Green University
GUBJSE: ISSN: 2409-0476

A Fast Multiplication Approach Using A Tree-Based Structure

Md. Solaiman Mia

***Abstract*—** This paper presents a technique for integer number multiplication using a tree-based structure. In the proposed method, both the generation of the partial products and the addition of partial products are completed in the tree structure. The proposed multiplication approach has been designed in two steps: Firstly, the partial products are generated in a tree-based structure using the fewest numbers of gates. Secondly, diagonal partial products additions have been done by the partial products residing in the diagonal partial product nodes to get a faster multiplication result, where two partial product nodes $P_{i,j}$ and $P_{k,l}$ are diagonal only if $|i-k| = |j-l|$ where i and k are the multiplicand bits; and j and l are the multiplier bits. The comparative study shows that the proposed multiplication algorithm outperforms the existing techniques; e.g., the proposed 4×4 multiplication algorithm improves 50% on the worst case running time complexity over the best known existing ones.

***Index Terms*—** Multiplication, Tree, Algorithmic Complexity.

I. Introduction

THE economics of present large-scale scientific computers is suggested to be benefitted from a greater investment in hardware to mechanize multiplication operation. As a move in this direction, a technique is to be developed for a multiplication operation which generates the product of two numbers using purely combinational logic [1].

At first, let's start with a very brief and very simplified explanation of how computers work. A highly simplified explanation of how computers work is given in the following. In general (i.e., actual mileage may vary in some specialized hardware):

- Every computer has at least one CPU (Central Processing Unit), the actual "brain" of the computer which performs every instruction in a program.
- Every CPU has at least one ALU (Arithmetic Logic Unit) or at least its functionality. An ALU could be a distinct unit in the CPU or else its functionality could be incorporated into the CPU's logic. An ALU performs a set of binary integer arithmetic and logic operations on binary values input into it.
- Every CPU also has a number of registers, which are used to contain binary values. A register is hardware circuitry used to hold a pre-determined number of binary digits (bits) specified by the computer design (e.g., 8 bits, 16 bits, 32 bits, 64 bits (tend to follow the powers of two, though many older designs had other word sizes).
- The binary numbers contained in a register can be used as integer values, parts of a floating-point value, the address of a memory location containing data, the address of a memory location containing the address of a memory location containing data, character data, status flags, special data formats, etc. Basically, virtually all the work that a computer does is done in the CPU registers.

Because of the frequent use of arithmetic units such as multipliers and adders, many low-power techniques have been proposed to optimize these functional units in terms of power consumption [2-5]. Among other computing systems, DSP (Digital Signal Processing) applications make extensive use of multiply and accumulate computations. Therefore, the design and the implementation of power-efficient arithmetic units, especially multipliers, is essential for the design of low-power DSP hardware [6].

Multipliers can be categorized to sequential and combinational ones. Sequential multipliers are attractive for their low area requirements. However, they take more time to complete a multiplication operation compared to combinational ones. The primary objective of this work is to introduce a new tree-based multiplication algorithm to reduce the implementation time in practice and to show through the performance analysis that this algorithm is competitive with other more commonly used algorithms when considering the worst case running time complexity. Modest improvement for the 4×4 multiplication algorithm (about 50%) over more conventional algorithms have been shown compared with the proposed algorithm. This work has also

This paper was received on 04 May 2020, revised on 27 September 2020 and accepted on 13 October 2020.

Md. Solaiman Mia, Assistant Professor, Department of Computer Science and Engineering, Green University of Bangladesh. E-mail: *solaiman@cse.green.edu.bd.*

shown that the proposed tree-based multiplication algorithm is based upon the partial product method since the savings due to the reduction of the partial products do not seem to justify the extra hardware required for the generation and distribution of the "Partial Product Nodes" and "Partial Product Addition Nodes".

The paper is organized as follows: the literature review of algorithmic complexity and various multiplication techniques are given in Section II. The proposed approach of tree-based multiplication is described in Section III. The experimental results and comparative analysis are presented in Section IV and the conclusion is given in Section V.

II. Literature Review

In this section, some preliminaries pertaining to algorithmic complexity and different multiplication techniques are described.

