Comparison of Modified Booth Multiplier Techniques

Meghana M N[1]; Mallikarjuna R Mulimani[2]; Shiva prasad A S[3]; Venkatesh R[4]; Vignesh D[5]

[1]Assistant Professor, [2, 3, 4,5]Students

[1, 2, 3, 4,5]Department of Electronics and Communication Engineering Maharaja Institute of Technology Thandavapura (MITT) Mysuru, India

Publication Date: 2025/06/03

Abstract: Booth's Algorithm is a multiplication algorithm used to perform signed binary multiplication efficiently. It minimizes the number of addition and subtraction operations by encoding runs of consecutive ones in the binary representation of a multiplier. This algorithm uses a technique called radix-4 encoding, which reduces the number of required arithmetic operations compared to standard long multiplication. Booth's Algorithm is widely used in computer arithmetic, especially in hardware multipliers, due to its ability to handle both positive and negative numbers uniformly. This paper provides an overview of the algorithm's working mechanism, its advantages, and its significance in digital computing.

Keywords: *Booth's Algorithm, Signed Binary Multiplication, Radix-4 Encoding, Computer Arithmetic, Hardware Multipliers, Digital Computing, Arithmetic Operations Optimization, Binary Multiplier, Multiplication Algorithm, Run-Length Encoding.*

How to Cite: Meghana M N; Mallikarjuna R Mulimani; Shiva prasad A S; Venkatesh R; Vignesh D (2025) Comparison of Modified Booth Multiplier Techniques. *International Journal of Innovative Science and Research Technology,* 10(5), 3017-3022. https://doi.org/10.38124/ijisrt/25may1725

I. INTRODUCTION

Booth's algorithm is a widely used algorithm to reduce the number of partial products in multiplication, thereby increasing computational efficiency and speed. It does so by encoding binary numbers into groups of bits, enabling operations to skip redundant calculations by handling multiple bits at once. Reduces the number of addition operations, especially useful for signed binary numbers, making it highly effective in handling two's complement numbers. By generating fewer partial products, Modified Booth reduces hardware complexity and overall power consumption, making it more suitable for VLSI designs.

II. METHODOLOGY

In this study, various Modified Booth Multiplier (MBM) techniques have been selected for comparative analysis, including the conventional MBM, Radix-4 MBM, Low Power MBM, and High-Speed MBM architectures. Each multiplier design was modeled and implemented using Verilog HDL and simulated using the Xilinx Vivado simulation environment to verify functional correctness. All designs were synthesized targeting the same FPGA device to ensure uniformity in performance evaluation. A consistent set of random and boundary condition input vectors was applied through a standardized test bench for all multiplier implementations. The performance of each technique was assessed based on key parameters such as area utilization (measured in slices), total Power consumption, critical path delay, and maximum operating frequency. Synthesis reports generated by the Vivado tool provided quantitative data for comparison. Additionally, the results were organized into comparative tables and graphical representations to clearly illustrate the trade-offs and advantages of each technique. This methodology ensures a fair, consistent, and comprehensive evaluation of different Modified Booth Multiplier architectures under identical design constraints. Through a clean and intuitive interface. Once an image is uploaded, the backend processes it, performs prediction using the deep learning model, and returns the results in real-time, including the predicted tumor type and heatmap visualization. The complete system is designed to aid doctors and users in early and efficient brain tumor detection, making it accessible via local or cloud-based deployment.

➢ *Booth Algorithm*

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation in efficient way, i.e.,less number of additions/subtractions required. It is also used to speed up the performance of the multiplication process.

