Volume 3, No. 5, May 2012
Journal of Global Research in Computer Science
ISSN-2229-371X
RESEARCH PAPER
Available Online at www.jgrcs.info

ZOUST FOR SUPREMECY-BOOTH'S & BKS MULTIPLICATION ALGORITHM

Barun Biswas*[1], Krishnendu Basuli[2] and Samar Sen Sarma[3]
*Dept. of Computer Science, West Bengal State University, India
barunbiswas9u6@gmail.com
[2]Dept. of Computer Science, West Bengal State University, India
krishnendu.basuli@gmail.com
[3]Department of Computer Science and Engineering,
University of Calcutta, 92, A.P.C. Road,
Kolkata – 700 009, India.
sssarma2001@yahoo.com

Abstract – The aroma of multiplication algorithm is ever lasting. It is due to time, space and cost effective as well as number representation. We study here that given a multiplier whether we will take one bit, two bit, three bit or in general n bit for multiplicative realization. Since initial inspiration is Booth's algorithm[1] we consider an interesting aberration of Booth's multiplication algorithm in this paper called BKS(named after the author Barun Biswas, Krishnendu Basuli & Samar Sen Sarma). The BKS algorithm surpasses Booth's algorithm with minimum cost.

Keywords- Binary multiplication, Arithmetic algorithmic complexity, trade of among relevant parameters, complements number representation.

INTRODUCTION

Multiplication is one of the most important arithmetic operations. In our childhood we learn to multiply two number by times table process and using our figures. After that when we grew up and enter the world of computer science we learnt about different process of multiplication. Many new algorithms of multiplication is introduced to. Now a day many new process of multiplication has been proposed. Using these process of multiplication many high speed of processors has been designed. Even we saw that many other arithmetic processes are performed by using repeated multiplication. Here is the point that if we can introduce a new process of high speed multiplication then accordingly using that process all the field where multiplication can be used we can gain speed, save time and also cost can be reduced. During the design of the proposed multiplication process we try to maintain flexibility of the algorithm, the unnecessary operation that can be reduced is reduced.

As we compared the proposed the process with Booth's here we mention that in Booth's process of multiplication [1] of the maximum number of partial product is N, whereas in our proposed process the maximum number of partial product is N/2. So it can easily be understand how much efficient this process is compared to Booth's multiplication. In most of the example of multiplying two 2's complement binary number we notice that in Booth's process there is a carry generation and carry propagation time which increase the time of execution time for a large number of multiplication.

HISTORY OF ALGORITHM: [3]

Word Origin[8]:

It's hard to see how, but the word "algorithm" came from the name of Abu Abdullah Muhammad ibn Musa Al-Khwariszmi, a ninth century, Persian mathematician. "Algorism" originally referred to arithmetical notation using Hindu-Arabic numerals. By the 18th century, "algorism" evolved into "algorithm" via the Latin translation of Al-Khwarizmi's name, reports Scriptol.com.

Mathematical Functions:

Al-Khwarizmi designed basic, precise rules for adding, multiplying and dividing numbers. These efficient, mechanical rule sets could extract square roots and calculate digits of pi, S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani point out in their book, "Algorithms."[7]

Benefits to Civilization:

Since Al-Khwarizmi's time, algorithms have played a progressive role in Western civilization, to forward science and technology, as well as benefit industry and commerce. As computers and data processing developed, algorithms began to embody the positional system in its bits, words and arithmetic unit, say Dasgupta, Papadimitriou and Vazirani.[7]

Today:

The design of algorithms stands as a centerpiece for computer science and information systems programmers. Energy exploration, financial management, entertainment, musical composition, and medicine all benefit from algorithmic procedures, according to the Music Algorithms website

WHAT DOES MULTIPLICATION MEAN?

If we minutely analyze different types of multiplication process then we notice that multiplication is nothing but repeated addition to the previous partial product performed on the multiplicand depending on the multiplier. Suppose we want to multiply two numbers M(multiplicand) and Q(multiplier), then we have to add M Q number of times. Here it gives the worst case complexity as the maximum number of operation is performed to per formed the multiplication operation. As a result different algorithm designers tried to reduce the number the partial product to reduce the complexity of the process.

