Algorithms for Ternary Number System

Subrata Das\textsuperscript{a}, Joy Prakash Sain\textsuperscript{b}, Parthasarathi Dasgupta\textsuperscript{b}, Samar Sensarma\textsuperscript{c}

\textsuperscript{a}Dept. of Information Technology, Academy of Technology, Aedconagar, India  
\textsuperscript{b}MIS Group, Indian Institute of Management Calcutta, India  
\textsuperscript{c}Dept. of Computer Science & Engineering, University of Calcutta, India

Abstract

Numbers are counted in twos by machines, and in tens by men. However, there are countably infinite numbers of ways of counting numbers, in general. It has been shown that for optimum number of computations in counting a number, the base-\( e \) number system is the most preferred. Since the number 3 is the integer nearest to \( e \), we focus on a number system with base 3. In this paper, we discuss a few important existing algorithms and propose related novel algorithms for some fundamental computations.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of C3IT

Keywords: Ternary number, Trits, Arithmetic operations, Rotational symmetry

1. Introduction

Computation using the base-two number system is simple as the binary devices are of two states: On and Off. To the best of our knowledge, there are no electronic devices that have more than two states and are as reliable as a binary device. Decimal system for numbers uses ten symbols (0 to 9) and is a natural choice for counting. Representation of large numbers may be done using other unrelated symbols for each number. However, this is next to impossible, as it would require theoretically infinite numbers of symbols. The positional notations [8] have thus evolved, which allow us to represent any or all numbers distinctly with a finite set of symbols. If the cost associated with numeric representation is based on the number of digits to be counted, then it is preferable to use a number system with the largest base. For example, a number system having 10,000 as the base can represent any number between 0 and 9999 using only one digit. However, a problem with such a system is that we need 10,000 different symbols to represent the numbers in the system. On the other hand, for a unary number system, only one unique symbol may be used to represent any number. But in this system, the number of digits required to represent a number is equal to the number itself. Thus, a pertinent question would be: which base would be the best for optimal number of computations? If we minimize the base then the number of different symbols used to represent the number is increased and vice versa. Thus, in order to obtain the optimal
base [4], the product of base and width (number of digits used to represent a number) of the digit would be an objective function that needs to be minimized. Thus, for optimal number of computations, it may be noted that the product $b \times w$ needs to be minimized, where $b$ is the base and $w$ is the width in the digits and $b^w$ is constant. Using simple mathematics it can be easily shown that for optimal result, $b = e$. Thus, the optimal value of the base is $e$. For practical purposes, the integer 3 nearest to $e$ is considered as the optimal value of base width.

2. Literature review

The design and implementation of ternary circuitry were reported in [6]-[7]. Complete discussions on third base were reported in [4]. In that paper the author give the justification of the use of third base. A complete architecture, design and implementation of 2 bit ALU slice were discussed in [10]. A new type of transmission functions theory was reported in [11] and with that theory the author suggests that this theory can explain all the CMOS ternary circuits. Ternary mirror symmetrical number system is discussed in [12]. In [13] a mixed binary-ternary number system and its application in elliptic curve cryptosystem were discussed. Rotation Symmetric Boolean function has beckoned the interest of theoretician as well as practitioners in the field of cryptography [1, 2, 14]. In [3] the author generalizes the counting results about RSBF's to the rotation symmetric Boolean polynomial over GF (p) where p is prime.

3. Arithmetic operations in Base-3 number system

In ternary logic the term trits is used to denote ternary digits. Instead of using 0, 1 and 2, an alternative representation of the symbols may be as -1, 0 and 1 [5]. For simplicity, the symbol used for -1 is $\bar{1}$, hereafter, we shall use the notations -1 and $\bar{1}$ to refer to the same value. The ternary number system with this set of symbols is known as a balanced ternary system. Some of the interesting properties of a balanced ternary number system include [5]:

1. The negative number is obtained by interchanging 1 and $\bar{1}$.
2. The sign of a number is given by its most significant nonzero trit.
3. The operation of rounding off to the nearest integer is identical to truncation.

Some arithmetic operations in the base-3 number system are given below.

3.1. Addition in base-3 number system

Table 1 illustrates some examples of additions for the ternary number system. Each column corresponds to a pair of trits to be added and a carry trit. Thus, the total number of possible columns would be $3^3 = 27$.

