IOSR Journal of Engineering (IOSRJEN)
ISSN: 2250-3021 Volume 2, Issue 6 (June 2012), PP 33-40
www.iosrjen.org

A close look towards Modified Booth's algorithm with BKS Process

Barun Biswas*, Krishnendu Basuli#, Samar Sen Sarma+
**Dept. of Computer Science, West Bengal State University, India*
#Dept. of Computer Science, West Bengal State University, India
+Department of Computer Science and Engineering,
University of Calcutta, 92, A.P.C. Road, Kolkata – 700 009, India.
"Let no such man be trusted" (The Merchant of Vanish)

Abstract – *This paper is remodeling of Modified Booth's algorithm, where shift three at a time. The result shows that classical concept of single shift multiplication algorithms are to be revised with a multiple number of shifts. The observation is that in general the complexity is N/3, which is astonishingly harmonic to the number of bit shift operation. Clearly three shift reduces complexity to reciprocal of three. The matter is that behavior needs a generalized revision of the Booth's multiplicative algorithm.*
Keywords:- Booth's algorithm, Binary multiplication, Arithmetic algorithmic complexity, trade of among relevant parameters, complement number representation.

I. INTRODUCTION

Arithmetic operations are the basic things we learnt from our childhood. We used to perform different arithmetic operations on paper and pencil. Among these different arithmetic operations, like addition, subtraction, multiplication and division, multiplication is one of the most important operations we learnt at our early age. We first learnt to multiply two number by times table and using figure process. As we grew up we learnt different process of multiplication. Now a day many new process of multiplication has been proposed. Multiplication is one of the most important operations used to perform many arithmetic operations. In the following discussion we will try to implement a new process of multiplication so that we can contribute something to speed up the operation execution time and cost of implementing the process in hardware level. During the design of the proposed multiplication process we try to maintain flexibility of the algorithm, we will try to optimize the time as we can.

As we compared the proposed process with Modified Booth's here we mention that in Modified Booth's process of multiplication [1] of the maximum number of partial product is N/2, whereas in our proposed process the maximum number of partial product is N/3. Here is our success to reduce the total number of partial product. In our proposed process there are some cases where we will consider two bits and rest of the cases we will consider three bits. For this reason in some operations we may require more than n/3 partial product (in few cases).

II. SOME BASIC DEFINITTIONS: [3]

A. Partial products:

Partial product is the product operation which is performed during the product of two numbers. Here based on the multiplier products are performed. We shall see later that during one product of a multiplicand and a multiplier we may require many partial products.

B. Complement operation(two's complement):

Two's complement is not a complicated scheme and is not well served by anything lengthy. Therefore, after this introduction, which explains what two's complement is and how to use it, there are mostly examples.
Two's complement is the way every computer I know of chooses to represent integers. To get the two's complement negative notation of an integer, you write out the number in binary. You then invert the digits, and add one to the result.

Suppose we're working with 8 bit quantities and suppose we want to find how -28 would be expressed in two's complement notation. First we write out 28 in binary form.
00011100
Then we invert the digits. 0 becomes 1, 1 becomes 0; i.e. one's complement.
11100011
Then we add 1.
11100100

That is how one would write -28 in 8 bit binary.

C. Shift operation:

In computer programming, an arithmetic shift is a shift operator, sometimes known as a signed shift (though it is not restricted to signed operands). The two basic types are the arithmetic left shift and the arithmetic right shift. For binary numbers it is a bitwise operation that shifts all of the bits of its operand; every bit in the operand is simply moved a given number of bit positions, and the vacant bit-positions are filled in. Instead of being filled with all 0s, as in logical shift, when shifting to the right, the leftmost bit (usually the sign bit in signed integer representations) is replicated to fill in all the vacant positions (this is a kind of sign extension).

III. PREVIOUS WORK:

Before discussing the proposed process of multiplication let us take a look to those previous works that are available at present. Here we first mention though we compare our proposed process of multiplication with Modified Booth's process still we require to discuss the previously worked on multiplication, so that we can show the differences in complexity field.

Repeated addition multiplication is one of the earliest and simplest multiplication processes. It has the complexity of O(n)[4][5] all the time. In this process the multiplier is decreased by one and multiplicand is added to the partial products until multiplier becomes zero.