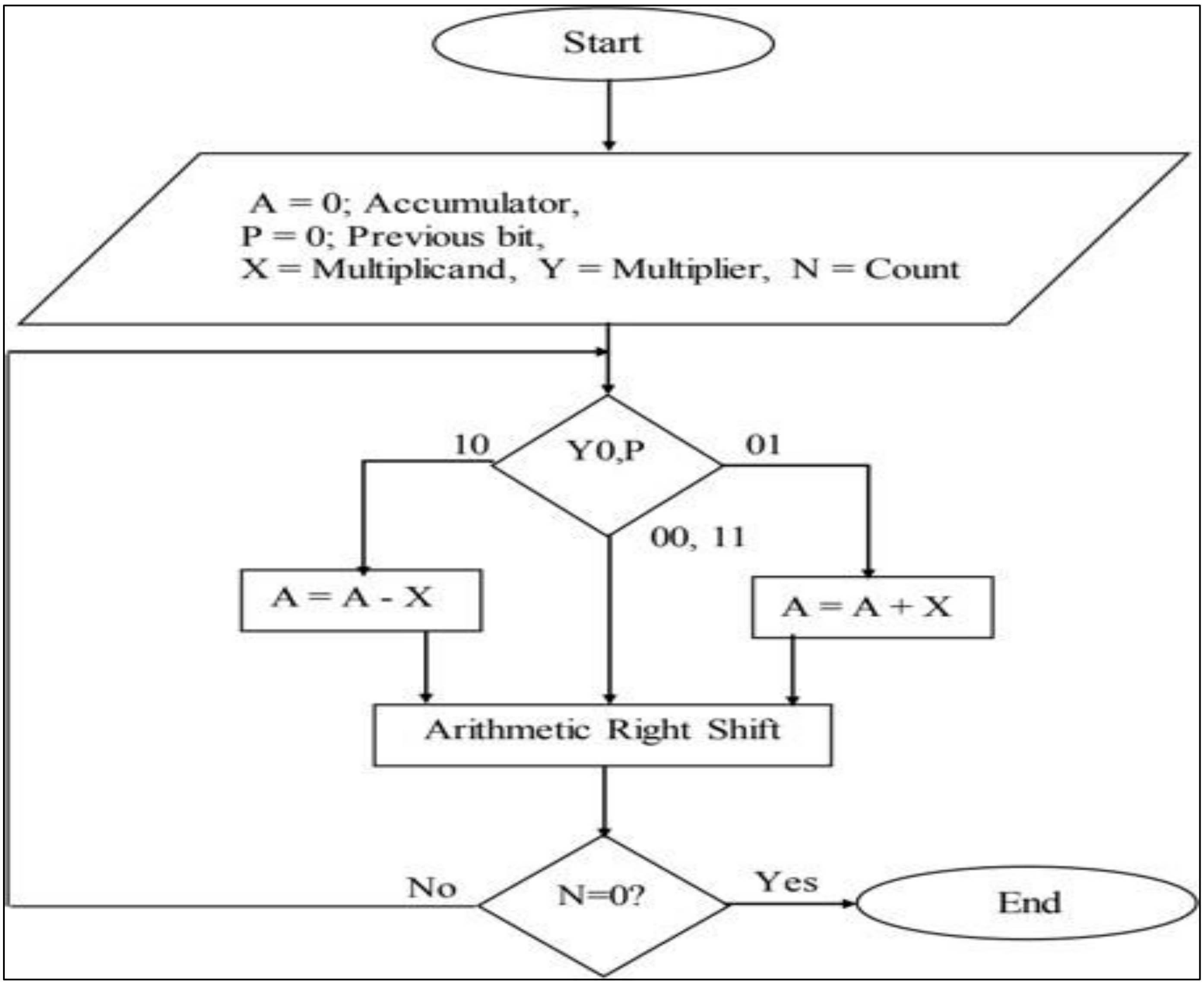


Fig 1 Flow Chart of Booth Algorithm

The Following Flowchart Outlines the Basic Steps of the Algorithm:

- *Step 1*

Begin by setting the value of the register "A = 0" to the first operand and the value of the Previous state register "P = 0" to the second operand.

- *Step 2*

Check the value of the Least Significant Bit (LSB) of the multiplier "Y" i.e. Y0 and compare With the value of register "P", if

✓ Y0, P = 00 or 11 – Perform Arithmetic Right Shift
✓ Y0, P = 10 perform A = A – X and then Arithmetic Right Shift
✓ Y0, P = 01 perform A = A + X and then Arithmetic Right Shift

- *Step 3*

Check if the value of Count "N = 0", If it is, the algorithm is complete and the value in the Register "Y" is the result. If the value is not zero, go back to step 2.

III. MODULES AND ITS IMPLEMENTATION

➢ *Booth's Encoding*

Booth encoding is a method of recoding binary numbers to optimize the multiplication process. It simplifies handling negative numbers and reduces the number of addition/subtraction operations required.