A. Algorithmic Complexity

An algorithm is a precise, systematic method for solving a class of problems. Algorithmic thinking, which is a form of mathematical thinking, refers to the thought processes associated with creating and analyzing algorithms. Both algorithms and algorithmic thinking are very powerful tools for problem solving. An integral component of algorithmic thinking is the study of algorithmic complexity, which addresses the amount of resources necessary to execute an algorithm. Through analyzing the complexity of different algorithms, one can compare their efficiencies, and the speed at which they can be performed [7].

There are basically two types of algorithmic complexity: i. Space complexity and ii. Running time complexity. Each complexity can be divided into three different cases like a. Best case complexity, b. Average case complexity and c. Worst case complexity. In this paper, worst case of running time complexities of different multiplication operation is considered and described.

B. Multiplication Techniques

In this subsection, different multiplication techniques along with the worst case running time complexity is explained.

i. Shift-and-add Multiplication [8]

The "long multiplication" of shift-and-add multiplication technique is illustrated in Fig. 1. This technique executes in two steps as following:

Step 1: Compute n partial values, each requiring n single-bit multiplications.

Step 2: Add the partial values (estimate as $(n-1)$ additions of pairs of $(2n-1)$-bit numbers (upper bound)).

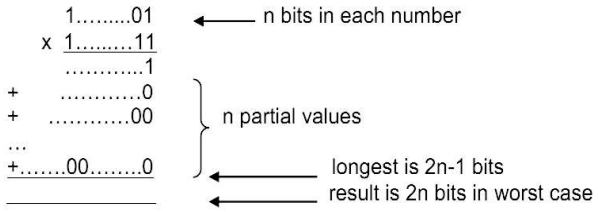

Fig. 1. Shift-and-add Multiplication.

So, the calculation of complexity of the single-bit operation is illustrated in Fig. 2.

Step(1)	n^2
Step(2)	$(2n-1)(n-1) = 2n^2-3n+1$
Total	$3n^2 - 3n+1$

Fig. 2. The Complexity of Shift-and-add Multiplication.

ii. A-La-Russe Method [9]

The idea of this method is to start with two columns where '/' in the first column means integer division by 2 (until the value reaches into 1) and the second column is multiplied by 2 (until the value of first column reaches into 1). This is illustrated in Fig. 3.

a	×	b
a/2		2b
a/4		4b
•		•
•		•
1		•

Fig. 3. Working Procedure of A-La-Russe Method.

Then a third column is created containing a copy of the number from the second column everywhere the number in the first column is odd. Finally, add up this third column to get the result. An example of this method is given in Fig. 4.

19	×	11	11
9	×	22	+22
4	×	44	
2	×	88	
1	×	176	+176
			=209

Fig. 4. An Example of A-La-Russe Method.

There are $O(n)$ entries in the columns, each involving work $O(1)$, since each entry is made by either a right-shift (left column) or by adding a zero (right column). Adding the third column is $O(n^2)$. So, the complexity of this method is also $O(n^2)$ overall – but it's slightly faster than shift-and-add multiplication technique because it is still only $O(n)$ before the addition stage.

iii. Divide-and-Conquer Method [10]

The technique of this method is: any number at first is divided to get the desired result and then the results are merged to get the final result. An example of divide-and-conquer multiplication algorithm is given in Fig. 5.

	Multiply		Shift	Result
i)	09	12	4	108….
ii)	09	34	2	306..
iii)	81	12	2	972..
iv)	81	34	0	2754
				1210554

Fig. 5. An Example of Divide-and-Conquer Method.

The complexity required for the divide operation is $log_2 n$ and for the conquer operation is n. So, the overall worst case running time complexity of divide-and-conquer algorithm is $O(n. log_2 n)$.

iv. Booth's Recoding Algorithm [11]

Booth's algorithm reduced the number of multiplicand multiples. For 4×4 multiplication operation, the worst case running time complexity is $O(n)$. The working procedure is given in follows:

Step 1: Booth's Recoding of the Multiplier Bits.
Step 2: Multiplication of the Recoded Number to the Multiplicand to Generate Partial Products.
Step 3: Adding Sign Extensions.
Step 4: Addition of the Partial Products.