PREVIOUS WORKED ON MULTIPLICATION

Before discussing the proposed process of multiplication let us take a look to those previous works that are available at present. To make our presentation clear it is necessary to discuss them.

Repeated addition multiplication: [4]:

Among the available multiplication algorithms the most simple one is repeated addition multiplication. It has the complexity of O(n) all the time. The logic behind this process is very simple, simply decrease the multiplier by 1 and add the multiplicand to the accumulator and repeat the process until the multiplier become 0(zero). It is clear that the process will run number of times equals to the multiplier.

Shift and add [4]:

The next process of multiplication is "shift and add". This process is performed on the binary number. Here the LSB of multiplier is checked, if it is 1 then multiplicand is added otherwise the previous partial product is shift one bit to the right. This process is continued until the MSB of the multiplier is checked. This is a sequential multiplication scheme with worst case time delay proportional to the numebr of bits O(n). So in this process we notice that the pertial product of the process is less tahn the previous process. But in this process if all the multiplier bit is 1 jthen the total number of pertial product is equal to the number of bit present in the multiplier.

Array multiplier [5]:

The next topic is to be discussed is Array multiplier. Checking the bits of the multiplier one at time and forming the pertial product is a shift operation that requirs a sequence of add and shift microperation. The multiplication of two binary numbers can be done with one micro operation by means of a combinational circuit that forms the product bits all at once.
Let us illustrate the process with an example, let there are two bit multiplicand b_0, b_1 and two bits multiplier $a_0 a_1$. Now the product is $c_3 c_2 c_1 c_0$.
The operation is as follows

$$\begin{array}{cccc} & b_1 & & b_0 \\ & a_1 & & a_0 \\ \hline & a_0 b_1 & & a_0 b_0 \\ & a_1 b_1 & a_1 b_0 & \\ \hline c_3 & c_2 & c_1 & c_0 \end{array}$$

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in multiplier. Binary out put in each level of AND gate is added in parallel with partial product of the product. For j multiplier bits and k multiplicand bits wee need j*k AND gates and (j-1)*k bits adders to produce a product of j+k bits.

As our process is based on the concept of the Booth's multiplication we shall discuss the Booth's process next .

Divide and conquer [2]:

In traditional process of multiplication we see that the process of multiplication of two n bit binary number require O(n^2) digit operation . but in divide-n-conquer technique the multiplication requires O(n^{log3}) or O($n^{1.59}$) (approximately) bit operation.
In this process binary number is divided into two parts: say the number is X, so the number can be represented as X= a + b. So in this process we see that there may be four n/2 bit operation but the product (say X*Y) can be represented as three n/2 bit multiplication. So in this process the complexity can be represented as O(n^{log3}) = O($n^{1.59}$).

Booth's Multiplication[1]:

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1951 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed. Booth's algorithm is of interest in the study of computer architecture

Booth's algorithm involves repeatedly adding one of two predetermined values Multiplicand and Multiplier to a product, then performing a rightward arithmetic shift on product. Let **M** and **Q** be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in **m** and **r**. Then the number of bit in the product is equal to the (x+y+1).

The Booth's multiplication process is based on the four basic steps. In this process the two multiplier bit is checked and depending on their combination (2^2=4) four different steps is performed. Now let us discuss the Booth's multiplication by taking an example.

```
  Let      M(multiplicand)           =0000000000001101
           Q(multiplier)             =0000000001010101
 M   =0000000000001101
 Q   =0000000001010101
   ____________________________________________________
  =1111111111110011                  // multiplier bit is 10
 =000000000001101*                   // multiplier bit is
  =11111111110011**                  // multiplier bit is 10
  =0000000001101***                  // multiplier bit is 01
  =111111110011****                  // multiplier bit is 10
  =00000001101*****                  // multiplier bit is 01
  =1111110011******                  // multiplier bit is 10
  =000001101*******                  // multiplier bit is 01
  __________________________________________________
  =0000010001010001
```

In the above example we notice that there are three operations performed, namely shift, addition, complement.
The number of operation is as follows
Shift = 7
Addition= 7
Complement= 4

In this case the complexity of the Booth's process is maximum.