The next process is to be discussed is "shift and add". Here maximum complexity is O(n)[4]]5]. In this process multiplier is shifted one bit, if it is 1 then multiplicand is added. Here maximum number of partial product is N.

Now the topic is array multiplier[4]]5]. An array multiplier is a digital combinational circuit that is used for the multiplication of two binary numbers by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array. 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.

Divide and conquer is one of the most important process of multiplication. Here the multiplicand and multiplier is divied into parts and then multiplied.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($n^{log 3}$) = O($n^{1.59}$)[2].

The next one of the most important process of multiplication is Booth's process for our discussion, as Modified Booth's process came from Booth's process. The complexity of Booth's process is O(n)[1]. Here the maximum number of partial is N.

As our process challenges Modified Booth's process we will discuss it in next step in details.

IV. MODIFIED BOOTH'S MULTIPLICATION[1]

Modified Booth's multiplication algorithm

is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. This algorithm is a modified version of the Booth's algorithm, 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. Because of some drawbacks of Booth's algorithm Modified Booth's algorithm was designed. In the multiplier bit if there is a series of 0's ansd1's then Booth's algorithm produces worst case complexity. This is why Modified Booth's algorithm was invented.

Modified Booth's algorithm involves repeatedly adding one of two predetermined values Multiplicand and Multiplier to a partial 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 **Q**. Then the number of bit in the product is equal to the (x+y+1).

The Modified Booth's multiplication process is based on the eight basic steps. In this process the three multiplier bit is checked and depending on their combination (2^3=8) eight different steps is performed. Now let us discuss the Modified Booth's multiplication by taking an example.

Let M(multiplicand) =0000000000001101

Q(multiplier) =0000011001100110

M =0000000000001101

Q =0000011001100110

=1111111111100110 // multiplier bit is 100
=0000000000011010** // multiplier bit is 011
= 111111100110**** // multiplier bit is 100
=00000011010****** // multiplier bit is 011
=11100110******** // multiplier bit is 100
=011010********** // multiplier bit is 011
=0101001100101110 //product M*Q

In the above example we notice that there are three operations performed, namely shift, addition and complement.
The number of operation is as follows
Shift = 16
Addition= 5
Complement= 3
In this case the complexity of the Modified Booth's process is maximum.

V. PROPOSED PROCESS OF MULTIPLICATION

In the following passage we shall discuss about our proposed process of multiplication BKS. 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.

A. BKS Algorithm:

The BKS process of multiplication is performed by considering three bits of multiplier. We check three bits and shift three places to the right except when it is last check. Based on the value of these two bits we shall perform four operations. Here we mention that there is some cases when we will consider two bits for the simplicity of our process. They are as follows:

1. If the multiplier bits are 000 then do nothing and shift three places to the right.
2. If the multiplier bits are 001 then add multiplicand and shift three places to the right.
3. If the multiplier bits are 010 then add 2*multiplicand and shift three places to the right.
4. If the multiplier bits are 100 then add 4*multiplicand and shift three places to the right.
5. If the multiplier bits are 111 then subtract the multiplicand, shift three places to the right and check the next two bits
   a) If it is 00 then add multiplicand and shift two places to the right.
   b) If it is 01 then add 2*multiplicand shift two places to the right.
   c) If it is 11 then do nothing, shift two places right and check next two bits.
   d) If it is 10 then subtract multiplicand, shift two places right and check next bit.
      i) If it is 1 then add 2*multiplicand and shift one place to the right.
      ii) Is it is 0 then don't consider this bit and add multiplicand.
6. If the multiplier bits are 011 then subtract multiplicand, shift two places to the right and check next bit.
   a) If it is 0 then add multiplicand and shift two places to the right.
   b) If it is 1 then do not consider this bit, add multiplicand and shift one place to the right.

7. If the multiplier bit is 101 then consider lower two bits and add multiplicand and shift two places to the right.
8. If the multiplier bit is 110 then consider lower two bits and add 2*multiplicand and shift two places to the right.

B. ANALYSIS:

So far we have seen the algorithmic form of the proposed algorithm for multiplication. Now let us take an example and analysis the procedure. To compare with the Modified 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 0000000001010101 respectively. Based on the proposed process let us compute the product.