An example along with the working procedure of Booth's algorithm is illustrated in Fig. 6.

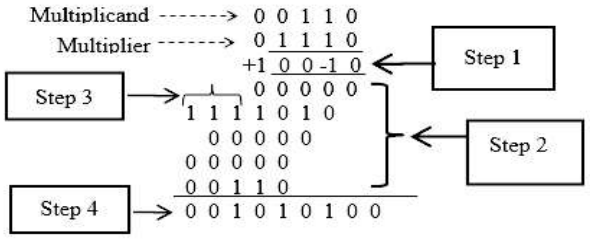

Fig. 6. An Example of Booth's Recoding Method.

All these methods [8-11] work perfectly to get the results of the multiplication operation. But, the methods have different worst case running time complexities. The target of this paper is to find an approach that will do the multiplication operation perfectly and the approach should have the minimum worst case running time complexity as well. The next section describes such an approach.

III. Proposed Approach of Tree-Based Multiplication

There are many multiplication techniques using tree-based architectures in the literature [1]. In this section, a new approach for multiplication using a conventional tree structure is shown, where both the generation of partial products and the addition of partial products for multiplier and multiplicand bits are done by the tree structure. In next subsections, basic definitions and properties are presented and the procedure of multiplication of the proposed approach is explained. The time complexity of the proposed multiplication approach is also discussed in this section.

A. Proposed Approach

Multipliers play an important role in various applications. To achieve a high speed, low power consumption and less area in the multiplier, a tree-based multiplication approach is proposed in this paper.

Some definitions along with some examples are discussed in the following for the better understanding of the proposed technique.

Definition 1: A *Tree-Based* multiplication is a graphical structure for representing the multiplicand bits, multiplier bits, partial product bits and the partial product addition bits, where the non-terminal nodes represent the multiplicand bits, multiplier bits and partial product bits; and the terminal nodes represent the partial product addition bits. In the tree, the non-terminal nodes are denoted as ○ and the terminal nodes are denoted as ▭.

Example 1: A tree-based structure consists of non-terminal nodes and terminal nodes. In Fig. 7, there are seven non-terminal nodes and two terminal nodes which form a tree.

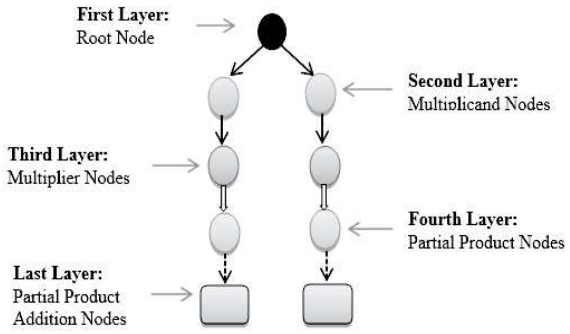

Fig. 7. Tree-Based Structure.

Definition 2: A *Root Node* is denoted by a black circled node which is connecting the multiplicand nodes of a tree-based multiplication structure. The position of the root node is at the first layer (top layer) of a tree.

Example 2: In Fig. 7, the root node can be found at the top of the tree which is a black circled non-terminal node.

Definition 3: *Multiplicand Nodes* are the non-terminal nodes at the second layer of the tree-based multiplication structure which represent the multiplicand bits. These non-terminal nodes are represented by orange circled nodes. Note that, the number of multiplicand nodes are as many as the

multiplicand bits. The most significant multiplicand bit resides at the left-most multiplicand node of the tree and the rest of the multiplicand bits reside in the rest of the multiplicand nodes sequentially towards the right-most node, where the least significant multiplicand bit resides at the right-most multiplicand node of the tree.

Example 3: In Fig. 7, the multiplicand nodes are shown at the second layer of the tree, where they are indicated by orange circled nodes. In Fig. 8, a 2-bit multiplicand A_1A_0 is shown at the second layer, where the most significant multiplicand bit A_1 resides at the left-most multiplicand node and the least significant multiplicand bit A_0 resides at the right-most multiplicand node.