PROPOSED PROCESS OF MULTIPLICATION: BKS

In the following passage we shall discuss about our proposed process of multiplication. The logic that we follow will be discussed in brief.

Before discussing the logic behind the proposed process of the multiplication algorithm we shall discuss on some points based on which we proceed. We know that when there is a n number of 1's sequentially present in a number then that part can be represented (of course in binary number system) 2^n- 2^0.

For example let us take a binary number 111111=63, i.e. in this binary number there is six 1's. So according to the rule this number can be represented as
$2^6 - 2^0$=1000000-1=111111 (64-1=63).

Therefore if we want to multiply some number by a number of sequences of 1's then the process can be as follows:

Say multiplicand=M and the multiplier is a number of n number of 1's then the product will be

M*(n number of 1's)
=M*($2^n - 2^0$)
=$2^n * M - 2^0 * M$
=$2^n * M - M$ [2^0=1]
=$2^n * M$ +M` [M` is 2's complements representation of –M]

Now $2^n * M$ can be obtained if we perform left shift on the number M n times. Therefore the conclusion is that add the complement of multiplicand M to $2^n * M$.
In our discussion of multiplication of two 2's complement presentation of number we shall use this process whenever we find a sequence of 1's in the multiplier.

BKS Algorithm:

The BKS process of multiplication is performed by considering two bits of multiplier. We check two bits and shift two places to the right except when it is last check. Based on the value of these two bits we shall perform four operations. They are as follows:

a. If the multiplier bits are 00 then do nothing and check the next bit
   a) If it is 0 then shift one place to right.
   b) If it is 1 then add multiplicand and shift one place to right.
b. If the multiplier bits are 01 then add multiplicand.
c. If the multiplier bits are 10 then add 2*multiplicand.
d. If the multiplier bits are 11 then subtract the multiplicand and check the next bit
   a) If it is 1 then do nothing and shift one place to the right.
   b) If it is 0 then add the multiplicand

Flow chart

The flow chart for the BKS algorithm is as follows:

