiplication computation on computer. The main goal of the original Booth's algorithm and in the higher radix Booth's algorithm [15, 16], is to decrease the number of partial product in the algorithm by decreasing the number of nonzero digits in the binary representation through the use of the symbol "-1" in radix-*2*.

NAF (Non-Adjacent-Form) [3] is another ternary numbering system that was introduced by Reitwiesner in 1960. *NAF* representation is obtained by scanning every 3 bits in a binary number (from right to left) and substituting "$x11$" with "$10\bar{1}$" where "$\bar{1}$" denotes "-1" and x represents any digit. The immediate advantage of the *NAF* representation is a low Hamming weight, which is 1/3 [21]. The generalization of Reitwiesner's *NAF* algorithm can be found in [22, 23].

Another important *SB* representation is *MOF* (Mutual Opposite Form). *MOF* has a Hamming weight of 1/2 [4]. However *MOF* can perform its calculation in both directions (right-to-left and left-to-right) and requires less memory compared to *NAF*.

Note that, the *SB* representation is not always used for optimizing arithmetic operations. For example, in [24] the *Highest-Weight Binary Form* (*HBF*) of scalars and randomization are proposed to resist power analysis in Elliptic Curve Cryptography (*ECC*). There is another ternary numbering system, *Balanced Ternary System* [5, 25], which is very similar to the *SB* numbering system. *Balance Ternary System* uses the same digits set as the *SB* numbering system but with a base of 3.

2.1.1.3. Multi-Base numbering system (*MBNS*)

The simplest system in *MBNS* category is the *double-base numbering system* (*DBNS*). This numbering system was first proposed by Dimitrov et al. [2] for computing the scalar multiplication in *ECC*. Numbers in *DBNS* are represented as

$\sum_i c_i 2^{a_i} 3^{b_i}$ where $c_i \in \{0, \pm 1\}$, and $a_i, b_i \in Z^+$.

If $d_i = c_i \times 3^{b_i}$, then this representation can be transformed to $\sum_i d_i 2^{a_i}$. This shows *DBNS* representation is a radix-*2* numbering system, with an enlarged set of integers. Other *MBNS* [6, 7, 26] use radix-*2* as one of the main bases, and similar to *DBNS*, it can be shown that they are all a radix-*2* numbering system with an enlarger digit set. *Multi-base NAF (mbNAF)*[7] is one of the latest numbering systems that falls under this category.

2.1.1.4. Window sliding method

A number $A = (a_n \dots a_2 a_1 a_0)_b$ where $a_i \in \{0, \dots, b-1\}$ can be represented by

$$A = \left(A_p \dots A_2 A_1 A_0\right)_r \qquad (2)$$

where $r = b^w,\ A_i = \sum_{j=0}^{w-1}(a_{j+iw} b^j)$ and $p = n \bmod r$.

The *digit set* for Equation (2) numbering system is $\{0,1,\dots,b^w-1\}$. In this way, we can represent numbers with fewer digits. The time complexity of algorithms (such as multiplication algorithm) utilizing this windowing method is related to the number of digits and the window size. In general, *Window Sliding Method* increases the arithmetic efficiency in an algorithm. Most popular bases of this group are in the form of $r = 2^w$.

Window Sliding Method can be combined with *SB* representation. *wNAF* [22, 23], *wMOF* [4], *wmbNAF* [7] are the windowing method of *NAF*, *MOF* and *mbNAF*, respectively. Mishra and et al. [27] presented in 2007 a new variant for the window version of *DBNS*. The windowing method improves the efficiency of the original algorithms by having pre-calculated calculations saved in a lookup table (*LUT*) and uses pre-calculated values in computations. However, the size of the window depends on application and is limited to the available memory. For example if we want to store a multiplication *LUT* for *w = 16* bits, we need $2^{16} \times 2^{16} \times (2 \times 16) = 128$ Gbyte of memory, which is not reasonable in most applications, even with today's technology.

2.1.2. Mixed-base numbering system

Fixed-base numbering system was first generalized to a mixed-base numbering system by Georg Cantor in 1869 [5]. Prime number system and factorial number system [5] are examples of mixed-base systems. Equation (3) shows the representation of integers in mixed-base numbering system. There are two sequences of numbers $(a_n \dots, a_2, a_1, a_0)$ and $(b_n \dots, b_2, b_1, b_0)$ where a_n's are the digits and b_n's are the bases. The weight of each digit is a multiple of the weight of previous digit.

$$\begin{Bmatrix} digits \\ radixes \end{Bmatrix} = \begin{Bmatrix} (a_n \dots, a_1, a_0) \\ (b_n \dots, b_1, b_0) \end{Bmatrix} =$$
$$a_n(b_n \dots b_0) + \dots + a_1(b_1 b_0) + a_0 b_0 \qquad (3)$$

2.2. Classical Multiplication Algorithm

Big-Integer multiplication algorithm is one of the fundamental algorithms for scientific computing, in which many mathematicians and computer scientists are continuously making improvements on the subject [10, 11, 28]. As mentioned earlier, among the multiplication algorithms, *Classical* multiplication algorithm is the most used. This is because of its efficiency in multiplying lower range numbers [12]. *Classical* multiplication algorithm is also being used in combination with the other multiplication algorithms to gain overall improvement. Better space complexity is another advantage of the *Classical* multiplication algorithm, resulted in the algorithm being used in memory constrained applications.