Definition 4: *Multiplier Nodes* are the non-terminal nodes at the third layer of the tree-based multiplication structure which represent the multiplier bits. These non-terminal nodes are represented by red circled nodes. Note that, the multiplier nodes are the successors of a multiplicand node and the number of successors is as many as the number of multiplier bits, where the most significant multiplier bit resides at the left-most multiplier node and the least significant multiplier bit resides at the right-most multiplier node.

Example 4: In Fig. 7, the multiplier nodes are shown at the third layer of the tree, where they are indicated by red circled nodes. In Fig. 8, two multiplier bits B_1B_0 are shown at the third layer, where the most significant multiplier bit B_1 resides at the left-most multiplicand node and the least significant multiplier bit B_0 resides at the right-most multiplicand node.

Definition 5: *Partial Product Nodes* (PPN) are the non-terminal nodes at the fourth layer of the tree-based multiplication structure which represent the partial products formed by ANDing the bits of the connected predecessor multiplicand nodes and the multiplier nodes. Partial Products are used as intermediate steps in calculating the larger products. These non-terminal nodes are represented by blue circled nodes.

Example 5: In Fig. 7, the partial product nodes are shown at the fourth layer of the tree, where they are indicated by blue circled nodes and represent the partial products of multiplicand and multiplier bits. In Fig. 8, two multiplicand bits A_1A_0 and two multiplier bits B_1B_0 are considered at the fourth layer, where the bits of the PPN nodes are formed by ANDing the bits of the connected predecessor multiplicand and multiplier nodes, i.e., PPN node $P_{1,1}$ is formed by ANDing the bits of the connected predecessor multiplicand node A_1 and multiplier node B_1.

Property 1: Let $P_{i,j}$ and $P_{k,l}$ be two PPN nodes, where i and k are the multiplicand bits and j and l are the multiplier bits. $P_{i,j}$ and $P_{k,l}$ are said to be diagonal, only if $|i-k| = |j-l|$.

Example 6: In Fig. 8, PPN nodes $P_{1,0}$ and $P_{0,1}$ are diagonal partial product nodes as $|1-0| = |0-1|$.

Definition 6: *Partial Product Addition Nodes* (PPAN) are the terminal nodes of the tree-based multiplication structure which represent the sum of the partial products of the connected predecessor multiplicand and multiplier bits of the corresponding multiplicand and multiplier nodes. These terminal nodes are represented by rectangle nodes.

Example 7: In Fig. 7, the partial product addition nodes are shown at the last layer of the tree, where they are indicated by rectangle nodes. In Fig. 8, the sum of the bits of PPN nodes $P_{1,0}$ and $P_{0,1}$ resides in the terminal M_1 node.

B. Approach of Multiplication Using a Tree-Based Structure

This approach of multiplication is based on a tree structure. The proposed multiplication approach has two steps: Firstly, a tree is created, where the non-terminal nodes in the second layer are considered as the multiplicand nodes, in which the multiplicand bits reside. All the multiplicand nodes are the successors of a single root node which is resided at the first layer. At the third layer, multiplier nodes are generated as the successors of the multiplicand nodes. At the fourth layer of a tree, the PPN nodes are produced by connecting its predecessor multiplicand and multiplier nodes. A tree-based multiplication is shown in Fig. 8 for multiplying a 2-bit multiplicand by a 2-bit multiplier. Firstly, a tree is constructed by multiplicand nodes, multiplier nodes, partial product nodes and partial product addition nodes. The partial product nodes consist of bits of partial products which are generated by using Algorithm-I. In Fig. 8, the orange colored non-terminal nodes are the multiplicand nodes, the red colored non-terminal nodes are the multiplier nodes, the blue colored non-terminal nodes are PPN nodes and the terminal nodes are the PPAN nodes. Secondly, the addition operation is performed among those PPN nodes which are diagonal. The addition operations are started from the right side of the tree and keep each result of the additions of PPN nodes to the next layer terminal nodes known as PPAN nodes. To get the result, the bits of the additions are collected by traversing PPAN nodes from the left-most side towards the right-most side of the tree. In this technique, the addition of the partial products will be done in parallel while the partial products are producing in the tree. This mechanism helps us to speed-up the multiplication process. This approach is described in Algorithm-I using Example 8.