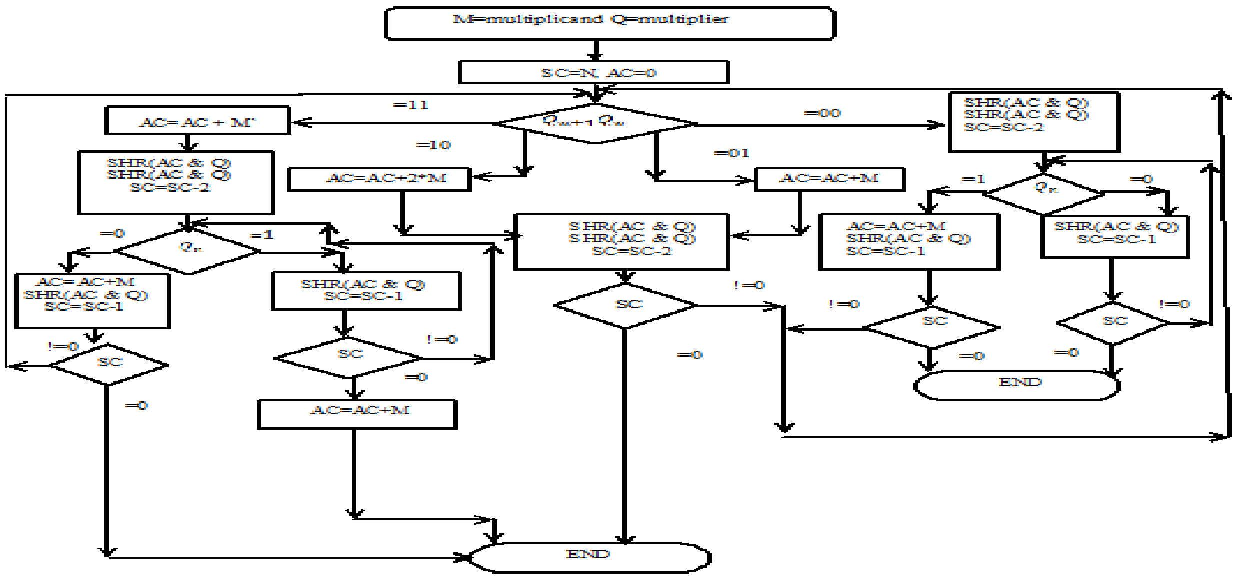

Figure 1: Flow chart for the BKS algorithm

So far we have seen the algorithmic form and flow chart of the proposed algorithm for multiplication. Now let us take an example and analysis the procedure. To compare with the Booth's algorithm let us take the same example discussed before on Booth's process.

Let M represent the multiplicand and Q represent the multiplier. And the M and Q are 0000000000001101 and 0000000001010101respectively. Based on the proposed process let us compute the product.

M(multiplicand) =0000000000001101
Q(multiplier) =0000000001010101

M =0000000000001101
Q =0000000001010101

=0000000000001101	// multiplier bits are 01
=00000000001101**	// multiplier bits are 01
=000000001101****	// multiplier bits are 01
=0000001101******	// MSB are 01

=0000010001010001

In the above process of multiplication we notice that the first two multiplier bits are 01 so according to our rule just add M and then shift partial product two place right and check the next two bit of the multiplier again it is 01 so do the same operation again. Continue this process until MSB is reached. And at the end of the process we notice that the number of shift, complement and addition operation are as
Shift = 6
Complement= 0
Addition= 3

So in this example we can see that we need to do 1 shift, 4 complement and 4 addition operations less. So we see that in worst case of Booth's multiplication we are gainer.

Now let us discuss some general case where we can prove that our process is advantageous than Booth's algorithm. For the two bits comparison there may be four possibilities 00, 01, 10, 11. For these possibilities the different operation is discussed. We also see that for n number of bits there is always (n+1) shift operation where as in our proposed process there may be less than n or equal to n or in very few case there may be greater than n shift operation. And as the multiplier bit is shifted one place at a time there is a possibility of occurring complement operation several times. The most important case is to notice is that in our proposed algorithm as we check and shift two bits at a time so there is a possibility of occurring maximum n/2 partial product, where as in Booth's process it is n times. In case of Booth's process if there are 01111…..1110 or 10000……00001 the operation is same one addition and one subtraction operation to be performed. But in our case when the bits are 1000000….000001 then two addition is to be performed (the complexity of subtraction operation is more than addition) and if the bits are 011111….11111110 then the case is same as booth or modified booth; i.e. one addition and one subtraction.

Now one of the most important topic still to be discussed is that the worst case complexity of our process. For our proposed BKS process the worst case complexity comes if the bit pattern of multiplier is ………..11011011011011011. in this case we can see the for the bit pattern 011 the operation need is 1 complement, 2 shift and 1 addition. For the same bit pattern in booth process the operations are 1 complement, 2 shift and 1 addition operation.

Let us discuss this by an example let

M =10101011010

Q =11011011011

For the above multiplicand and multiplier the operation required in our proposed process is 4 complement operations, 11 shift operations, and 7 addition operations, which is same as Booth's process. So we can say that even in our worst case the complexity of our proposed process of multiplication does not exceed that of Booth's process.

RESULT

We notice that to perform the multiplication process of both Booth's process and our proposed BKS process only three basics operations are needed namely shift, complement and addition. We perform our proposed process of multiplication and Booth's process of multiplication on 500 randomly generated data, and we got the result that our process is advantageous over Booth's process is
Shift 4.88%
Complement 31.64%
Addition 18.52%

These results are purely an average case advantageous of our proposed BKS process over Booth's process.