The efficiency of the *Classical* multiplication algorithm is related to the numbering system used. The complexity of the *Classical* multiplication algorithm (see Algorithm 1 [5]) is $O(n^2)$, where *n* is the size of the numbers being multiplied. Therefore, the number representation that has fewer digits is theoretically should run faster than the number representation that has more digits in its representation. In addition, the density of nonzero digits in the numbers influences the number of addition that has to be carried out in the *Classical* multiplication algorithm. Equation (4) describes the *Classical* multiplication equation in radix-*r*, in which the Algorithm 1 is based on.

$$A \times B = \sum_{i=0}^{n}\sum_{j=0}^{m} a_i b_j (r^{i+j}). \qquad (4)$$

Algorithm 1: Classical Multiplication CM (A,B)

```
Input:  A = $(a_n \dots a_0)_r$
        B = $(b_m \dots b_0)_r$
Output: C = $(c_{m+n} \dots, c_1 c_0)_r$
1. carry = 0; temp = 0
2. for ( i = 0; i ≤ m + n; i++ )
3.     $c_i = 0$
4. for ( i = 0; i ≤ n; i++ )
5.     for ( j = 0; j ≤ m; j++ )
6.         temp = $c_{i+j} + (a_i \times b_j) + carry$
7.         carry = $\lfloor temp/r \rfloor$
8.         $c_{i+j}$ = temp − carry × r
9. return C
```

In Algorithm 1, Steps 2 and 3 initialize the output array to zero. Multiplicand and multiplier digits are scanned in a row manner in Steps 4 and 5.

"*temp*" is a temporary memory used to keep the summation of the partial product, with the last result being saved in the output array and carry. In Steps 7 and 8, the value of the output and the new carry is calculated from *temp*.

3. BIG-DIGIT NUMBERING SYSTEM (*BDNS*)

In this section we first introduce a new *digit set* of radix-*2*, followed by a new numbering system based on the new proposed *digit set*.

3.1. Definitions

Definition 1: Let $\hat{O} = \{O_1, O_2 \ldots, O_n, \ldots\}$ be a set, where O_n is a sequence of n consecutive binary symbol *"1"*. We call O_n as *Big-One* (*BO*) with length n and $\hat{O}$ as the set of *Big-Ones*.

Example: $\boldsymbol{O_1 = 1_2,\ O_5 = 11111_2}$ **and** $\boldsymbol{\hat{O} = \{O_1, O_2, O_3, \ldots\} = \{1_2, 11_2, 111_2, \ldots\}}$.

Definition 2: Let $\hat{T} = \{T_1, T_3, T_5 \ldots, T_n, \ldots\}$, where T_n is a sequence of $(\frac{n-1}{2})$ consecutive two binary symbols *"10"* with additional *"1"* at the rightmost of the sequence. We call T_n as *Big-Two* (*BT*) with length n and $\hat{T}$ as the set of *Big-Twos*.

Example: $T_1 = 1_2,\ T_5 = 10101_2$ and $\hat{T} = \{T_1, T_3, T_5, \ldots\} = \{1_2, 101_2, 10101_2, \ldots\}$.

Definition 3: *Big-Digits* set ($\hat{D}$) is defined as $\hat{D} = \hat{O} \cup \hat{T} \cup \{0\}$. Each element of $\hat{D}$ is a *Big-Digit* (*BD*). (Note: to prevent redundancy, we remove T_1 and use O_1 in $\hat{D}$ to represent "1_2")

Based on these Definitions (1-3), there are three new possible numbering systems of base 2.

1. **System 1:** $\hat{O} \cup \{0\}$ as the *digit set*.
2. **System 2:** $\hat{T} \cup \{0\}$ as the *digit set*.
3. **System 3:** $\hat{D}$ as the *digit set*.

Given a positive integer *A*, if we represent *A* by using *System 1*, the number of O_1 in the number will increase because the existence of pattern "010". Similarly, if we represent *A* by using *System 2*, the number of T_1 in the number will increase because the existence of pattern "11". *System 3* which is a hybrid of both previous systems produces the best result. The following examples describe the phenomenon. Let $A = 11101010101111_2$ and consists of 10 non-zero digits.

- Based on *System 1:*

$$A = (111)_2 \times 2^{11} + (1)_2 \times 2^9 + (1)_2 \times 2^7 + (1)_2 \times 2^5 + (1111)_2 \times 2^0 = \underbrace{O_3 0 O_1 0 O_1 0 O_1 0000 O_4}_{5\ non-zero\ digits}$$

- Based on *System 2*:

$$A = (1)_2 \times 2^{13} + (1)_2 \times 2^{12} + (101010101)_2 \times 2^3 + (1)_2 \times 2^2 + (1)_2 \times 2^1 + (1)_2 \times 2^0 = \underbrace{T_1 T_1 00000000 T_9 T_1 T_1 T_1}_{6\ \text{non-zero digits}}$$

- Based on *System 3*:

$$A = (11)_2 \times 2^{12} + (101010101)_2 \times 2^3 + (111)_2 \times 2^0 = \underbrace{O_2 00000000 T_9 00 O_3}_{3\ \text{non-zero digits}}$$

Therefore, to reduce the number of nonzero in *Big-Digits*, *System 3* is preferred. Our experiment with 10,000 random bits (see Table 1) shows that *System 3A* yields the best result.

Table 1. Number of nonzero in different Big-Digit numbering systems, based on 10,000 random bits

System 1	System 2	System 3A	System 3B
2,492	3,789	2,186	2,346

As shown in Table 1 there are two versions of *System 3*. In converting a binary number to *Big-Digit* representation, priority can be given to convert *Big-Ones* first followed by *Big-Twos*, or vice-versa. For example, given a number, 1110111_2, *System 3* can produce either $O_3 000 O_3$ (priority on *Big-Ones*) or $O_2 00 T_3 0 O_2$ (priority on *Big-Twos*). Note that, the first representation has less nonzero *Big-Digits*. Similar finding is observed from Table 1, where *System 3A* which supports *Big-One* priority out-performed *System 3B* which supports *Big-Two* priority in terms of producing fewer digits. From the examples above, it is clear that when converting from binary to *Big-Digit*, the priority must be given in converting *Big-Ones*, followed by *Big-Twos*. With this information, we can define a unique *Big-Digit* representation that supports the minimum nonzero *Big-Digits*.

