Review on Enhancing Efficiency and Performance of Booth's Multiplication Algorithm

Dr. Renu Chaudhary[1], Akshat S. Malik[2], Dr. Gesu Thakur[3], Dr. Swati Rawat[4]
[1] CSA, COER University, Roorkee, India
[2]CSE, MMU, Ambala, Haryana, India
[3]CSA, COER University, Roorkee, India
[4] CSE, Swami Rama Himalayan University, Dehradun, India

***Abstract*—Booth's multiplication algorithm is a fast and efficient method for multiplying signed integers in 2's complement format. This paper explores its implementation using a controller and data path design. The system uses special storage units (called registers) to hold the multiplier, multiplicand, and results during the calculation. A controller sends signals to manage how data moves through the system. The algorithm looks at two specific bits (Q_0 and Q_{-1}) to decide whether to add, subtract, or shift the numbers. This helps the system correctly multiply both positive and negative numbers. The paper explains how the method works using 8-bit signed numbers and shows how shifting and basic math operations are used to get the final answer. Tests done through simulations show that Booth's algorithm works well for multiplying signed numbers. The paper concludes that Booth's algorithm is an important tool in modern computing for tasks in processors, signal processing, and cryptography.**

Keywords—Booth's multiplication, versatile, cryptography, optimizations, scalability, multiplier, multiplicand, 2's complement.

I. INTRODUCTION

Multiplication is a crucial operation in many electronic and computing systems, and reducing power consumption is essential for modern applications. Among various multipliers, the Booth multiplier is widely used because it makes circuits smaller and faster by efficiently processing signed 2's complement numbers. It enhances efficiency by minimizing the amount of partial products generated, which speeds up the multiplication process compared to traditional methods.

In VLSI design, key goals are high speed, low power usage, and compact size, with speed being especially important. The Booth multiplier works by generating and adding partial products, and its efficiency depends on how quickly these steps are completed. This paper discusses techniques to optimize the Booth multiplier, aiming to save power, reduce area, and enhance speed while minimizing delays [1].

Multipliers in hardware can be designed for speed or efficiency. Fast multipliers need more hardware, while slower ones need less. So, it's important to find a good balance between speed and how much hardware is used. Since multipliers are very important in high-performance systems, their design can affect how well the whole system works. That's why multipliers should be made to work quickly without using too much hardware or space.This has led to a review of Booth's Algorithm, including its modified versions like radix-2, radix-4, and radix-8, to find optimized solutions [2].

Saving power is very important in today's technology because devices are used in many different applications. Multipliers are key parts of fast systems like FIR filters, microprocessors, and DSPs, and they are important in designing circuits. Booth's algorithm is commonly used to create multipliers, but it has some problems—it creates more intermediate results, which can make the chip bigger and slower. To fix this, the Modified Booth Algorithm (MBA) is used because it reduces the number of these results. This paper discusses different ways and techniques to design better multipliers that work faster, use less space, and save more power [3].

This paper shows how a Booth multiplier is designed to work with both positive (unsigned) and negative (signed) numbers. The design is explained using an RTL diagram. Since multiplication uses a lot of hardware and is important in areas like digital signal processing and graphics, the paper focuses on building a 256-bit Booth multiplier. It also compares how well it performs with smaller versions, like 64-bit and 128-bit multipliers. The Booth algorithm simplifies operations by uniformly handling positive and negative operands through addition and subtraction.

The multiplier is designed and verified using Verilog HDL, with Model Sim for simulation and Xilinx for synthesis. The paper offers an in-depth examination of the Booth algorithm's radix-64, radix-128, and radix-256 variants, highlighting their efficiency and performance [4].

This paper reviews existing research on the classical Booth's algorithm for multiplication and introduces an enhanced version of its general form. In both the original and modified Booth's algorithms, two or three bits of the multiplier are processed at a time. The proposed approach allows for the use of any number of bits from the multiplier, aiming to identify the optimal number of bits that reduces operation complexity. The goal is to achieve more efficient multiplication by minimizing the overall complexity of the process [5].

