INDIAN INSTITUTE OF MANAGEMENT CALCUTTA

WORKING PAPER SERIES

WPS No. 714/ September 2012

Arithmetic Algorithms for Ternary Number System

by

Subrata Das
Assistant Professor, Department of Information Technology,
Academy of Technology, West Bengal

Parthasarathi Dasgupta
Professor, IIM Calcutta, Diamond Harbour Road, Joka, Kolkata 700104, India

&

Samar Sensarma
Professor, Department of Computer Science & Engineering University of Calcutta

Arithmetic Algorithms for Ternary Number System

Subrata Das, Parthasarathi Dasgupta and Samar Sensarma

Abstract—The use of multi-valued logic in VLSI circuits ... system. Some new algorithms for arithmetic ... for Rotation Symmetric Boolean Function in base-3 is also proposed in this paper.

Index Terms—Ternary number, trit, arithmetic, 3-valued logic,VLSI.

I. INTRODUCTION

number of ways to represe

i.e. ternary number

g a number.Ternary

nal binary logic. The

formation Technology ,
ndia.

IS group,Indian Institute

Science and Engineering

22]

some of the related recent works. Section III introduces some of the basic terminologies to be used in subsequent discussion. Section IV discusses the applications of ternary logic in computer science specially in the field of $L\ I$, $Co\ puter\ Architecture$ and $Codding\ theory$. Section V discusses different arithmetic algorithms for ternary number system. Section VI discusses hardware implementation of multiplication and division algorithms . Section VII analyzes the performances of multiplication and division algorithms. Section VIII discusses a special of *Boolean Function* known as *Rotation Symmetric Boolean Function* and different algorithms for generating the same. Finally, Section IX concludes the chapter and briefly states the future scopes of work.

II. LITERATURE REVIEW

A detailed review on third base number systems and the justification of the use of third base are reported in [1]. Several advantages of using ternary logic(multi-valued logic) over the traditional binary logic appear in [18]. A survey on the development of the algebras and techniques for the realization of three valued function are reported in

a	b	Y_0^{NOR}	Y_1^{NOR}	Y_2^{NOR}	Y_0^{NAND}	Y_1^{NAND}	Y_2^{NAND}
0	0	2	2	2	2	2	2
0	1	0	1	2	2	2	2
0	2	0	0	0	2	2	2
1	0	0	1	2	2	2	2
1	1	0	1	2	0	1	2
1	2	1	2	1	0	1	0
2	0	0	0	0	2	2	2
2	1	0	0	0	0	1	2
2	2	0	0	0	0	0	0

TABLE VI
TRUTH TABLE FOR TERNARY NOR AND **NAND** GATES

D. Ternary Full Adder

The following Table VII is the truth table of full adder. The expression for M is $A \bigoplus B \bigoplus C$.

a	b	cin	cout	sum
0	0	0	0	0
0	0	1	0	1
0	0	2	0	2
0	1	0	0	1
0	1	1	0	2
0	1	2	1	0
0	2	0	0	2
0	2	1	1	0
0	2	2	1	1
1	0	0	0	1
1	0	1	0	2
1	0	2	1	0
1	1	0	0	2
1	1	1	1	0
1	1	2	1	1
1	2	0	1	0
1	2	1	1	1
1	2	2	1	2
2	0	0	0	2
2	0	1	1	0
2	0	2	1	1
2	1	0	1	0
2	1	1	1	1
2	1	2	1	2
2	2	0	1	1
2	2	1	1	2
2	2	2	2	0

TABLE VII
TERNARY FULL ADDER

The expression for carry is $\overline{s} \wedge ((A \wedge B \wedge C) \vee (A \wedge B \wedge \overline{C})) \vee (A \wedge$

symbols s in terms of another system of symbols then the main problem of representation is the following