Definition 4: A sequence of *Big-Digits* is called *Big-Digits* representation of *A* if and only if

$$A = (a_n, \ldots, a_1, a_0)_2 = a_n 2^n + \cdots + a_0 \qquad (5)$$

where $a_i \in \hat{D}$.

This numbering system is called *Big-Digit numbering system* (*BDNS*).

3.2. ZOT-Binary: A canonic Big-Digits numbering system

Definition 5: A sequence of *Big-Digits* $(x_k \dots, x_1, x_0)_2$ is known as *ZOT-Binary* representation if and only if for every two "neighboring" nonzero *Big-Digits* x_q and $x_{p<q}$, where length of x_p is *n*:

1. $q \geq p + n + 2$, when x_p and x_q are both either *Big-Two* or O_1 .
2. $q \geq p + n + 1$, for other cases.

Big-Digit $A = O_1 0000 T_2 0000 O_5$ is not a *ZOT-Binary* representation since $x_0 = O_5$ $(p = 0, n = 5)$ and $x_5 = T_2$ $(q = 5)$ and therefore $(q = 5) < (p + n + 1 = 6)$ which does not satisfy Condition 2 of Definition 5.

Big-Digit $B = O_1 0 O_1 00000 T_3$ is not a *ZOT-Binary* representation since $x_6 = O_1$ $(p = 6, n = 1)$ and $x_8 = O_1$ $(q = 8)$ and therefore $(q = 8) < (p + n + 2 = 9)$ which does not satisfy Condition 1 of Definition 5.

Definition 5 introduces the *ZOT-Binary* representation. The representation is named *ZOT-Binary* to highlight the base, which is base two, and the digits used in the representation, which are Zero, *Big-One* and *Big-Two*.

Theorem 1. Every non-negative integer *A* has a *unique* representation in *ZOT-Binary*.

Proof. Assume a *ZOT-Binary* representation of integer $A = (a_m, \dots, a_p, \dots, a_1, a_0)_2$ is not unique, and therefore let

$B = (b_m, \dots, b_p, \dots, b_1, b_0)_2$ and
$C = (c_m, \dots, c_p, \dots, c_1, c_0)_2$

be the two different *ZOT-Binary* representations of integer *A*. Compare *B* and *C* by scanning from right to left, *Big-Digit* by *Big-Digit*, to find the first *Big-Digit* which is not the same in *B* and *C*. Let the position of the first non-similar *Big-digit* denoted by *p*. Subsequently, we can represent *B* and *C* as follow:

$$B = \sum_{i=0}^{m} b_i \times 2^i = \overbrace{\sum_{i=p}^{m} b_i \times 2^i}^{B_L} + \overbrace{\sum_{i=0}^{p-1} b_i \times 2^i}^{B_R} \quad (6)$$

$$C = \sum_{i=0}^{m} c_i \times 2^i = \overbrace{\sum_{i=p}^{m} c_i \times 2^i}^{C_L} + \overbrace{\sum_{i=0}^{p-1} c_i \times 2^i}^{C_R} \quad (7)$$

As mentioned above, $\forall i \in \{0, p-1\}, b_i = c_i$. Therefore from (6) and (7) we can conclude that

$B_R = C_R$ and $B_L = (B - B_R) = (C - C_L) = C_R$.

Therefore

$$\sum_{i=p}^{m} b_i \times 2^i = \sum_{i=p}^{m} c_i \times 2^i \quad \text{where} \quad c_p \neq b_p. \quad (8)$$

In Table 2, we listed all valid cases of c_p and b_p (symmetrical cases have been ignored) and we showed that for each case, there is a contradiction. Therefore, by contradiction, the *ZOT-Binary* represents non-negative integers uniquely.

Table 2. Summary Of Contradictions For Two Different ZOT-Binary Representations

b_p	c_p	Conditions		Contradictions
o	O_m or T_m			$a_p = 0$ *in* B_L $a_p = 1$ *in* C_L
O_m	O_n			$a_{p+m} = 0$ *in* B_L $a_{p+m} = 1$ *in* C_L
O_m	T_n	$m = 1$	$z = 1$	$a_{p+3} = 1$ *in* B_L $a_{p+3} = 0$ *in* C_L
			$z > 2$	$a_{p+2} = 1$ *in* B_L $a_{p+2} = 0$ *in* C_L
		$m \geq 2$		$a_{p+1} = 1$ *in* B_L $a_{p+1} = 0$ *in* C_L
T_m	T_n		$z = m + 1$	$a_{p+m+2} = 1$ *in* B_L $a_{p+m+2} = 0$ *in* C_L
			$z > m + 1$	$a_{p+m+1} = 0$ *in* B_L $a_{p+m+1} = 1$ *in* C_L

- *z* denotes the number of zeros on the left of *Big-Digits* in the first column.
- *p, q, m* and *n* are positive integers and *m*<*n*.

3.3. ZOT-Binary Conversion Algorithm

Based on Definition 5, suppose $m = q - (p + n)$, which indicates that m is the amount of zeros between x_p and x_q in the binary representation. Therefore we can conclude that two nonzero neighboring *Big-Digits* in binary representation satisfy the conditions of *ZOT-Binary* representation if the number of zeros between them is at least two, in which either one or both of them are *Big-Twos* or O_1. The *ZOT-Binary* conversion algorithm (see Algorithm 2) described below, is using this property to convert a binary number to *ZOT-Binary* representation.