<table>
<thead>
<tr>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>$\bar{1}$</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bar{1}$</td>
<td>$\bar{1}$</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\bar{1}$</td>
<td>$\bar{1}$</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\bar{1}$</td>
<td>0</td>
<td>1</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>1</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bar{1}$</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>1</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>0</td>
<td>$\bar{1}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Examples of Addition of trits

3.2. Subtraction in base-3 number system

The operation of Subtraction can be viewed simply as negation of a number followed by addition.
3.3. *Shift operation in base-3 number system*

3.3.1. *Arithmetic right shift operation.*

In a given number A having n trits, let \( R_{n-1} \) and \( R_0 \) respectively denote the most significant trit (MST) and least significant trit (LST). Since in case of balanced ternary system the most significant non-zero trit represents the sign, after arithmetic right shift the new MST becomes zero and the original LST is lost. If the LST of A is \( \overline{1} \), then the arithmetic right shift of A yields respectively \( \overline{\frac{1}{2}} \) and \( \frac{1}{2} \).

3.3.2. *Arithmetic left shift operation.*

In arithmetic left-shift operation of the number A an overflow flip-flap-flop \([5]\) can be used to store the MST, and the new LST is 0. Arithmetic left shift operation yields \( A \times 3 \).

4. *Hardware algorithm for multiplication and division of two base-3 numbers*

In this Section we describe multiplication and division of two ternary numbers.

4.1. *Multiplication Algorithm*

For multiplication we store multiplicand in a register \( BR \), say, and Multiplier in register \( QR \), say. Initially, we assume that product is zero. This is known as the partial product, where a partial product is obtained by multiplying the multiplicand with one trit of the multiplier. In simple multiplication, if the trit of the multiplier is 1(\( \overline{1} \)) then multiplicand is added (subtracted) with the partial product to generate a new partial product. Now the next trit of the multiplier is multiplied with multiplicand and the product is shifted by one trit to the left and added with the partial product to generate a new partial product. But in case of hardware multiplication (using registers), instead of shifting the (multiplicand \( \times c \)) (where \( c \) is a trit of the multiplier, having value 0 or 1 or \( \overline{1} \)) to the left we shift the partial product one trit to the right. We use the term \( Ashr \) to indicate arithmetic shift right. The multiplication algorithm is shown in Figure 1(a). This operation has been defined for trits in \([5]\). Example for multiplication is given in Figure 2.
4.2. Division Algorithm

In order to divide a number by another (both are non negative), we store the dividend in register Q and divisor in register M. During division we take a set of trits of dividend and if it has a value less than that of the divisor, then we have to take another trit of dividend and insert 0(zero) in the quotient. On the other hand, if the value of a set of trits of dividend is greater than or equal to the value of the divisor, then either 1 or 2 (i.e., $11$) is inserted in the quotient. For this we have to subtract the divisor from the set of trits of the dividend; if the result is negative we put 0 in the quotient and add divisor to the result to get back those set of trits of dividend. This is known as restoration of the dividend. If the result of subtraction is positive then quotient may be either 1 or 2 (i.e. $11$). For this, we put 1 in the quotient and again subtract divisor from the partial subtraction. If the result of subtraction is negative, then the result must be 1 and here we have to perform restoration operation. On the other hand, if the result of the subtraction is positive, then we put $11$ in the quotient to get 2 (i.e.$11$). The entire division operation is illustrated in the flow chart of Figure 1(b). Example for division is given in Figure 3.

Multiplicand $BR=\overline{1001}$ (i.e. 55 in decimal) and multiplier $QR=\overline{11011}$ (i.e. 56 in decimal). $\overline{BR}=\overline{11001}$

<table>
<thead>
<tr>
<th>$Q_0$</th>
<th>Operation</th>
<th>ER</th>
<th>AC</th>
<th>QR</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>0</td>
<td>00000</td>
<td>11011</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Final product = 001110011 (3080)₁₀

Figure 2: Example multiplication of two 3-base numbers

5. Complexity Comparison

5.1. Complexity comparison for multiplication algorithm

Let us consider two numbers a having n trits and b having m trits (n > m, i.e. a > b), both of base 3. Now, if a(b) is the multiplier, then in traditional repetitive addition algorithm we have to perform $O(3^n)$ ($O(3^m)$) number of additions. For the proposed algorithm for multiplication, the complexity is $O(n^2)$. For n-trit multiplier and multiplicand, there are $n$ numbers of iterations in the proposed multiplication algorithm. Each iteration involves a shift operation and a set of addition and subtraction. There are total $2n$ shift operations and each additions or subtraction requires time $O(1)$ time. Thus, the overall time complexity for multiplication is $O(2n^2) + O(n)$.