This paper reviews different multiplier designs used in applications like DSP, microprocessors, and microcomputers. It focuses on the Booth algorithm but highlights its limitations, such as increased partial products, which lead to larger area, height, and latency. To overcome these issues, the paper explores a modified Booth algorithm that reduces partial products. It also covers various addition algorithms used in multipliers and emphasizes the importance of reducing power dissipation in modern VLSI designs [6].

Low power consumption and compact design are essential for DSP and high-performance systems. Optimizing the speed and area of multipliers is challenging, as improving one often sacrifices the other—faster multipliers require more area, and reducing area can slow them down. This project seeks to find the best balance by comparing different multipliers. Since multipliers are key to overall system performance, it's important to design them to be both fast and efficient. This need made us study Booth's Algorithm carefully [7].

This review paper looks at different ways to design the Booth multiplier and explains why it is important in digital signal processing, where fast multiplication is needed. The Modified Booth multiplier works faster and better because it produces fewer partial results during the multiplication process. The paper also covers various addition algorithms used in multiplier operations [8].

This paper explains a new way to build a parallel multiplier that uses an improved Booth method to work with 2's complement numbers. Traditional Booth's recoding algorithms are sequential, requiring add and shift operations. Parallel multiplication can be done using Booth's method and Brown's adder array, but it needs more adders to get correct answers. The proposed design improves this by using a modified Booth's recoding unit and a reduced number of adders. The design also includes multiplexers and supports radix-2 and radix-4 architectures for 4-bit and 8-bit operations. It has been checked and built using Xilinx 13.2 software[9].

This paper shows a method called Redundant Binary Partial Product Generator that helps reduce one row from the partial product array in a radix-16 Modified Booth multiplier, without making the process slower. The optimization targets radix-4 modified Booth recoded multipliers, this lowers the maximum height of partial product columns to [n/4] for 64-bit numbers, instead of the usual [(n+1)/4]. This makes Arithmetic Logic Units (ALUs) and processors work more efficiently. The proposed approach is compared to standard Booth multipliers, showing enhancements in area, delay, and power consumption through logic synthesis. Simulation results confirm that the optimized multiplier design outperforms traditional designs, especially for 64-bit and nnn-bit operands. The delay and area of the architecture are analysed using the Xilinx 14.2 tool, demonstrating its superior performance [10].

Multipliers are important parts of multimedia and DSP systems because they greatly affect how well the system performs and how much power it uses. For optimal performance, efficient multiplier design is essential. In many DSP applications, fixed-point arithmetic is used, where n-bit signals are multiplied by n-bit coefficients, and the resulting 2n-bit products are reduced to n bits to prevent overflow. In multimedia systems, some loss of accuracy is acceptable in exchange for better performance in terms of speed, area, and power dissipation.

This paper presents a design for a complex multiplier using a radix-4 Booth multiplier and a carry-select adder (CSLA) with a BEC adder. The design of a 64-

bit complex multiplier is simulated using Xilinx software, and key parameters like the number of lookup tables, flip-flops, and maximum combinational path delay are calculated [11].

Importance of Booth Multiplication
Booth's Algorithm is a simple method used to multiply two signed binary numbers. It works with two's complement form and helps make multiplication faster by reducing how many times we need to add or subtract.

- Easily Works with Positive and Negative Numbers

Booth's algorithm can easily work with positive and negative numbers because it uses two's complement form. This makes it simpler to build hardware that can do signed number multiplication.

- "+" or " - " Uses Fewer Steps to Multiply

Booth's algorithm makes multiplication quicker by reducing the number of times we need to add. When there are many 1s in a row, it treats them as a group instead of handling each one separately.

- Works Well with Computer Circuits

Booth's algorithm is built to fit well with digital hardware, which makes it simple to use in parts of a computer like the Arithmetic Logic Unit (ALU) that handles calculations.

- Improves Performance for Certain Patterns

When the number we are multiplying has a lot of 1s in a row, Booth's algorithm makes the work easier by doing fewer steps. It replaces many additions with just one subtraction and a simple move (shift).

- Easy Way to Learn Bigger Methods

Booth's algorithm is a simple method that helps beginners understand how multiplication works in computers. It also makes it easier to learn more advanced methods later on.

- Faster Multiplication

Booth's algorithm helps to multiply numbers quickly, especially when the numbers are big.

- Less Adding, More Shifting

Instead of adding again and again, it just moves the numbers (shifting), which is faster.