Let $A = (a_n \dots, a_2, a_1, a_0)_2$ be a binary number and let the *ZOT-Binary* representation of A be $(c_n \dots, c_2, c_1, c_0)_2$, where $c_i = (c_{it}, c_{il})$ represents the *i*-th *Big-Digit* in *A*. c_{it} denotes the type of c_i and c_{il} denotes the length of c_i. Following are the steps to convert a binary number to its *ZOT-Binary* representation.

Algorithm 2: *ZOT-Binary* Conversion Algorithm (right-to-left)

Input: $A = (a_n \dots, a_1, a_0)_2$ is a binary number
Output:$C = (c_n \dots, c_1, c_0)_2$ where , $c_i = (c_{it}, c_{il})$ is the *ZOT-Binary* representation of A

```
1.  $a_{n+1} = 0$;
2.  for $(i = 0 \text{ to } n + 1; i++)$
3.      if $a_i = 1$ then
4.          $c_{it} = (a_{i+1} + 1); P = i$;
5.          if $(c_{Pt} = 2)$ then
6.              while $(a_i = 1)$
7.                  $i++; c_{Pl}++; c_{it} = 0$;
8.          else
9.              while $(a_{i+1}a_i = 01)$
10.                 $I += 2; c_{Pl} += 2$;
                    $c_{it} = 0; c_{(i-1)t} = 0$;
                    $i--;\ c_{Pl}--$;
11. if $(c_{Pl} = 1)$ then
12.     $c_{Pt} = 2$;
13. return C
```

Step 1: Initializing: Set all c_{it} to zero.

Step 2: Identifying the position of nonzero in *BD*: Scan the binary number from right to left to identify the first nonzero bit a_i. This indicates the beginning of a new nonzero in *BD*.

Step 3: Identifying the type of the *BD*: The *Big-Digit*, a_{i+1}, can identify the type for a_i. If $a_{i+1} = 0$ then the type is *BT* ($c_{it} = 1$), otherwise the type is *BO* ($c_{it} = 2$).

Step 4: Identifying the length of the *BD*: If the type of a_i is *BO* then we count all of 1's before the first zero. However, if the type of a_i is *BT* then we count the number of adjacent '01'. The length of *BT* is double of this number minus one. For consistency, we replace all occurrence of T_1 with its equivalent O_1 .

Step 5: Completing the conversion: Repeating Steps 2-4 until the last bit.

We can also represent a *ZOT-Binary* number in a mixed-base number representation. Such representation is very useful for some calculations such as multiplication, since we can have a more compact representation of *ZOT-Binary* by removing all the zeros. The following describes the conversion of *ZOT-Binary* to its mixed-base form.

$$A = \overbrace{(1010100011100011111)_2}^{\text{binary}}$$
$$= \overbrace{(T_5 00000 O_3 000000 O_4)_2}^{ZOT-binary}$$
$$= \overbrace{\begin{Bmatrix}(T_5, O_3, O_4)\\(2^6, 2^7, 2^0)\end{Bmatrix}}^{\text{mixed-based } ZOT-binary}$$

Algorithm 3, a modified version of Algorithm2, describes the conversion process of converting a binary number to a mixed-base *ZOT-Binary* representation. These changes are mainly related to the relative position calculation of two neighboring nonzero *BD*. In this algorithm, P_k indicates the relative position of c_k and c_{k-1}.

Algorithm 3: Mixed-Base *ZOT-Binary* Conversion Algorithm (right-to-left)

Input: $A = (a_n \dots, a_1, a_0)_2$

Output: $C = \begin{Bmatrix}(c_k \dots, c_1, c_0)\\(2^{p_k} \dots, 2^{p_1},, 2^{p_0})\end{Bmatrix}$ where

$c_i = (c_{it}, c_{il})$ and p_i is position of c_i

```
1.  $a_{n+1} = 0$; k=0; iold=0;
2.  for $(i = 0 \text{ to } n + 1; i++)$
3.      if $a_i = 1$ then
4.          $C_{kt} = (a_{i+1} + 1)$;
                $p_k = i - \text{iold}; \text{iold} = i$;
5.          if $(c_{kt} = 2)$ then
6.              while $(a_i = 1)$
7.                  $i++; c_{kl}++$;
8.          else
9.              while $(a_{i+1}a_i = 01)$
10.                 $i += 2; c_{kl} += 2$;
11.             $i--;\ c_{kl}--$;
12.             if $(c_{kl} = 1)$ then
13.                 $c_{kt} = 2$;
14. return C
```

Journal of Theoretical and Applied Information Technology
10th February 2013. Vol. 48 No.1
© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The reverse conversion, converting *ZOT-Binary* representation to binary is relatively simple. The conversion can be done by applying these replacements:

$$\overbrace{0\dots0}^{n-1}O_n \rightarrow \overbrace{1\dots1}^{n} \quad \text{and} \quad \overbrace{0\dots0}^{n-1}T_n \rightarrow \overbrace{101\dots101}^{n} \quad (9)$$

Therefore the procedure for converting a mixed-base *ZOT-Binary* representation *A* to its binary equivalent can be described as follows:

$$A = \left\{\begin{matrix}(a_k \dots, a_1, a_0)\\(2^{p_k} \dots, 2^{p_1},, 2^{p_0})\end{matrix}\right\}$$

$$= \left(a_k \dots a_i \overbrace{0\dots0}^{p_i} a_{i-1} \dots a_0 \overbrace{0\dots0}^{p_0}\right) \quad (10)$$

where all a_i in Equation (10) are replaced with their equivalent binary values as shown in Equation (9). For example:

$$A = \left\{\begin{matrix}(O_3, T_3, O_5)\\(2^5, 2^7,, 2^4)\end{matrix}\right\}$$

$$= \left(O_3 \overbrace{00000}^{5} T_3 \overbrace{0000000}^{7} O_5 \overbrace{0000}^{4}\right)$$

$$= (1110010100111110000)_2.$$

3.4. Multiplication On BDNS

This subsection explains the process of performing multiplication in *BDNS*. *Classical* multiplication algorithm that runs on *ZOT-Binary* is also described in this subsection.