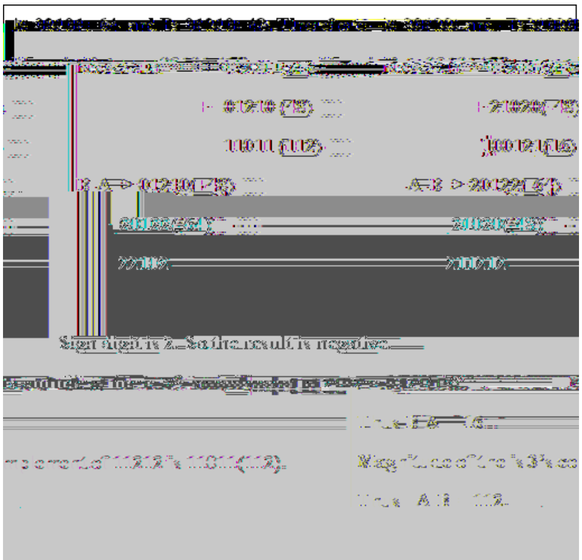
1) How to represent the source symbols so that their representation if far apart in some suitable sense. As a result in spite of small changes(noise),the altered symbols can be discovered to be wrong and even possibly corrected.
2) How to represent the source symbols in a minimal form for purposes of efficiency. The average code length, $L=\sum_{=1} p$ is minimized where is the length of the representation of the i symbol s.

In some early days one variable length ternary code was popularly used for communication known as Morse code.Three different symbols of this code are dash(-),dot(.) and space( ).The length of the high frequency alphabet such as "E" is small and that of low frequency alphabet such as "J" is long. As a result the average length of the code is reduced.[6]

V. Arithmetic Operation on T

D. Addition and subtraction of two conventional ternary numbers

For addition of conventional ternary numbers we have to use the truth table for full adder as shown in Table VII.For substation(A-B) we have to take 3's complement of B and add it to A.3's complement of a number can be easily obtained by interchanging 0 and 2 followed by add 1 to it.
The Figure 1 shows few examples of addition and subtraction of two conventional ternary numbers.

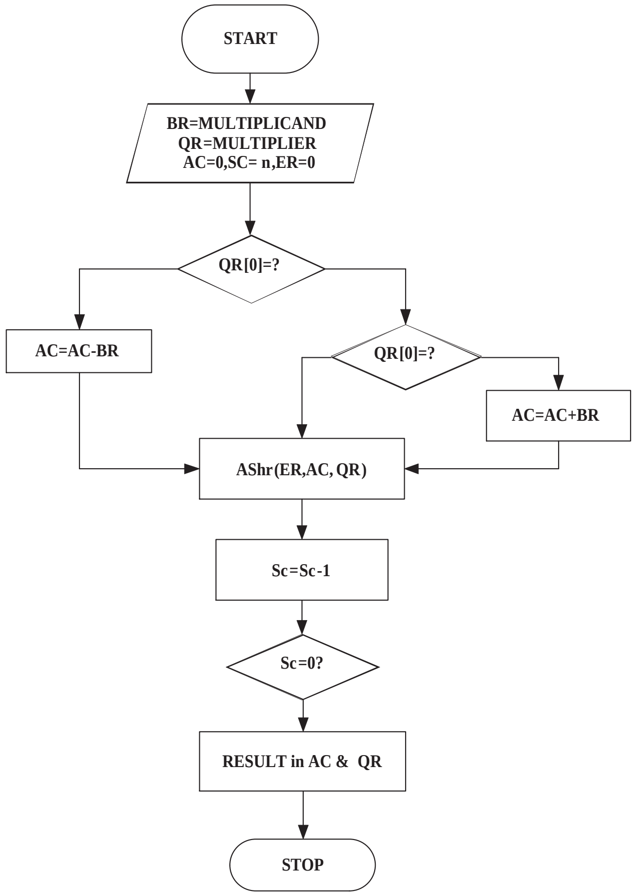


BR=011110, QR=022112, -BR=211120

QR[0]QR[-1]	Operation	AC	QR	QR[-1]	SC
	Initialization	000000	022112	0	6
20	AC=AC-BR AShr and Sc=Sc-1	211120 211120 221112	 002211	 2	 5
12	AC=AC+BR AC=AC+BR Ashr and Sc=Sc-1	011110 [1]002222 011110 021102 002110	 200221	 1	 4
11	AC=AC+BR Shift right Sc=Sc-1	011110 020220 002022	 020022	 1	 3
21	AC=AC-BR Ashr & Sc=Sc-1	211120 220212 222021	 202002	 2	 2
22	Ashr	222202	120200	2	1
02	AC=AC+BR Ashr & Sc=Sc-1	011110 [1]011012 001101	 212020	 0	 0