- Used in Computers and Calculators

Many computers and calculators use this method because it is simple and works well.

- Good for Learning Binary Math

It helps students understand how to multiply using 0s and 1s in binary.

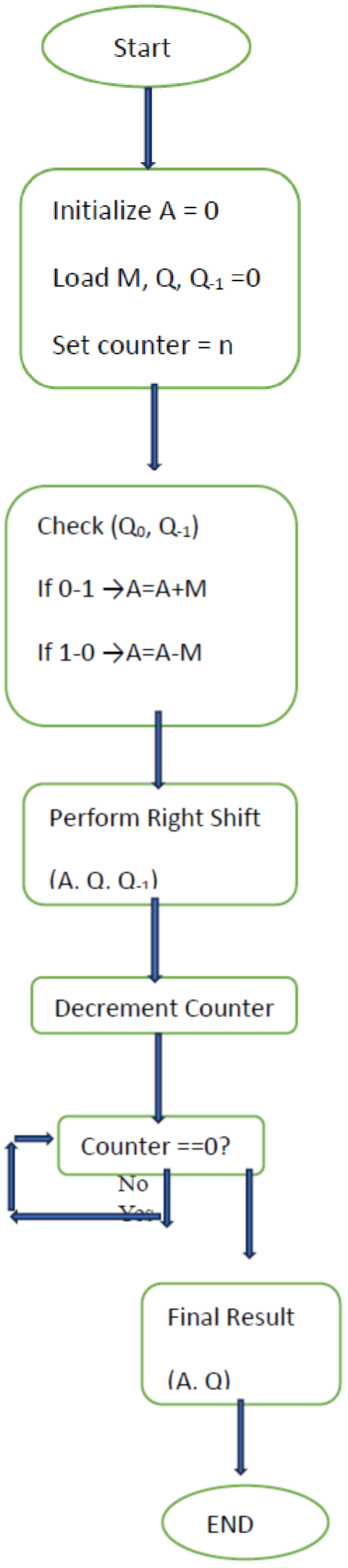
- Takes Less Space in Machines

It uses fewer parts in computer circuits, so devices can be smaller and cost less.

- Lets Computers Multiply Automatically

Booth's algorithm allows computers to do multiplication on their own without extra help.

Flow chart of Booth multiplication :



Booth's algorithm can be done in different ways. In this experiment, it is done using two main parts: a controller and a data path. The controller gives instructions, and the data path follows them. The data path has important parts like registers to hold the multiplier, multiplicand, and results, along with units that do calculations, such as the ALU and adder/subtractor. The diagram shows Booth's multiplier for two 4-bit 2's complement numbers. The data path includes three registers: M, Q, and A. Here, M stores the multiplicand, Q holds the multiplier, and A contains the results from the adder/subtractor operations. A down counter tracks the operations needed. There are five control signals—load, add, subtract, shift, and decrement—that help move the data around. The controller creates these signals based on information it gets from parts called Q_0, Q_{-1}, and the counter.

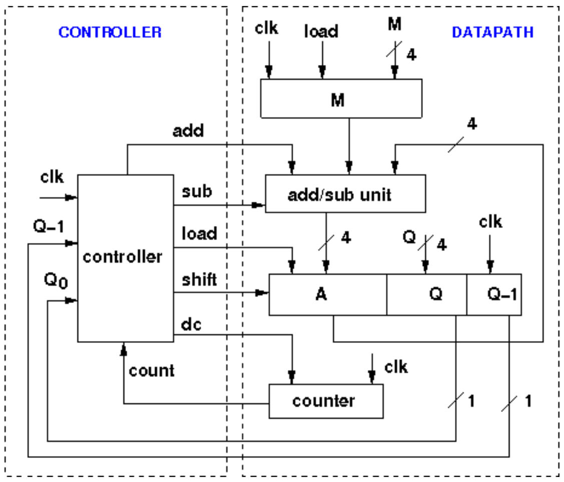


Figure 1: Overview of CPU Components – Control Unit and Data Handling Path

Booth's multiplication algorithm is a way to multiply two signed numbers that use 2's complement form. As shown in the picture, this method uses fewer addition and subtraction steps than simple multiplication methods. In Booth's multiplication algorithm, the multiplicand is placed in the A register, and the multiplier is stored in the Q register. There's also a 1-bit register called Q_{-1}, located just to the right of Q_0, which is the least significant bit of Q. At the beginning, both the A register and Q_{-1} are initialized to 0.