Algorithm-I: Algorithm of the Proposed Approach of Tree-Based Multiplication

Input: (A_0, A_1, A_2, … , A_{n-1}), (B_0, B_1, B_2, … , B_{n-1})
Output: PP_Node[ ][ ], PPA_Node[ ][ ]
1. Begin
2. For i:=0 to (n-1) do

```
3.        For  j:=0  to  (n-1) do
4.          PP_Node[i][j] = 0
5.        End Loop
6.      End Loop
7.      For  i:=0  to  (n-1) do
8.        For  j:=0  to  (n-1) do
9.          PP_Node[i][j] = AiBi
10.         PP_Node[i][j+1] = AiBj+1
11.         j=j+2
12.       End Loop
13.     End Loop
14.     For  i:=0  to  (n-1) do
15.       For  j:=0  to (n-1) do
16.         if(i==0 && j==0)
17.           M0 = PPA_Node[i][j]
18.         else if(i==j)
19.           Mi+j = PPA_Node[i][j]
20.         End if
21.       End Loop
22.     End Loop
23.   End
```

Example 8: Consider the multiplication of 101 and 10, where 101 is the multiplicand and 10 is the multiplier. As there are three multiplicand bits of 101 and two multiplier bits of 10, a tree is set with three multiplicand nodes (orange colored nodes) $A_2A_1A_0$, where $A_2 = 1$, $A_1 = 0$, $A_0 = 1$, at the second layer. Each multiplicand node has two successive multiplier nodes (red colored nodes) B_1B_0 at the third layer of the tree, where $B_1 = 1$, $B_0 = 0$. At the fourth layer of the tree, the partial product nodes are produced (blue colored nodes) by connecting its predecessor nodes which are given below:

$$P_{0,0} = A_0.B_0 = 1.0 = 0$$
$$P_{0,1} = A_0.B_1 = 1.1 = 1$$
$$P_{1,0} = A_1.B_0 = 0.0 = 0$$
$$P_{1,1} = A_1.B_1 = 0.1 = 0$$
$$P_{2,0} = A_2.B_0 = 1.0 = 0$$
$$P_{2,1} = A_2.B_1 = 1.1 = 1$$

This is the first step of the proposed multiplication method. At the last layer, the terminal PPAN nodes are produced by connecting the diagonal PPN nodes to get the final result 1010 which are shown below:

$$M_0 = P_{0,0} = 0$$
$$M_1 = P_{1,0} + P_{0,1} = 0 + 1 = 1$$
$$M_2 = P_{2,0} + P_{1,1} = 0 + 0 = 0$$
$$M_3 = P_{2,1} = 1$$

So, the result of the multiplication is $M_3M_2M_1M_0 = 1010$.

This is the second step of the proposed multiplication method. The whole process of multiplication is shown in Fig. 9.

IV. Experimental Results and Comparative Analysis

In this section, the worst case running time complexity of the proposed method are analyzed. Then a comparison among the proposed approach and existing methods are given. The analysis of space complexity has not been focused here, rather the worst case running time complexity is the main focus of this paper. For this reason, there is no comparison about space complexity for the multiplication approaches in this paper. In addition, it is known that the best case is the function which performs the minimum number of steps on input data of *n* elements, worst case is the function which performs the maximum number of steps on input data of size *n* and average case is the function which performs an average number of steps on input data of *n* elements. In real-time computing, the worst-case execution time is often of particular concern since it is important to know how much time might be needed in the worst case to guarantee that the algorithm will always finish on time [12]. So, the comparison is made focusing only the worst case running time complexity in this paper.