5.2. Complexity comparison for division algorithm

In case of repetitive subtraction algorithm numbers of subtractions $\geq 3^{n-m}$.

So in repetitive subtraction algorithm the time complexity is $O(3^{n-m})$. But with the algorithm described here time complexity for division is $O(2n^2) + O(n)$.

Here we divide 11(-1)01 by 00111 (i.e. in decimal we divide 100 by 13)
Operation | A | Q | SC
--- | --- | --- | ---
Initialization | 00 0 0 0 | 11-101 | 5
Left Shift(AQ) | 00 0 0 1 | 1-101[] | 
A=A-M | 00-1-1-1 | | 
A is -ve | 00-1-0 | | 
Set Q[0]=0 & A=A+M | 00 1 1 1 | 1-101[0] | 4
Size=Size-1 | 00 0 0 1 | | 
Left Shift(AQ) | 00 0 1 1 | -101[0][] | 
A=A-M | 00-1-1-1 | | 
A is -ve | 00-1 0 0 | | 
Set Q[0]=0 & A=A+M | 00 1 1 1 | -101[0][0] | 3
Size=Size-1 | 00 0 1 1 | | 
Left Shift(AQ) | 00 1 1 1 | 01[0][0][] | 
A=A-M | 00-1-1-1 | | 
A is -ve | 00 0-1 1 | | 
Set Q[0]=0 & A=A+M | 00 1 1 1 | 01[0][0][0] | 2
Size=Size-1 | 00 1 1-1 | | 
Left Shift(AQ) | 01 1-1 0 | 1[0][0][0][] | 
A=A-M | 00-1-1-1 | | 
A is -ve | 01-11-1 | | 
Set Q[0]=1 & A=A-M | 00-1-1-1 | 1[0][0][0][1] | 1
A is +ve & Set Q[0]=1 & Size=Size-1 | 00 1-1 1 | 1[0][0][0][1][1-1] | 
Left Shift(AQ) | 01 1 1 1 | [0][0][0][1][1-1][] | 
A=A-M | 00-1-1-1 | | 
A is +ve | 00 1 0 0 | | 
Set Q[0]=1 & A=A-M | 00-1-1-1 | [0][0][0][1][1][1] | 
A is -ve | 00 0-1 1 | | 
A=A+M | 00 1 1 1 | | 
Size=Size-1 | 00 1 0 0 | | 

Final Remainder=(001003)=(9)\text{10} , Quotient=(0001(-1)1)\text{3}=(7)\text{10}

Figure 3: Example division of two 3-base numbers

6. Rotation Symmetric Boolean Function in GF(3)

Rotation Symmetric Boolean Functions (RSBF) have huge application in cryptosystem. A Boolean function is symmetric if it is invariant under any permutation of its variables [8]. A Boolean function f(.) of n variables is rotation symmetric if and only if f(x_{n-1}, x_{n-2}, \ldots, x_0) = f((x_{n-2}, \ldots, x_0, x_{n-1}) = \ldots = f(x_0, x_{n-1}, \ldots, x_1). It is known that, for n-variable RSBF functions, the associated set of input trit strings can be divided into a number of subsets (called partitions), where every element of a subset can be obtained by simply rotating the string of trits of some other element of the same set. Now RSBFs are a class of Boolean functions which have good combination of nonlinearity, correlation immunity, balancedness and algebraic degree [9]. A Boolean function is applicable to cryptography if it has the above mentioned properties. Now since base three is optimal, it is quite expected that the Boolean function in GF(3) is better. In this paper, we propose an algorithm to generate the partitions of RSBFs where a partition is a set of a trits string and the rotations of this string, such that the output of each of these strings as input provides the same output. Formula for generating the partitions for RSBF in any base

$$g_{n,p} = \frac{1}{n} \sum_{i/n} \phi(i)p^j \ [3].$$ Table 2 shows the partitions generated for n = 4. The proposed algorithm for the generation of partitions of RSBFs in GF(3) is given below. This algorithm generates the starting string of each partition and total numbers of partitions. We use the symbols 0, 1, 2 to represents the numbers
instead of using \(0,1, \bar{1}\). Table 3 shows the number of partitions for different values of \(n\) (number of trits).

A formal description of the proposed algorithm is given in Figure 4.

**Definition 1.** If a Boolean function \(f(x_{n-1}, x_{n-2}, \ldots, x_0)\) exhibits rotation symmetry, then the period over which its exhibits this property is defined to be the cycle length for the function.