3.4.1. Big-Digits multiplication

In many multi-digits multiplication implementations, multiplication time tables are used. These tables are a pre-computed lookup-table (*LUT*) that saves the information on digit by digit multiplications. *Window-based numbering systems* as describe earlier uses *LUT* in its calculations. *LUT* is also being used to improve calculation speed as documented in [29-31]. In the same way, we propose three multiplications *LUT*-based on *Big-Digits*, they are: *Big-One-Big-One multiplication LUT (BOBO-MLUT), Big-Two-Big-Two multiplication LUT (BTBT-MLUT),* and *Big-One-Big-Two multiplication LUT (BOBT-MLUT).*

Tables 3, 4 and 5 show the corresponding multiplication tables, which we call them as *Big-Digits multiplication LUT (BD-MLUT)*.

Table 3. Big-One-Big-One MLUT (BOBO-MLUT)

×	**1**	**11**	**111**	**...**
1	1	11	111	**...**
11	11	1001	10101	**...**
111	111	10101	110001	**…**
…	**…**	**…**	**…**	**…**

Table 4. Big-Two-Big-Two MLUT (BTBT-MLUT)

×	**1**	**101**	**...**
101	101	11001	**...**
10101	10101	1101001	**...**
1010101	1010101	1101100001	**…**
…	**…**	**…**	**…**

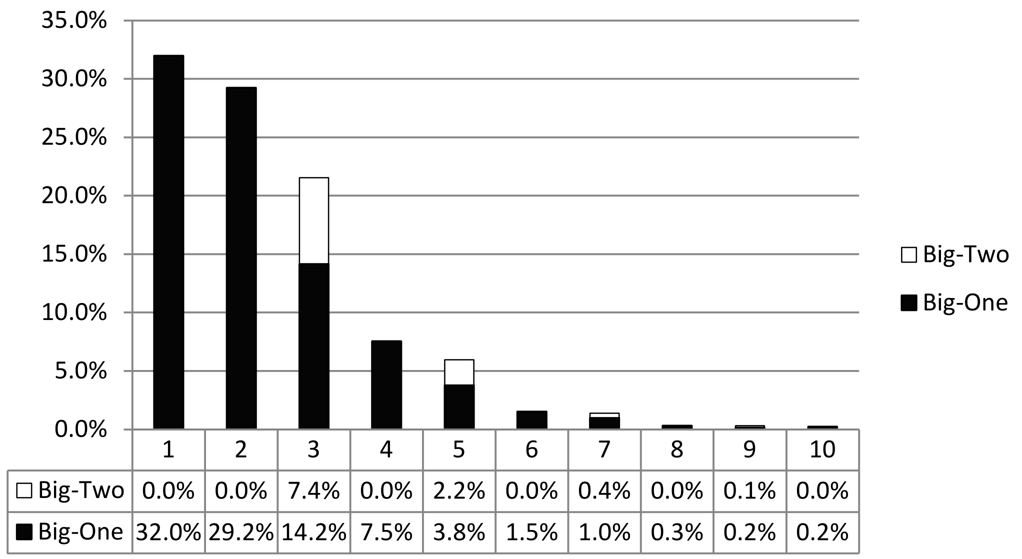
Table 5. Big-One-Big-Two MLUT (BOBT-MLUT)

×	**1**	**11**	**...**
101	101	1001	**...**
10101	10101	111111	**...**
1010101	1010101	11111111	**…**
…	**…**	**…**	**…**

In general, rows and columns in Tables 3-5 are unlimited and therefore tables size do not limit to a certain value. One advantage of *BD-MLUT* is that it grows relatively slower compared to decimal or binary multiplication time-tables, which suggest that storing *BD-MLUT* requires notably less memory comparing to storing decimal or binary multiplication tables. Nevertheless storing big size tables will put a constraint on a computer memory. To solve this problem, many implementation such as in [30, 31] divide the numbers into smaller parts so that smaller *LUTs* are used.

For *Big-Digits*, we had analyzed 10,000 random bits representing random integers and found that the following five *Big-Digits*, 1, 11, 111, 1111 and 101 occurs 90.3% of the time (see Figure. 1).

This finding has a big implication in determining the size of *BD-MLUT* that we need to use effectively for our multiplication algorithm. A small size of *BD-MLUT* of 5 rows by 5 columns is enough to capture almost 90.3% of random *Big-Digits*. For the rest of the 9.7% cases, we need to run the following conversion which will break the *Big-Digits* into the identified five *Big-Digits* mentioned above.



	1	2	3	4	5	6	7	8	9	10
□ Big-Two	0.0%	0.0%	7.4%	0.0%	2.2%	0.0%	0.4%	0.0%	0.1%	0.0%
■ Big-One	32.0%	29.2%	14.2%	7.5%	3.8%	1.5%	1.0%	0.3%	0.2%	0.2%

Figure 1: Nonzero Big-Digits distribution in ZOT-Binary representation based on 10,000 random bits

In general, suppose the length of a Big-Digit is n and BD-MLUT support only Big-Digits less than m, then the following conversions can be applied.