Final product=001101212020

Fig. 3. Example of multiplication of two conventional ternary numbers



B. Multiplication Algorithm using balanced ternary numbers

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 bit of the multiplier is 1 then multiplicand is added with the partial product to generate a new partial product. Now the next bit 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 $\mu u\ tip\ icand \times c$ (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. This operation has been defined for trits in [2].The entire operation is shown in Figure 4.

Lemma 1. *If a and b are two ternary numbers such that a is minimum and b is maximum then b $3 \times a$.*

Proof: Let a=10 0=3^{n-1} and b=222 2=$2\times\sum_{=0}^{n-1} 3$. Now $\frac{}{}=\frac{2\times\sum_{=0}^{n-1} 3}{3^{n-1}}=2+\frac{2\times\sum_{=0}^{n-2} 3}{3^{n-1}}=2+\frac{(\sum_{=1}^{n-1} 3 -1)}{3^{n-1}}$.
Now 1 $\frac{\sum_{=1}^{n-1} 3}{3^{n-1}}$ 2. $\therefore$ $\frac{}{}$ 3. $\therefore b$ $3 \times a$. ∎

C. Division Algorithm using conventional ternary number system

In order to divide a number by another, we store the dividend in register Q and divisor in register M. During

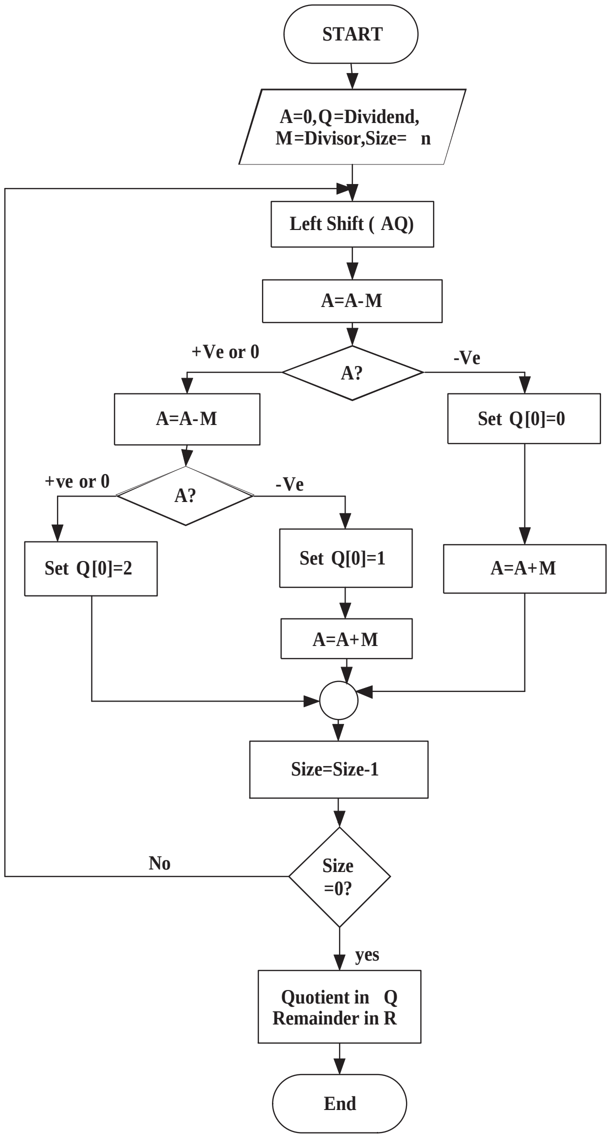
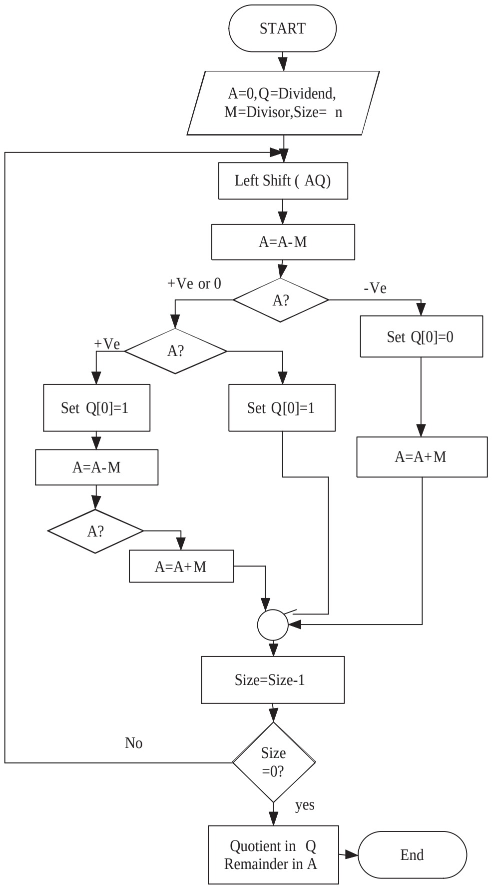