M(multiplicand) =0000000000001101
Q(multiplier) =0000011001100110

M =0000000000001101
Q =0000011001100110

=0000000000011010 // multiplier bits 110
=00000000001101** // multiplier bits 001
=11111110011*****// multiplier bits 011
=000001101******* // next bit 0
=1110011********* // multiplier bits 011
=01101********** // MSB 0
=0101001100101110 // product M*Q

In the above process of multiplication we notice that the first three multiplier bits are 110 so according to our rule we consider first two bits and add 2*M and then shift partial product two place right and check the next three bits of the multiplier and it is 001 so add M to the partial product. Next three bits are 011 so subtract M, then check next bit, it is 0 so add M. Again next three bits are 011so do the same operation. Here MSB is 0(zero) so stop the process here. And at the end of the process we notice that the number of shift, complement and addition operation are as Shift = 11
Addition= 5
Complement= 2

So in this example we can see that we need to do 5 shift and 1 complement operations less. So we see that in worst case of Modified Booth's multiplication we are gainer.

Now let us discuss some general case where we can prove that our process is advantageous than Modified Booth's algorithm. For the three bits comparison there may be eight possibilities 000, 001, 010, 011, 100, 101, 110 and 111. For these possibilities the different operation is discussed. We also see that for n number of bits there is always at least (n+1) shift operation in Modified Booth's process 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. The complement operations may occur three times where as in our case it is two. The most important case is to notice is that in our proposed algorithm as we check and shift three bits at a time so there is a possibility of occurring N/3 partial product (more than N/3 in some especial case), where as in Modified Booth's process it is N/2 times.

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 …….010111. in this case we can see the for the bit pattern 111 the operation need is 2 complement, 6 shift and 2 addition. For the same bit pattern in Modified Booth process the operations are 1 complement, 6 shift and 2 addition operation. Here is the only case where Modified Booth's require one complement operation less than our, where other process are same.

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.