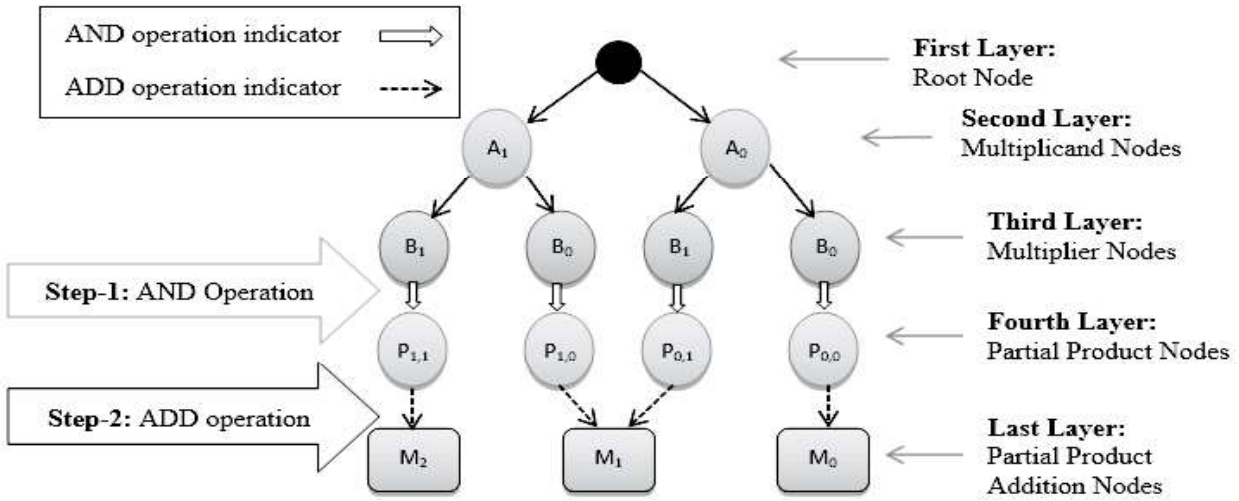

Fig. 8. Proposed Approach of Multiplication Using a Tree-Based Structure.

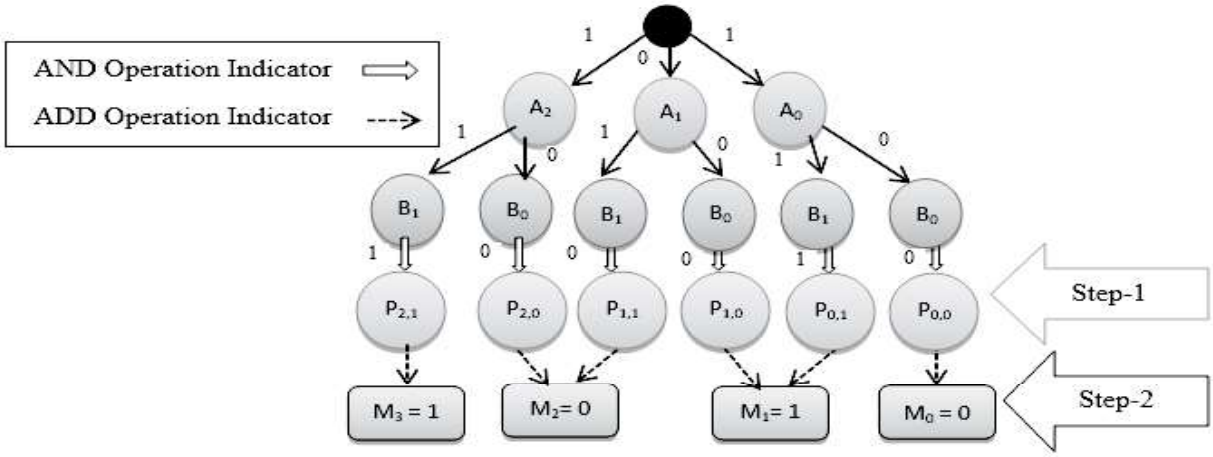

Fig. 9. Example of the Proposed Approach of Multiplication Using a Tree-Based Structure.

Definition 7: The worst case running time complexity is the way in which the number of steps required by an algorithm varies with the size of the problem it is solving. The worst case running time complexity is normally expressed as an order of magnitude, e.g.,$O(n^2)$, where n is the size of the problem.

Theorem 1: The worst case running time complexity of the proposed approach of multiplication is $O(2.log_2 n - 2)$, where n is the number of bits of the multiplier and the multiplicand.

Proof: The above statement is proved by the method of contradiction. Suppose, the worst case running time complexity of the proposed approach of multiplication is not $O(2.log_2 n - 2)$.