Let $p = n \bmod m$ and $n = k \times m - p$, then

$$\overbrace{0 \dots 0}^{n-1} O_n = \overbrace{0 \dots 0}^{p-1} O_p \underbrace{\overbrace{0 \dots 0}^{m-1} O_m \dots \overbrace{0 \dots 0}^{m-1} O_m}_{k} \ .$$

Or let $p = n \bmod (m+1)$ and $n = k \times (m + 1) - p$, then

$$\overbrace{0 \dots 0}^{n-1} T_n = \overbrace{0 \dots 0}^{p-1} T_p \underbrace{\overbrace{0 \dots 0}^{m} T_m \dots \overbrace{0 \dots 0}^{m} T_m}_{k} \ .$$

The memory requirement to store the *MLUT* for *ZOT-CM is* only 25 byte as shown below:

$$\text{Size of } MLUT = \left(\underbrace{5}_{\text{rows}} \times \underbrace{5}_{\text{columns}} \times 8 \right) = 25 \text{ byte.}$$

Table 6 shows another perspective of the *Big-Digits* distribution in range of 128 bits to 32 kbits random integer. It indicates that *ZOT-Binary* representation reduce the percentage of nonzero symbols to 21.86%, while the average of Hamming weight for *MOF* and *NAF* is 50% [4] and 33% [21] respectively.

Table 6. Distribution of nonzero Big-Digits in ZOT-Binary representation in range of 128 bits to 32 kbits

Length	128 bits	256 bits	512 bits	1 kbits	2 kbits	4 kbits	8 kbits	16 kbits	32 kbits	Average
$(\frac{nonzero\ BD}{Length})$ %	21.1	22.7	21.9	21.2	21.7	22.0	21.6	21.8	22.0	21.8

3.4.2. mb-CM: A modified mixed-base Classical multiplication algorithm

Let $A = \left\{ \begin{matrix} (a_n \dots, a_2, a_1, a_0) \\ (r^{p_n} \dots, r^{p_2}, r^{p_1}, r^{p_0}) \end{matrix} \right\}$ and

$B = \left\{ \begin{matrix} (b_m \dots, b_2, b_1, b_0) \\ (r^{q_m} \dots, r^{q_2}, r^{q_1}, r^{q_0}) \end{matrix} \right\}$

be the two mixed-base integer multiplicands such that p_i, q_i, m, n and r are integers. Integer A and B can be written as series, as shown by Equations (11) and (12), respectively.

$$A = \sum_{i=0}^{n} a_i (r^{p_i} \times \dots r^{p_1} \times r^{p_0}) \\ = \sum_{i=0}^{n} a_i (r^{\sum_{j=0}^{i} p_j}) \qquad (11)$$

$$B = \sum_{k=0}^{m} b_k (r^{q_k} \times \dots r^{q_1} \times r^{q_0}) \\ = \sum_{k=0}^{m} b_k (r^{\sum_{l=0}^{k} q_l}) \qquad (12)$$

Multiplying A and B, we have:

$$A \times B = \left(\sum_{i=0}^{n} a_i \left(r^{\sum_{j=0}^{i} p_j}\right)\right) \times \left(\sum_{k=0}^{m} b_k (r^{\sum_{l=0}^{k} q_l})\right)$$

Then

$$A \times B = \left(\sum_{i=0}^{n} a_i (r^{P_i})\right) \times \left(\sum_{k=0}^{m} b_k (r^{Q_k})\right)$$

Where $P_i = \sum_{j=0}^{i} p_j$ *and* $Q_k = \sum_{l=0}^{k} q_l$.

Then

$$A \times B = \sum_{i=0}^{n} \sum_{k=0}^{m} a_i (r^{P_i}) \times b_k (r^{Q_k})$$
$$= \sum_{i=0}^{n} \sum_{k=0}^{m} a_i b_k r^{(P_i + Q_k)} \quad (13)$$

Series in Equation (13) has a similar structure to the series in Equation (4). Therefore, Algorithm 1 can be generalized to Algorithm 4 to support the mixed-base *Classical* multiplication and result is saved in radix-*r*.

Algorithm 4: Mixed-Base *Classical* Multiplication $\mathrm{mbCM}(A_{(r^{p_i})}, B_{(r^{q_i})})$

Input: $A = \left\{ \begin{matrix} (a_n \dots, a_2, a_1, a_0) \\ (r^{p_n} \dots, r^{p_2}, r^{p_1}, r^{p_0}) \end{matrix} \right\}$ is a mixed-base number in terms of (r^{p_i})

$B = \left\{ \begin{matrix} (b_m \dots, b_2, b_1, b_0) \\ (r^{q_m} \dots, r^{q_2}, r^{q_1}, r^{q_0}) \end{matrix} \right\}$ is a mixed-base number in terms of (r^{q_i})

Output: $C = \left(c_{(\sum_{i=0}^{n} p_i + \sum_{j=0}^{m} q_j)} \dots, c_2, c_1, c_0\right)_r$

```
1.  carry = 0
2.  for( i = 0; i ≤ n; i++)
3.        I+= p_i;
4.        for ( j = 0; j ≤ m; j++)
5.              J+= q_i;
6.              temp = $(c_{I+J+q_{(i+1)}} \dots c_{I+J}) + (a_i \times b_j)_r$ + carry;
7.              carry = $\lfloor temp / r^{q_{(i+1)}} \rfloor$
8.              $(c_{I+J+q_{(i+1)}} \dots c_{I+J})$ = temp − carry × $r^{q_{(i+1)}}$;
9.  $(c_{I+J+q_m+p_i} \dots c_{I+J+q_m})$ = carry;
10. return C
```

There are two main differences in Algorithm 4 over the normal *Classical* multiplication algorithm. First, Algorithm 4 ignores zeros between every two *Big-Digits* (Steps 3 and 5) in its calculation. Second, digits in this algorithm are processed in a group (Steps 6, 7 and 8). Both modifications will additionally speed-up the multiplication calculation compared to the normal *Classical* algorithm.