The control logic keeps checking the values of Q_0 and Q_{-1}. If both bits are the same (either 00 or 11), it shifts all bits in the A, Q, and Q_{-1} registers one position to the right. If the bits are different, and the pattern is 10, it subtracts the multiplicand from A. If the pattern is 01, it adds the multiplicand to A. In either case, the result is stored back in A, and afterwards, A, Q, and Q_{-1} undergo a right shift operation.

The arithmetic right shift keeps the sign (positive or negative) of the numbers in A and Q by moving the leftmost bit one place to the right and keeping it the same. The final answer of the multiplication is saved in the A and Q registers.

II. EXPLANATION

- Initialization phase: Start by assigning the initial values to the variables A, Q, M, and Q_{-1}.
- Iteration: For each bit in the multiplier Q, examine the current bit (Q_0) and the previous bit (Q_{-1}) to decide whether to add or subtract the multiplicand M from the accumulator A.
- Shift Operation: Following each addition or subtraction, perform an arithmetic right shift on A, Q, and Q_{-1}.
- Result: Once all bits of Q have been processed, the final multiplication result is contained in A.

Step-by-Step Calculation using Booth's Algorithm
In the context of Booth's multiplication algorithm, Q_{-1} (which you've referred to as Qres) is an additional bit used to assist with the decision-making process in the algorithm. It starts at 0 and is important in deciding whether to add, subtract, or take no action at each stage of the multiplication process. Here's an explanation of how Q_{-1} (or Qres) is used:

If the current least significant bit Q_0 *and* Q_1 form the pair " 01", then this indicates a transition from 0 to 1, and the multiplicand should be added to the accumulator.

If Q and Q_{-1} form the pair "10", this indicates a transition from 1 to 0, and the multiplicand's complement should be subtracted from the accumulator.

If Q and Q_{-1} form the pairs "00" or "11", no arithmetic operation is needed, and only a right arithmetic shift is performed.

This auxiliary bit helps to determine the correct operation at each step and is updated during the arithmetic shift operations.

To show how Booth's multiplication algorithm works with the numbers -22 and 24, we'll go through the steps mentioned earlier. This algorithm is especially effective for multiplying signed numbers, as it can easily handle both positive and negative values.
First, let's represent the numbers in binary:
−22 in 8-bit binary (using two's complement): −22=11101010

24 in 8-bit binary: = 00011000
A is initialized to 0
Q is the multiplicand, 24 in binary
Q_{-1} is initialized to 0
M is the multiplicand −22
Here's the step-by-step process of Booth's multiplication:

Initial Values:
- M = −22 = 11101010
- −M = 22 = 00010110
- Q = 24 = 00011000
- Q_{-1} = 0
- A = 0

Table 1: *for Booth's Algorithm*

Step	A	Q	Qres	Operation
Init	000000000	00011000	0	Initialization
1	11101010	00011000	0	A = A + M
	11110101	10001100	0	Arithmetic Shift Right
2	11110101	10001100	0	No Operation
	11111010	11000110	0	Arithmetic Shift Right
3	11111010	11000110	0	No Operation
	11111101	11100011	0	Arithmetic Shift Right
4	00010111	11100011	0	A = A + -M
	00001011	11110001	0	Arithmetic Shift Right
5	00001011	11110001	0	No Operation
	00000101	11111000	0	Arithmetic Shift Right
6	00000101	11111000	0	No Operation
	00000010	11111100	0	Arithmetic Shift Right
7	00000010	11111100	0	No Operation
	00000001	11111110	0	Arithmetic Shift Right
8	00000001	11111110	0	No Operation
	00000000	11111111	0	Arithmetic Shift Right

III. RESULTS AND DISCUSSION

A = 00000000
Q = 11111111
So, the final product is Q=1111111111111110, which represents -528 in decimal
(as it should, because −22×24=−528).
Booth's multiplication algorithm multiplies two binary numbers correctly, including negative ones. It uses shifting and adding or subtracting steps, which are controlled by signals based on the values of Q and Q_{-1}.