In the proposed approach, there are only $O(2.log_2 n - 2)$ steps. Each step has $O(1)$ transmission delay. As a result, the delay of making partial products or the delay of adding partial products is $O(1)$. So, the total delay of all steps is $O(2.log_2 n - 2)$. Therefore, the overall worst case running time complexity is $O(2.log_2 n - 2)$.

This contradicts the supposition that the worst case running time complexity of the proposed approach of multiplication is not $O(2.log_2 n - 2)$. Hence, the supposition is false and Theorem 1 is true and this completes the proof. ■

Example 9: Consider the 4×4 multiplier circuit performing 0110×1110 multiplication. The multiplication using the proposed tree-based technique is shown in Fig. 10. This figure shows that the time complexity of a 4×4 multiplier circuit is $O(2.log_2 4 - 2)$.

A comparison of worst case running time complexity among the proposed approach and existing methods are given in Table I. In addition, since Fig. 10 illustrates an example of the proposed approach of 4×4 multiplication, the 4×4 multiplication is also calculated for all the methods. The results are given in Table II. From this table, it is clearly understand that, for 4×4 multiplication, the best known existing method [11] requires 4 unit of time whereas, the proposed method requires 2 unit of time which is the 50% lower worst case running time.

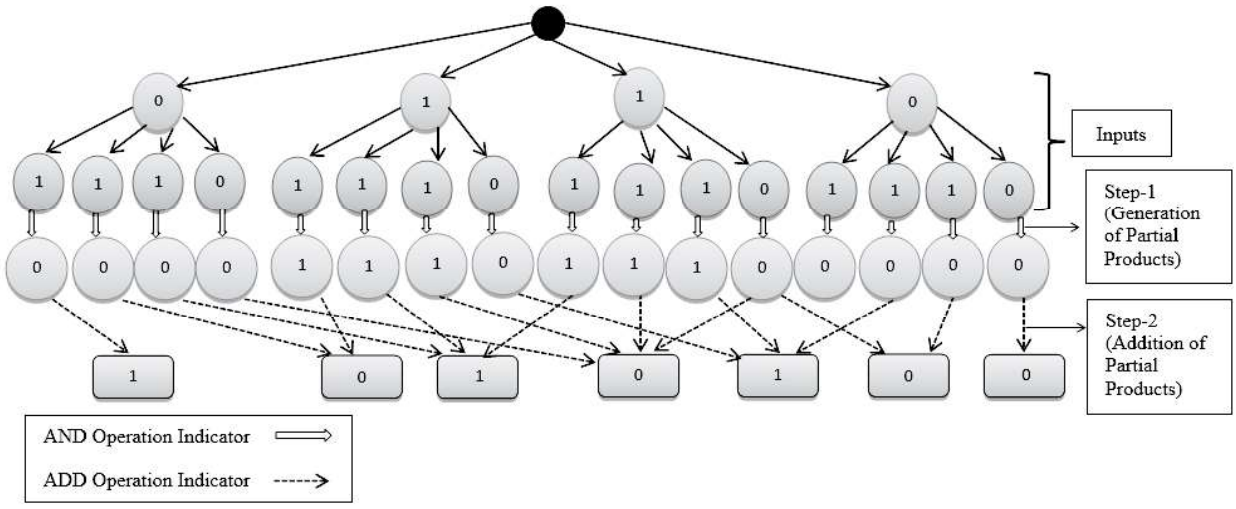

Fig. 10. An Example of the Proposed Approach of Multiplication for *4 × 4* Multiplication.

This statement is also true for 8×8 multiplication also. Table II shows the details results and thus it can be said that, the proposed tree-based multiplication approach gains 50% improvement than the existing methods in the literature. From Table II, it can be said that the improvement is getting higher for the more number of bits of the multiplication.