The graph representation of the result (shift, complement and addition operation) performed on 50 data is shown below. Here the randomly generated number (in binary form) is given and also the number of operation required.

Table: 1

BINARY NUMBER	BKS process			Booth's process		
	Shift	comp	add	shift	comp	add
00000000000000000000000010000010	4	0	4	8	2	3
00000000000000000000010001000010	7	0	3	11	3	5
00000000000000000010110110001000	14	1	5	14	4	7
00000000000000000100010010111011	12	2	7	15	6	9
00000000000000000001100100001111	14	1	4	13	4	5
00000000000000000010001100101100	13	0	6	14	4	7
00000000000000000011100011011110	13	3	8	14	3	5
00000000000000000000110111110011	11	3	7	12	4	5
00000000000000000100100000111100	11	1	5	15	3	5
00000000000000000000010101010000	8	0	4	11	4	7
00000000000000000001010100100100	10	0	5	13	5	9
00000000000000000101011110111111	15	2	5	15	5	7
00000000000000000110100111101111	17	2	6	15	5	7
00000000000000000001110000010110	16	2	6	15	5	7
00000000000000000111100111101110	14	1	6	13	3	5
00000000000000000011100110111011	14	3	8	15	3	5
00000000000000000100110101101000	13	4	9	14	5	7
00000000000000000101010010110001	14	1	7	15	5	9
00000000000000000100110011000011	15	0	7	15	7	11
00000000000000000100011001001011	14	1	7	15	5	7
00000000000000000101110000101001	13	1	7	15	6	9
00000000000000000000111010010110	15	1	6	15	6	9
00000000000000000000111110001011	13	1	7	12	4	7
00000000000000000100100000111001	12	2	6	12	4	5
00000000000000000010110000011000	13	3	7	14	3	5
00000000000000000110111100110010	12	1	6	15	5	7
00000000000000000010000111011000	15	2	8	15	4	7
00000000000000000001011100000100	11	1	5	14	3	5
00000000000000000011011000100111	12	1	4	13	3	5
00000000000000000101100011101110	14	2	7	14	5	7
00000000000000000001010011110100	15	2	7	15	4	7
00000000000000000100010011100111	10	1	6	13	4	7
00000000000000000101001010011111	11	2	7	15	5	7
00000000000000000000000001100100	12	1	6	15	6	9
00000000000000000110010011000011	13	2	7	15	5	7
00000000000000000011000101111010	8	0	2	7	2	3
00000000000000000011000000111011	11	1	6	14	4	7
00000000000000000100101001100010	14	1	5	14	3	5
00000000000000000100110100111110	12	2	6	15	5	9
00000000000000000010001111010001	13	1	7	15	4	7
00000000000000000001010111010101	11	1	6	14	5	7
00000000000000000110011100001111	16	1	6	13	7	11
00000000000000000010010111000010	16	2	6	15	4	5
00000000000000000010101010111101	12	1	6	14	4	7
00000000000000000011010001001110	17	1	6	14	7	11
00000000000000000001111011000110	15	1	6	14	4	7
00000000000000000011010011111101	14	1	6	13	3	5
00000000000000000000011011110011	16	1	5	14	5	7
00000000000000000011111100111110	10	3	7	11	4	5
00000000000000000111110111101001	13	2	6	14	2	3