3.4.3. ZOT-CM: Classical multiplication on ZOT-Binary representation

By substituting $r = 2$ in Algorithm 4, we can modify Algorithm 4 to Algorithm 5 that supports *ZOT-Binary* numbers. We call this algorithm as *ZOT-CM*. Let the sequence of *A* and *B* be the two *ZOT-Binary* numbers.

Algorithm 5: *Classical* Multiplication for *ZOT-Binary* Numbers ZOT-CM(A , B)

Input: $A = \left\{ \begin{matrix} (a_n \dots, a_2, a_1, a_0) \\ (2^{p_n} \dots, 2^{p_2}, 2^{p_1}, , 2^{p_0}) \end{matrix} \right\}$ is a *ZOT-Binary* number

$B = \left\{ \begin{matrix} (b_m \dots, b_2, b_1, b_0) \\ (2^{q_m} \dots, 2^{q_2}, 2^{q_1}, 2^{q_0}) \end{matrix} \right\}$ is a *ZOT-Binary* number

Output: $C = (c_{p_n+q_m} \dots, c_2, c_1, c_0)_2$

```
1.  carry = 0
2.  for( i = 0; i ≤ n; i++)
3.        I+= p_i;
4.        for ( j = 0; j ≤ m; j++)
5.              J+= q_i ;
6.              temp = $(c_{I+J+q_{(i+1)}} \dots c_{I+J}) + (a_i \times b_j)_2$ + carry ;
7.              carry = $\lfloor temp / 2^{q_{(i+1)}} \rfloor$
8.              $(c_{I+J+q_{(i+1)}} \dots c_{I+J})$ = temp − carry × $2^{q_{(i+1)}}$;
9.  $(c_{I+J+q_m+p_i} \dots c_{I+J+q_m})$ = carry ;
10. return C
```

4. EXPERIMENTAL RESULTS

In Section 3.4, it has been shown that the average of nonzero *Big-Digit* for relatively large sample of random numbers is about 24%. Since the complexity of the *Classical* multiplication is $O(n^2)$, then theoretically, the execution time of *ZOT-CM* compare to the *Classical* multiplication algorithm must be:

$$\frac{\text{Partial multiplication of ZOT−CM}}{\text{Partial multiplication of CM}} = \frac{0.24n \times 0.24n}{n^2} \cong 0.058.$$

Table 7 shows the execution time of the *Classical*, *Karatsuba* and *ZOT-CM* multiplication algorithms against different bit length numbers which are randomly generated. Each value in Table 8 is based on an average of 20 actual readings and the conversion overhead had already been included in the readings. Experiment was conducted on AMD Phenom (TM), 9950, Quad core processor, 2.6 GHz, 3.25GB RAM, with Windows XP (Service Pack 2) Professional and Dev-C++ version 4.9.9.2 compiler.

Table 7. Execution Time (Msec) Of Different Multiplication Algorithms

Algorithm \ Length	128 bits	256 bits	512 bits	1 kbits	2 kbits	4 kbits	8 kbits	16 kbits	32 kbits
Classical	0.1140	0.451	1.793	7.163	28.66	114.70	458.8	1,916.7	7,703
Karatsuba	0.1726	0.522	1.582	4.753	14.29	43.00	129.0	388.2	1,165.6
ZOT-CM	0.0087	0.030	0.111	0.427	1.68	6.65	26.6	108.4	434.92

Table 8. Execution Time Ratio Of ZOT-CM Multiplication Algorithm Against Classical And Karatsuba Multiplication Algorithms

Ratio \ Length	128 bits	256 bits	512 bits	1 kbits	2 kbits	4 kbits	8 kbits	16 kbits	32 kbits
ZOT-CM / CM	8%	7%	6%	6%	6%	6%	6%	6%	6%
ZOT-CM / KM	5%	6%	7%	9%	12%	15%	21%	28%	37%

Tables 7 and 8 show that *ZOT-CM* multiplication algorithm performs better than the *Classical* and *Karatsuba* multiplication algorithms, for multiplying integers in the range of 128 bits – 32 kbits. The execution time of *ZOT-CM* multiplication algorithm is 8% of the *Classical* multiplication algorithm for 128 bits integers, and this ratio decreases to 6% for 32,000 bits numbers. For multiplying numbers that are bigger than 32,000 bits, the ratio should be 6% or lower.

ZOT-MC multiplication algorithm is also faster than the *Karatsuba* algorithm for multiplying numbers in the range of 128 – 32,000 bits. For 128 bits integers, the execution time of *ZOT-CM* multiplication algorithm is about 5% of the *Karatsuba* multiplication algorithms execution time, and the ratio gradually increases to 37% for 32,000 bits integers. The reason for the increment in the ratio is because *Karatsuba* algorithm posses a better algorithm complexity than the *ZOT-CM* algorithm. However, from the readings found in Tables 7 and 8, it is expected that *Karatsuba* multiplication algorithm will out-perform *ZOT-CM* multiplication algorithm only when multiplying numbers that are bigger than 442.6 kbits long.

5. CONCLUSION AND FUTURE WORK

Having new perspective to existing numbering systems might reveal some hidden advantages. Based on this believe a new positional numbering system (*BDNS*) and its canonic numbering system (*ZOT-Binary*) were created. *BDNS* and *ZOT-Binary* representation was extracted from the binary representation and posses some interesting properties, such as lower percentage of nonzero symbols. In this paper we are focusing on Big-Integer multiplication on *ZOT-Binary*. The modified *Classical* multiplication algorithm, *ZOT-CM*, increases the efficiency of the multiplication algorithm in such way that *ZOT-CM* not only can replace *Classical* multiplication algorithm but also can replace *Karatsuba* multiplication algorithm when multiplying numbers in a certain range. It is believe that many other calculations that depend on binary systems can benefit from the *ZOT-Binary* representation.