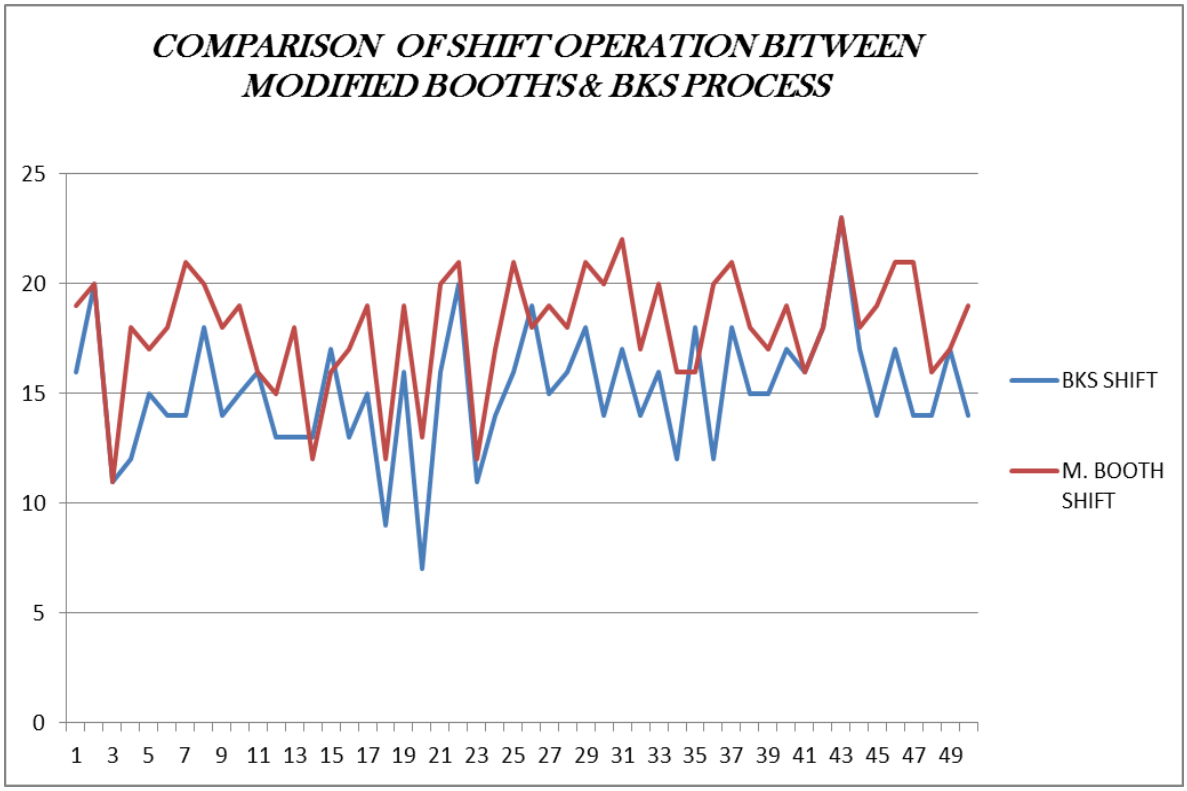
VI. RESULT

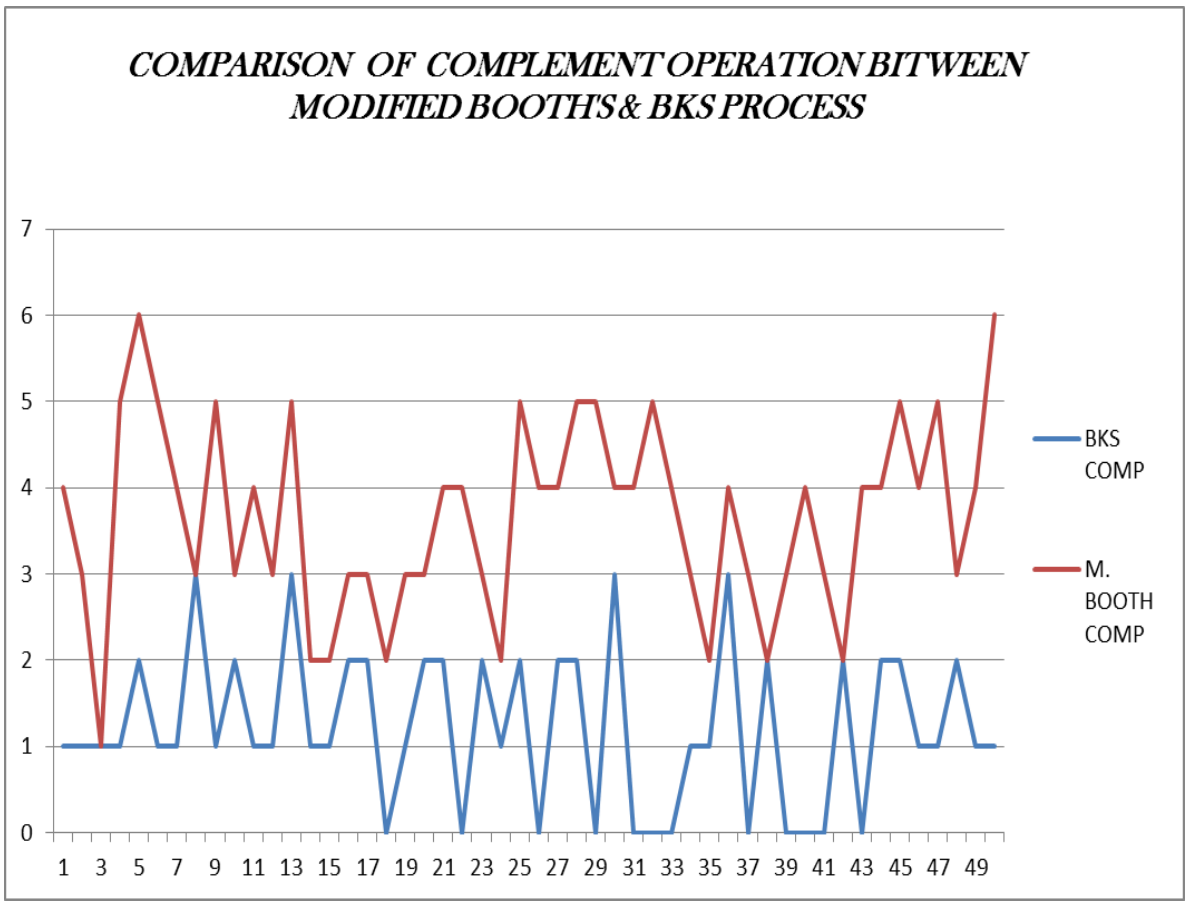
We notice that to perform the multiplication process of both Modified Booth's process and our proposed BKS process only three basics operations are needed namely shift, complement and addition. We perform our proposed process of multiplication and Modified Booth's process of multiplication on 5000 randomly 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%