Table I
Worst Case Running Time Complexity among the Proposed and Existing Methods

Multiplication Approaches	Worst Case Running Time Complexity
Shift-and-add [8]	$O(3n^2 - 3n + 1)$
A-la-russe [9]	$O(n^2)$
Divide and Conquer [10]	$O(n.log_2 n)$
Booth's Recoding [11]	$O(n)$
Proposed Approach	$O(2.log_2 n - 2)$

Table II
Worst Case Running Time Complexity for Different Bits among the Proposed and Existing Methods

No. of Bits	Existing [8]	[9]	[10]	Existing [11]	Proposed Approach
4×4	53	16	8	4	2
8×8	489	64	24	8	4
16×16	721	256	64	16	6
32×32	2977	1024	160	32	8
64×64	12097	4096	384	64	10

V. Conclusions

This paper presents the design methodology of a $n \times n$ multiplier using a tree-based multiplication approach, where n is the number of bits of multiplicand and multiplier. In the proposed multiplication approach, the partial products are produced in a tree-based structure and in the tree, the diagonal partial product nodes perform addition operation to produce the final result. An efficient algorithm is shown based on the proposed multiplication technique to get the desired result of the multiplication operation. The multiplication operation executes in two steps: At first, partial products are generated in a tree-based architecture. After that, the diagonal partial products are added to produce final results. The comparative results prove that the proposed approach is more scalable and performs much better than the existing [8-11] methods.

Acknowledgement

This work was partially supported by the "Research Fund" of Green University of Bangladesh.

References

[1] C. S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electronic Computers, vol. 13, pp. 14-17, 1964.

[2] Y. Liu and S. Furber, "The design of a low power asynchronous multiplier", in Proc. International Symposium on Low Power Electronics and Design, pp. 301-306, 2004.

[3] Z. Huang, "High-level optimization techniques for low-power multiplier design", PhD dissertation in Computer Science, UCLA, 2003.

[4] M. C. Wen, S. J. Wang and Y. N. Lin, "Low-power parallel multiplier with column bypassing", Electronics Letters, vol. 41, no. 10, pp. 581-583, 2005.

[5] I. S. A. Khater, A. Bellaouar and M. I. Elmasry, "Circuit techniques for CMOS low-power high-performance multipliers", IEEE Journal on Solid-State Circuits, vol. 31, no. 10, pp. 1535-1546, 1996.

[6] S. Hong, S. Kim, M. C. Papaefthymiou and W. E. Stark, "Low power parallel multiplier design for DSP applications through coefficient optimization", in Proc. 12th IEEE International ASIC/SOC Conference, pp. 286-290, 1999.

[7] A. A. Nasar, "The history of algorithmic complexity", The Mathematics Enthusiast, vol. 13, no. 3, pp. 217-242, 2016.

[8] F. de Dinechin, S. I. Filip, L. Forget and M. Kumm, "Table-Based versus Shift-And-Add constant multipliers for FPGAs", 26th IEEE Symposium on Computer Arithmetic, pp. 1-8, 2019.

[9] B. Smestad and K. Nikolantonakis, "Historical methods for multiplication", in Proc. European Summer University on the History and Epistemology of Mathematics Conference, 2010.

[10] D. R. Smith, "The design of divide and conquer algorithms", Science of Computer Programming, vol. 5, pp. 37-58, 1985.

[11] A. Booth, "A signed binary multiplication technique", Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236-240, 1951.

[12] "Best, worst and average case." *Wikipedia.org.* https://en.wikipedia.org/wiki/Best,_worst_and_average_case (accessed *Sep.* 22, 2020).

Md. Solaiman Mia received his B.Sc. (Hons.) and M.S. degree from the Dept. of Computer Science and Engineering (CSE), University of Dhaka in 2011 and 2012, respectively. He is currently working as an Assistant Professor in the Dept. of CSE, Green University of Bangladesh. Before this, he worked as an Assistant Professor in the Dept. of CSE in Shanto-Mariam University of Creative Technology, and Dhaka International University. He also worked as a Lecturer in the Dept. of CSE in Hamdard University Bangladesh and Asian University of Bangladesh. Solaiman received the MOICT Fellowship from Bangladesh Government for his research work. His research interests include Reversible Logic Synthesis, Reversible Computing, Quantum Computing, Data Mining etc. He has some experiences as a Reviewer in some international journals and conferences. He is a Professional Member of IEEE, IEEE Computer Society, Life Time Member of Bangladesh Computer Society (BCS) and Internet Society Bangladesh, Dhaka Chapter.