![Table 2: Partitions for RSBF for \(n=4\)](image)

<table>
<thead>
<tr>
<th>Partitions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{(0000)}</td>
<td>partition0</td>
</tr>
<tr>
<td>{(0001), (0010), (0100), (1000)}</td>
<td>partition1</td>
</tr>
<tr>
<td>{(0002), (0020), (0200), (2000)}</td>
<td>partition2</td>
</tr>
<tr>
<td>{(0011), (0110), (1101), (1011)}</td>
<td>partition3</td>
</tr>
<tr>
<td>{(0012), (0120), (1200), (2001)}</td>
<td>partition4</td>
</tr>
<tr>
<td>{(0021), (0210), (2100), (1002)}</td>
<td>partition5</td>
</tr>
<tr>
<td>{(0022), (0220), (2200), (2002)}</td>
<td>partition6</td>
</tr>
<tr>
<td>{(0101), (1010)}</td>
<td>partition7</td>
</tr>
<tr>
<td>{(0102), (2010), (0210), (1020)}</td>
<td>partition8</td>
</tr>
<tr>
<td>{(0111), (1110), (1101), (1011)}</td>
<td>partition9</td>
</tr>
<tr>
<td>{(0112), (1120), (1201), (2011)}</td>
<td>partition10</td>
</tr>
<tr>
<td>{(0121), (1210), (2101), (1012)}</td>
<td>partition11</td>
</tr>
<tr>
<td>{(0122), (1220), (2201), (2102)}</td>
<td>partition12</td>
</tr>
<tr>
<td>{(0202), (2020)}</td>
<td>partition13</td>
</tr>
<tr>
<td>{(0211), (2110), (1102), (1021)}</td>
<td>partition14</td>
</tr>
<tr>
<td>{(0212), (2120), (1202), (2021)}</td>
<td>partition15</td>
</tr>
<tr>
<td>{(0221), (2210), (2102), (1202)}</td>
<td>partition16</td>
</tr>
<tr>
<td>{(0222), (2220), (2202), (2202)}</td>
<td>partition17</td>
</tr>
<tr>
<td>{(1111)}</td>
<td>partition18</td>
</tr>
<tr>
<td>{(1112), (1121), (1211), (2111)}</td>
<td>partition19</td>
</tr>
<tr>
<td>{(1122), (1221), (2211), (2112)}</td>
<td>partition20</td>
</tr>
<tr>
<td>{(1212), (2121)}</td>
<td>partition21</td>
</tr>
<tr>
<td>{(1222), (2221), (2212), (2122)}</td>
<td>partition22</td>
</tr>
<tr>
<td>{(2222)}</td>
<td>partition23</td>
</tr>
</tbody>
</table>

**Algorithm 1: Algorithm genpart()**

Data structures: Counter = Number of partitions, Answer=Starting string of partition  
Input: Number of trits  
Output: Starting string of every partition and total number of partitions

1. Initialization: Counter=0;  
2. Answer[0] = \(0^n\), (* \(a^n\) means a string of \(n\) trits *)  
3. Counter=counter +1;  
4. Consider the trit-string \(s\) corresponding to the next number store its decimal form in Answer[].  
5. while trit-string corresponding to \(s = (2^n)\) do  
6. while the starting string of any partition is less than of any element of that particular orbit then go to step 8.  
7. Take the trit-string corresponding to \(s\) as the starting string of the next partition  
8. Take the trit-string corresponding to next number  
9. endwhile  
10. end

Figure 4: Generation of Partitions of Rotation Symmetric Boolean Function in GF (3)
Correctness of the proposed algorithm: As the successive numbers are being considered, thus if the number of rotations of any number is less than the number of trits in that number, then the number must have appeared before in some element of some partition.

Time complexity of the algorithm: The time complexity of the above algorithm is \(O(3^n)\). Since the formula for number of partitions is exponential, the time complexity of the above algorithm is inherently exponential.

7. Conclusion

In this paper, we discuss some existing algorithms for arithmetic operations, and propose few novel related algorithms. An algorithm for Rotation Symmetric Boolean Functions is also proposed. To the best of our knowledge, this is the first algorithmic approach in RS BFs in base-3. In terms of computational complexity for arithmetic operations, the proposed and existing algorithms for base-3 number systems are observed to be better than those for the traditional base-2 number system. Encouraged by these initial observations, we would like to extend the work to other application domains as well.

References