IV. CONCLUSION

Booth's multiplication algorithm is very important in computing because it can multiply signed binary numbers quickly and efficiently. It is used in many places, from simple math in processors to advanced tasks like signal processing and cryptography, making computing faster and better in many fields.

Booth's algorithm provides a quick and simple way to multiply large binary numbers. It uses shifting instead of repeated addition, making it efficient. It's commonly used in computers and calculators, helps save space in hardware, and is great for learning binary math.

REFERENCES

[1] B. L. Gundapaneni and J. R. K. Kumar Dabbakuti, "Booth algorithm for the design of multiplier," Int. J. Innov. Technol. Explor. Eng., vol. 8, no. 7, pp. 1506–1509, 2019.
[2] D. Chandel, G. Kumawat, P. Lahoty, V. V. Chandrodaya, and S. Sharma, "Booth Multiplier : Ease of multiplication," vol. 3, no. 3, pp. 2–6, 2013.
[3] B. Kaur and V. Thakur, "Review of Booth Algorithm for Design of Multiplier," vol. 4, no. 4, pp. 2–5, 2014.
[4] G. S. Surabhi, J. Bhavana, S. Madhu, and B. P. Arunarao, "Design and Implementation of 256 * 256 Booth Multiplier and its Applications," vol. 8, no. 11, pp. 285–288, 2020.
[5] B. Biswas and B. B. Chaudhuri, "Generalization of Booth's Algorithm for Efficient

Multiplication," Procedia Technol., vol. 10, no. July, pp. 304–310, 2013, doi: 10.1016/j.protcy.2013.12.365.

[6] D. Rathore, "A Review Articles of Booth Multiplier," vol. 5, no. 3, pp. 815–819, 2019.

[7] N. Goyal, K. Gupta, and R. Singla, "Study of Combinational and Booth Multiplier," vol. 4, no. 5, pp. 4–7, 2014.

[8] J. Kalia and V. Mittal, "A Review of Different Methods for Booth Multiplier," Int. J. Eng. Res. Appl., vol. 07, no. 05, pp. 60–63, 2017, doi: 10.9790/9622-0705046063.

[9] M. Santhosh and D. S. Rao, "A Novel Architecture of Parallel Multiplier Using Modified Booth's Recoding Unit and Adder for Signed and Unsigned Numbers," vol. 2, no. 8, pp. 55–61, 2015.

[10] S. Q. Victoria and K. V. Babu, "Design Of Area And Power Efficient Booth Multipliers Using Modified Booth Encoding," vol. 5, no. 8, pp. 1–7, 2020.

[11] U. S. Sengar, C. Engineering, and C. Engineering, "Minimize Delay and High Speed Complex Booth Multiplier using Carry Select Adder with BEC," vol. 10, no. 5, pp. 140–146, 2023.

[12] S. Bhongale and R. Patel, "Implementation of Radix-4 (32 bit) Booth Multiplier using VHDL," vol. 12, no. 06, pp. 297–306, 2023.

[13] Y. J. Chang, Y. C. Cheng, S. C. Liao, and C. H. Hsiao, "A Low Power Radix-4 Booth Multiplier with Pre-Encoded Mechanism," IEEE Access, vol. 8, pp. 114842–114853, 2020, doi: 10.1109/ACCESS.2020.3003684.

[14] R. HariKrishna and V. Srinivasulu, "DESIGN OF BOOTH ENCODED MODULO 2n-1 MULTIPLIER USING RADIX-8 WITH HIGH DYNAMIC RANGE RESIDUE NUMBER SYSTEM," vol. 3, no. 1, 2014.

[15] Jason, K, Arivazhagan, P., and Arunkumar, P., "Design of Accuracy Configurable Booth Multiplier using Sorting based Compressors," Irish Interdiscip. J. Sci. Res., vol. 07, no. 03, pp. 98–105, 2023, doi: 10.46759/iijsr.2023.7311.

[16] V. Sharma and R. P. Agarwal, "Multiplier Design Based on Booth and Sequential Algorithm," Int. J. Recent Technol. Eng., vol. 12, no. 6, pp. 1–4, 2024, doi: 10.35940/ijrte.f7997.12060324.