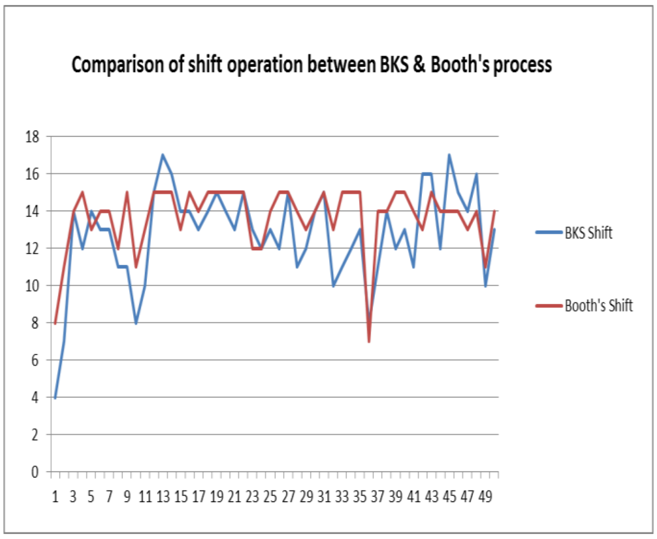

Figure: 2

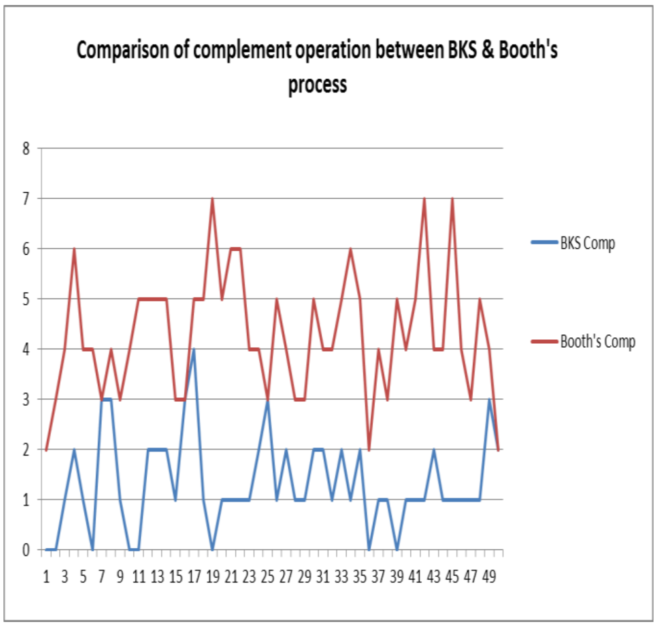

Figure: 3

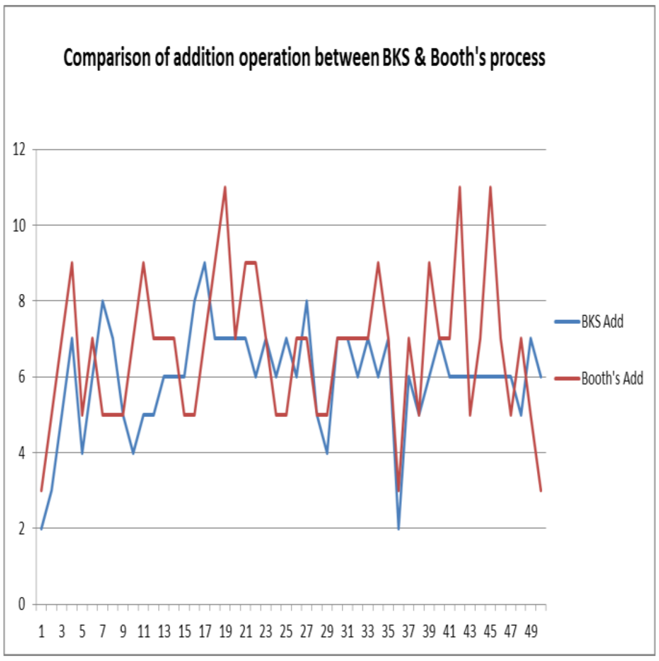

Figure: 4

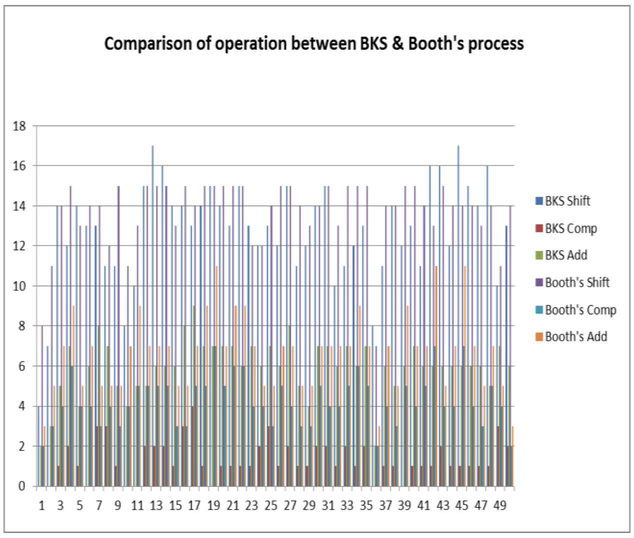

Figure: 5

The configuration of the computer we used to execute the program is Intel® Pentium® Dual CPU E2160 @1.80GHz RAM 1 GB, Operating system Windows 7, and the software used is Turbo C++ 4.5. The result on the 500 data for Booth's process is that total number of shift, complement and addition is 7000, 2241, 3976 respectively and that for BKS process are 6756, 659, 3050. If we consider chips to perform these operations then we can see that the time required for shift operation (4 bit bidirectional Universal shift register LS 194) is 1/36 ns [9] [10], add operation (4 bit carry look ahead adder LS 7483) is 45 ns [9] [10] and for complement operation (LS hex inverter 7404) is 12 ns [9] [10]. Time required for shift, complement operation and add operation in Booth's process is 194 ns, 26892 ns and 178920 ns. Therefor total 206006 ns time is required. And that for our BKS process is 187 ns for shift operation, 7908 ns for complement operation and 137250 ns for add operation. There for total time required is 145345 ns. So here we can understand the gain in time delay.

Here we again mention that in our worst case the complexity of our proposed process of multiplication does not exceed that of Booth's process. It is always less or equal to Booth's process.

COMPLEXITY ANALYSIS

To compute the complexity of our proposed BKS algorithm we have notice the number of partial products. In our proposed algorithm as we check and shift two bits at a time so there is a possibility of occurring maximum n/2 partial product. So the complexity of the proposed process is O(n/2).

FUTURE SCOPE