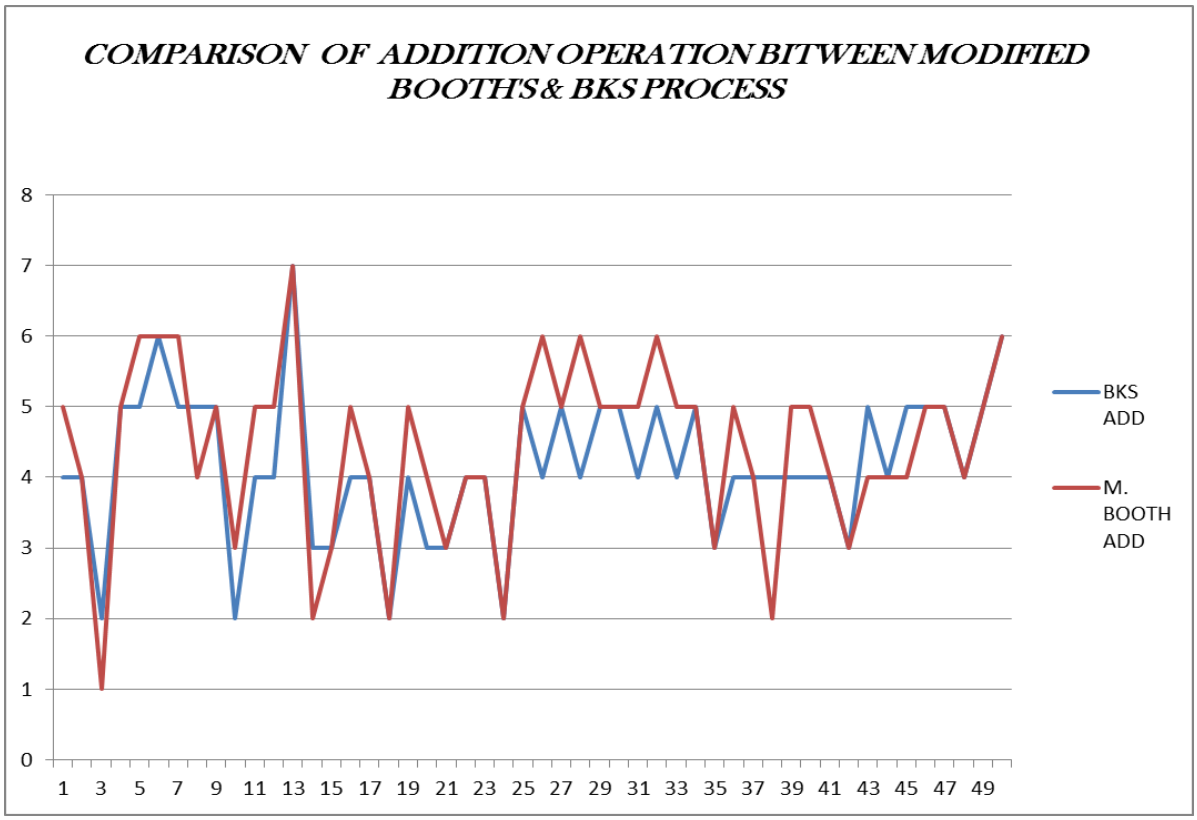
These results are purely an average case advantageous of our proposed BKS process over Modified Booth's process.