Fig. 5. Flowcharts for Division Algorithm for two non negative numbers using conventional ternary number system

D. Division Algorithm using balanced ternary number system

The division of two nonnegative ternary numbers is discussed in[21] and the flow chart for that algorithm is shown in Figure 7.Here we describe the algorithm when the dividend is negative.In this case instead of subtracting the divisor from the set of trits of dividend is added with


START
A=0,Q=Dividend,
M=Divisor,Size= n
Left Shift (AQ)
A=A-M
+Ve or 0
A?
-Ve
+Ve
A?
Set Q[0]=0
Set Q[0]=1
Set Q[0]=1
A=A-M
A=A+M
A?
A=A+M
Size=Size-1
No
Size
=0?
yes
Quotient in Q
Remainder in A
End

we propose an algorithm to generate the partitions of rotation symmetric Boolean functions where a *partition* is a set of a trit string and the rotations of this string, such that the output of each of these strings as input provides the same output.Generation of these functions are known to be combinatorially explosive. It is known that, for n-variable *RSBF* functions, the associated set of input bit 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 bits of some other element of the same set.Formula for generating the partitions for Rotation Symmetric Boolean Function in any base $g_n = \frac{1}{n}\sum_{\,|n} (t)p^{\frac{n}{}}$[5]. Figure 10 shows the partitions generated for n = 4.

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

{(0000)}				partition 0
{(0001)	(0010)	(0100)	(1000)}	partition 1
{(0002)	(0020)	(0200)	(2000)}	partition 2
{(0011)	(0110)	(1101)	(1011)}	partition 3
{(0012)	(0120)	(1200)	(2001)}	partition 4
{(0021)	(0210)	(2100)	(1002)}	partition 5
{(0022)	(0220)	(2200)	2002}	partition 6
{(0101)	(1010)}			partition 7
{(0102)	(2010)	(0201)	(1020)}	partition 8
{(0111)	(1110)	(1101)	(1011)}	partition 9
{(0112)	(1120)	(1201)	(2011)}	partition 10
{				

Algorithm genpartC()

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

1. Initialization: Counter=0,number1=$0^{n-1}1^1$ and number2=$0^{n-1}2^1$;
2. A [0] = 0^{n}; (* ^{n} means a string of trits*)
3. Counter=counter +1;
4. while trit-string corresponding to = $\{1^1 2^{n-1}\}$ do
5. number1=number1+3 and number2=number2+3
6. While the starting of any partition(i.e. number1 and number2) is less than any element of the particular orbit then goto step 8
7. If the next number becomes $\{1^1 0^{n-1}\}$ then number1=$\{1^1 0^1 2^{n-2} + 1\}$ and number2=$\{1^1 0^1 2^{n-2} + 2\}$
8. Take the trit-string corresponding to number1=number1+3 and number2=number2+3
9. Answer[counter++]=number1 and Answer[counter++]=number2
10. Endwhile
11. counter=counter+1
12. Answer[counter]=$\{2^{n}\}$