6. ACKNOWLEDGMENT

We are pleased to make an acknowledgement to the Universiti Sains Malaysia, for their support on this research (Grant number: 1001/PKOMP/817059).

REFERENCES

[1] A. D. Booth, "A signed binary multiplication technique," *Quarterly J. Mechanical and Applied Math*, vol. 4, pp. 236-240, 1951.

[2] V. Dimitrov, L. Imber, and P. K. Mishra, "Efficient and secure elliptic curve point multiplication using double-base chains," *in Advances in Cryptology, ASIACRYPT'05, ser.*

Lecture Notes in Computer Science, vol. 3788, pp. 59-78, 2005.

[3] G. W. Reitwiesner, "Binary arithmetic," *Advances in Computers* vol. 1, 1960.

[4] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, "Signed binary representations revisited," in *Advances in Cryptology - Crypto 2004, Proceedings*. vol. 3152, M. Franklin, Ed. Berlin: Springer-Verlag Berlin, 2004, pp. 123-139.

[5] E. Knuth, *The Art of Computer Programming* vol. 2: Addison-Wesley, 1997.

[6] V. Dimitrov, L. Imber, and P. K. Mishra, "The double-base number system and its application to elliptic curve cryptography," *Mathematics of Computation*, vol. 77, pp. 1075-1104, 2008.

[7] P. Longa and A. Miri, "New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems (Extended Version)," *in Cryptology ePrint Archive, Report 2008/052*, 2008.

[8] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady (English translation)*, vol. 7, pp. 595-596, 1963.

[9] A. Cook, "On the Minimum Computation Time of Functions," Harvard: Harvard University, May 1966.

[10] A. L. Toom, "The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers. ," *Soviet Mathematics* vol. 3, pp. 714-716, 1963.

[11] A. Schonhage and V. Strassen, "Schnelle Multiplikation großer Zahlen," *Computing in Science & Engineering*, vol. 7 pp. 139-144, 1971.

[12] F. Xianjin and L. Longshu, "On Karatsuba Multiplication Algorithm," in *Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium on*, 2007, pp. 274-276.

[13] G. W. Leibniz, "Memoires de 1," *Academie Royale des Sciences*, pp. 110-1161, Paris 1703.

[14] R. Hashemian, "A new number system for faster multiplication," in *Circuits and Systems, 1996., IEEE 39th Midwest symposium on*, 1996, pp. 681-684 vol.2.

[15] H. Sam and A. Gupta, " A generalized multibit recoding of two's complement binary numbers and its proof with applications in multiplier implementations," *IEEE Trans. Computers*, vol. 39, pp. 1006-1015, 1990.

[16] S. Vassiliadis, E. M. Schwartz, and D. J. Hanrahan, "A general proof for overlapped multiple-bit scanning multiplications," *IEEE Trans. Compurers*, vol. 38, pp. 172-183, 1989.

[17] P. E. Madrid, B. Millar, and E. E. S. Jr., "Modified Booth Algorithm for High Radix Fixed-Point Multiplication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Man and Cybernetics, IEEE Transactions on*, vol. 1, pp. 164--167, June 1993.

[18] J. Penhollow, "A study of arithmetic recoding with applications to multiplication and division," *Dep. of Computer Sci., UNv. of Illinois, Urbana, Rep. 128*, Sept. 1962.

[19] B. Moller, "Improved Techniques for Fast Exponentiation," *In: ICISC 2002,LNCS*, vol. 2587, pp. 298-312, 2003.

[20] D. M. Gordon, "A Survey of Fast Exponentiation Methods," *J. Algorithms*, vol. 27, pp. 129-146, 1998.

[21] C. Heuberger and H. Prodinger, "THE HAMMING WEIGHT OF THE NON-ADJACENT-FORM UNDER VARIOUS INPUT STATISTICS," *Periodica Mathematica Hungarica*, vol. 55, pp. 81–96, 2007.

[22] J. A. Solinas, "Efficient Arithmetic on Koblitz Curves," *Designs, Codes and Cryptography*, vol. 19, pp. 195-249, 2000.

[23] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography* vol. 265. Cambridge: Cambridge University Press, 1999.

[24] N. Zhang, Z. Chen, and G. Xiao, "Efficient elliptic curve scalar multiplication algorithms resistant to power analysis," *Information Sciences*, pp. 2119-2129, 2007.

[25] G. Frieder and C. Luk, " Algorithms for Binary Coded Balanced and Ordinary Ternary Operations," *IEEE Transactions on Computers*, vol. 24 pp. 212-215, Feb. 1975.

[26] S. Maitra and A. Sinha, "A single digit triple base number system - a new concept for implementing high performance multiplier unit for DSP applications," in *Information, Communications & Signal Processing, 2007 6th International Conference on*, 2007, pp. 1-5.

[27] P. K. Mishra and V. Dimitrov, "Window-Based Elliptic Curve Scalar Multiplication using Double Base Number Representation," in *INDOCRYPT'07 Proceedings of the*

cryptology 8th international conference on Progress in cryptology 2007.

[28] A. Karatsuba, "The Complexity of Computation," *Proceedings of the Steklov Institute of Mathematics*, vol. 211, 1995.

[29] G. Harper, A. Menezes, and S. Vanstone, "Public-Key Cryptosystems with Very Small Key Lengths," *Proc. Advances in Cryptology ĐEUROCRYPT '92*, pp. 163-173, 1992.

[30] M. A. Hasan, "Look-up table-based large finite field multiplication in memory constrained cryptosystems," *IEEE Transactions on Computers*, vol. 49, pp. 749-758, Jul 2000.

[31] A. Mahboob and N. Ikram, "Lookup table based multiplication technique for GF(2(m)) with cryptographic significance," *IEE Proceedings-Communications*, vol. 152, pp. 965-974, Dec 2005.