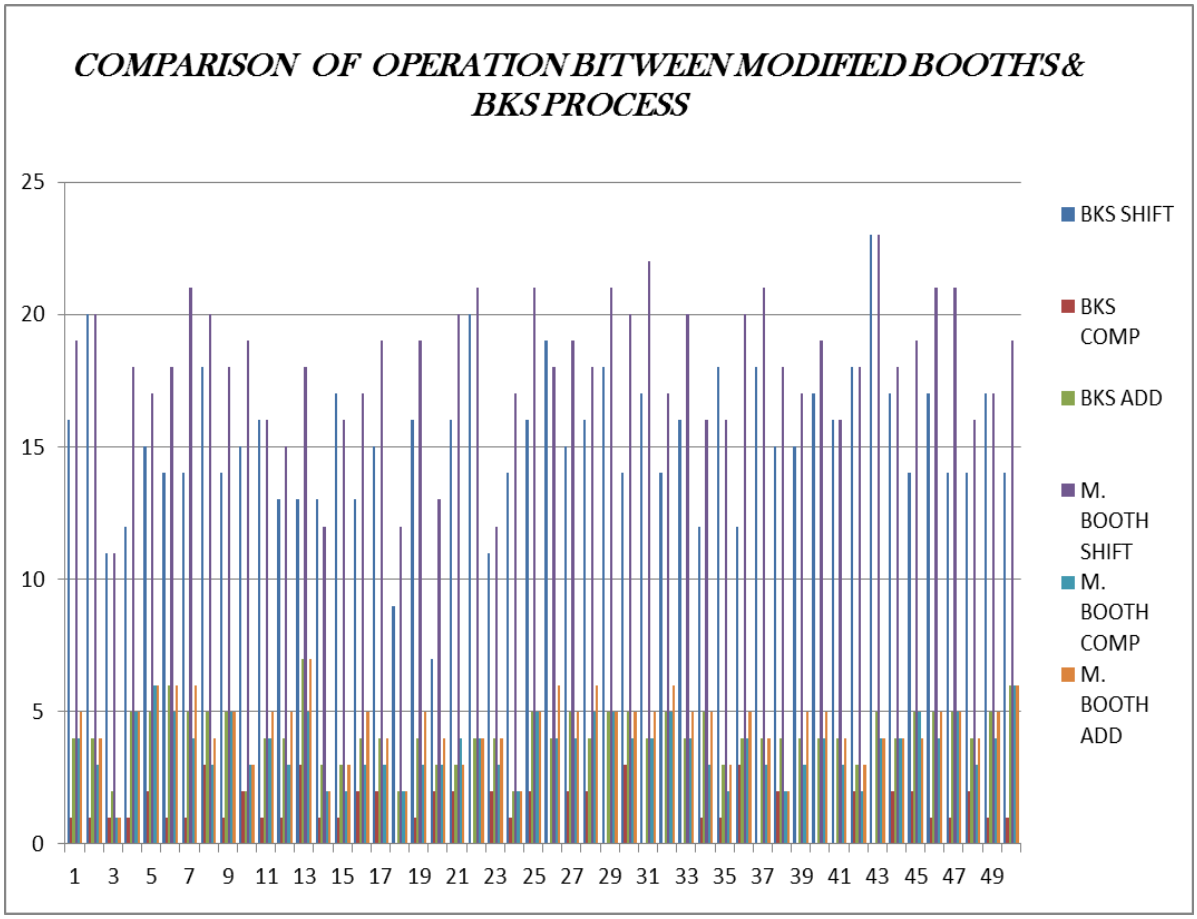
The graph representation of the result (shift, complement and addition operation) performed on 50 data is shown below. Here the randomly generated number (in binary form) is given and also the number of operation required.