In the proposed process of multiplication we deal with two bits of multiplier. Here we see that this process gives advantages over Booth's process of multiplication. and we reduce the complexity from O(n) to O(n/2). The actual logic was to reduce the number of partial products. In future we will try to deal with three bits of multiplier so that we can reduce the complexity of the multiplication.

CONCLUSION

The multiplication algorithm presented here is a deviation from Booth’s algorithm only doubling of shift parameter. The result shows that it challenges Booth’s algorithm in time space and cost parameter. Since modified of Booth’s algorithm shows another way of treating our multi objective gain, we attacked the problem in a novel way that may resemble historically Stassen’s matrix multiplication. The analogy may not be right at the first look. However we observed that “History repeats itself”.

REFERENCES

[1]. A. Booth, “A signed binary multiplication technique,” Q. J. Me& Appl. March., vol. 4, pp. 236-240, 19.51.

[2]. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullmant “The design and analysis of computer algorithm” Pearson, 1974.

[3]. Donald K. Knuth “The art of computer programming”, Vol. 1, Addition-Wesley, 2004.

[4]. John P. Hayes “Computer architecture and organization”, Second Edition, McGraw-Hill, 1988.

[5]. M. Morris Mano “Computer System Architecture”, Third Addition, Prentice Hall, 1993.

[6]. Morvin Lee. Minsky “Computation: Finite & Infinite machine: Prentice Hall, 1967.

[7]. S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani “Algorithms” Paperback - Sep 13, 2006.

[8]. http://www.scriptol.com/programming/algorithm-history.php#Origin-Word-Algorithm visited on 18 May 2012

[9]. William D.Anderson, Roberts Loyd Morris, John Miller “Designing with TTL integrated circuit”, McGraw-Hill, 1971

[10]. “TTL data book for design Engineer”, 2nd edition, Texas instrument Inc, 1981