Binary number	BKS FOR 3 BIT			M. BOOTH'S		
	Shift	Comp	Add	Shift	Comp	Add
00000000000000000100100010111111	16	1	4	19	4	5
00000000000000000111000010010100	20	1	4	20	3	4
00000000000000000000000011100000	11	1	2	11	1	1
00000000000000000001010100100111	12	1	5	18	5	5
00000000000000000010101010111010	15	2	5	17	6	6
00000000000000000101001011010001	14	1	6	18	5	6
00000000000000000100100111001001	14	1	5	21	4	6
00000000000000000111000011011101	18	3	5	20	3	4
00000000000000000010100101000111	14	1	5	18	5	5
00000000000000000100110111111111	15	2	2	19	3	3
00000000000000000000111010010010	16	1	4	16	4	5
00000000000000000000100011001111	13	1	4	15	3	5
00000000000000000100110010110011	13	3	7	18	5	7
00000000000000000000001111101100	13	1	3	12	2	2
00000000000000000011111111001100	17	1	3	16	2	3
00000000000000000010101111100101	13	2	4	17	3	5
00000000000000000100001110111001	15	2	4	19	3	4
00000000000000000000000100010001	9	0	2	12	2	2
00000000000000000010000011100010	16	1	4	19	3	5
00000000000000000000001011101001	7	2	3	13	3	4
00000000000000000101101111111010	16	2	3	20	4	3
00000000000000000110010000010010	20	0	4	21	4	4
00000000000000000000001111010011	11	2	4	12	3	4
00000000000000000000011000100000	14	1	2	17	2	2
00000000000000000100101011100110	16	2	5	21	5	5
00000000000000000010100010010100	19	0	4	18	4	6
00000000000000000101011100110001	15	2	5	19	4	5
00000000000000000010110111010010	16	2	4	18	5	6
00000000000000000101000110010010	18	0	5	21	5	5
00000000000000000100101110011101	14	3	5	20	4	5
00000000000000000101100010001000	17	0	4	22	4	5
00000000000000000010010101010001	14	0	5	17	5	6
00000000000000000010001001000110	16	0	4	20	4	5
00000000000000000010110100000101	12	1	5	16	3	5
00000000000000000011000000111111	18	1	3	16	2	3
00000000000000000101110011001000	12	3	4	20	4	5
00000000000000000100001001100100	18	0	4	21	3	4
00000000000000000011111111001110	15	2	4	18	2	2
00000000000000000010110010000001	15	0	4	17	3	5
00000000000000000100100001010001	17	0	4	19	4	5
00000000000000000010110000110000	16	0	4	16	3	4
00000000000000000011011111100100	18	2	3	18	2	3
00000000000000000110010010000110	23	0	5	23	4	4
00000000000000000101000010111100	17	2	4	18	4	4
00000000000000000010101011110110	14	2	5	19	5	4
00000000000000000101110000100010	17	1	5	21	4	5
00000000000000000101100101000010	14	1	5	21	5	5
00000000000000000011111111001011	14	2	4	16	3	4
00000000000000000011100101001100	17	1	5	17	4	5
00000000000000000010100110101010	14	1	6	19	6	6


COMPARISON OF SHIFT OPERATION BITWEEN MODIFIED BOOTH'S & BKS PROCESS
25
20
15
10
5
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
BKS SHIFT
M. BOOTH SHIFT


COMPARISON OF COMPLEMENT OPERATION BITWEEN MODIFIED BOOTH'S & BKS PROCESS
7
6
5
4
3
2
1
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
BKS COMP
M. BOOTH COMP


COMPARISON OF ADDITION OPERATION BITWEEN MODIFIED BOOTH'S & BKS PROCESS
8
7
6
5
4
3
2
1
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
BKS ADD
M. BOOTH ADD


COMPARISON OF OPERATION BITWEEN MODIFIED BOOTH'S & BKS PROCESS
25
20
15
10
5
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
BKS SHIFT
BKS COMP
BKS ADD
M. BOOTH SHIFT
M. BOOTH COMP
M. BOOTH ADD

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 5000 data for Modified Booth's process is that total number of shift, complement and addition is 93319, 18124, 24927 respectively and that for BKS process are 76667, 8417, 24680. 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 194) is 1/36 ns [7[8], add operation (4 bit carry look ahead adder 7483) is 45 ns [7][8] and for complement operation (hex inverter 7404) is 12 ns [7][8]. Time required for shift, complement operation and add operation in Modified Booth's process is 2592 ns, 217488 ns and 1121715 ns. Therefor total 1341795 ns time is required. And that for our BKS process is 2129 ns for shift operation, 101004 ns for complement operation and 1110600 ns for add operation. There for total time required is 1213733 ns. So here we can understand the gain in time delay.

VII. 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/3 partial product. So the complexity of the proposed process is O(n/3).

VIII. FUTURE SCOPE

In the proposed process of multiplication we deal with two bits of multiplier. Here we see that this process gives advantages over Modified Booth's process of multiplication and we reduce the complexity from O(n/2) to O(n/3). The actual logic was to reduce the number of partial products. In future we will try to deal with generalized number of bits n.

IX. CONCLUSION

The multiplication algorithm presented here is a deviation from Modified Booth's algorithm only three shift at a time. The result shows that it challenges Modified Booth's algorithm in time space and cost parameter. Here we notice that when the number of shift changes the complexity is also changes. The analogy may not be right at the first look. However we observed that "History repeats itself".

X. 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] William D.Anderson, Roberts Loyd Morris, John Miller "Designing with TTL integrated circuit", McGraw-Hill, 1971
[8] "TTL data book for design Engineer", 2nd edition, Texas instrument Inc, 1981