➢ *Radix 4 Booth's Encoding*

This examines 3 bits at a time, the current pair of bits from the multiplier and an additional bit from the previous position generating partial product

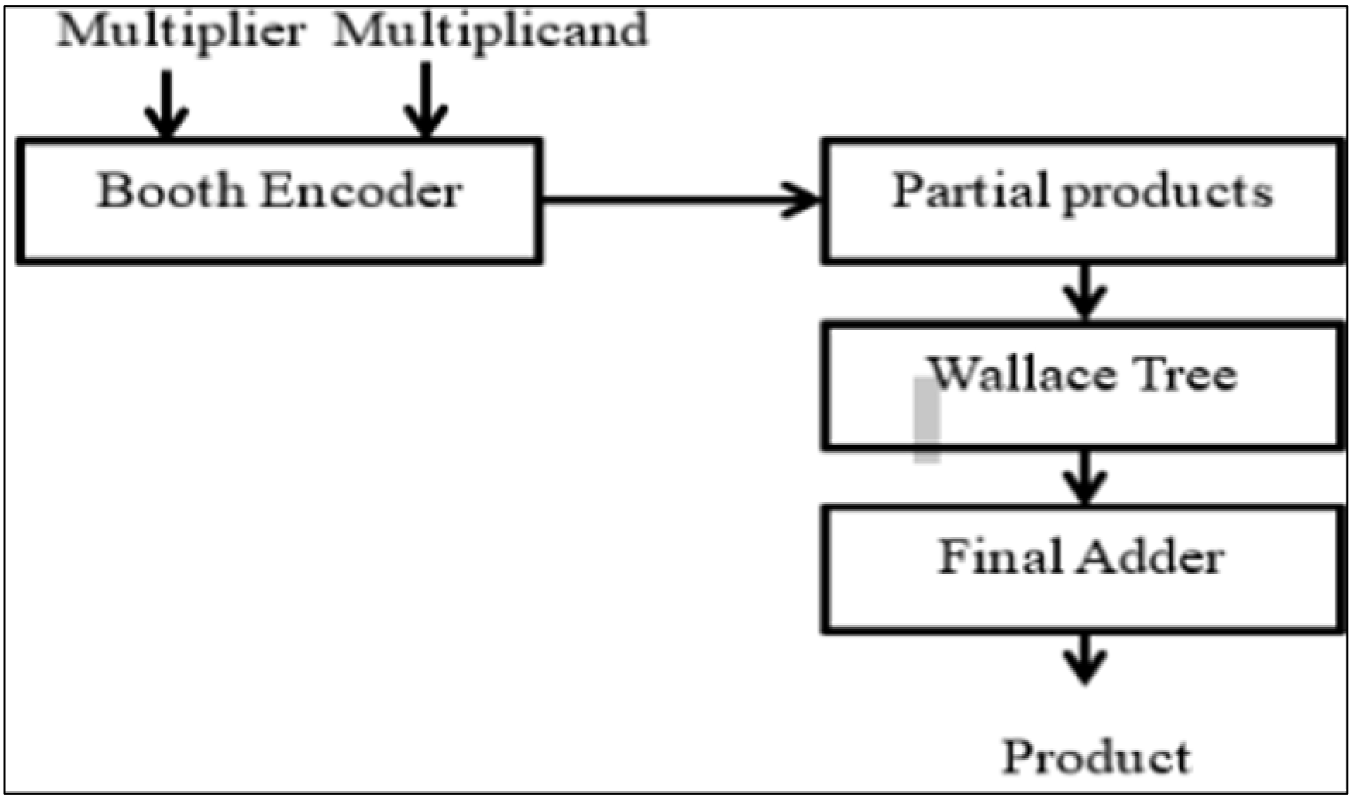


Fig2 Flow Chart of Booth Encoding (Radix 4)

The overall working of the algorithm can be divided into the following stages:

- Radix-4 Booth Encoding
- Partial Product Generation
- Wallace Tree Reduction
- Pipeline Stages
- Final Addition
- FSM

All modules are written in Verilog and tested using a simulation testbench. Each stage is validated individually and then integrated

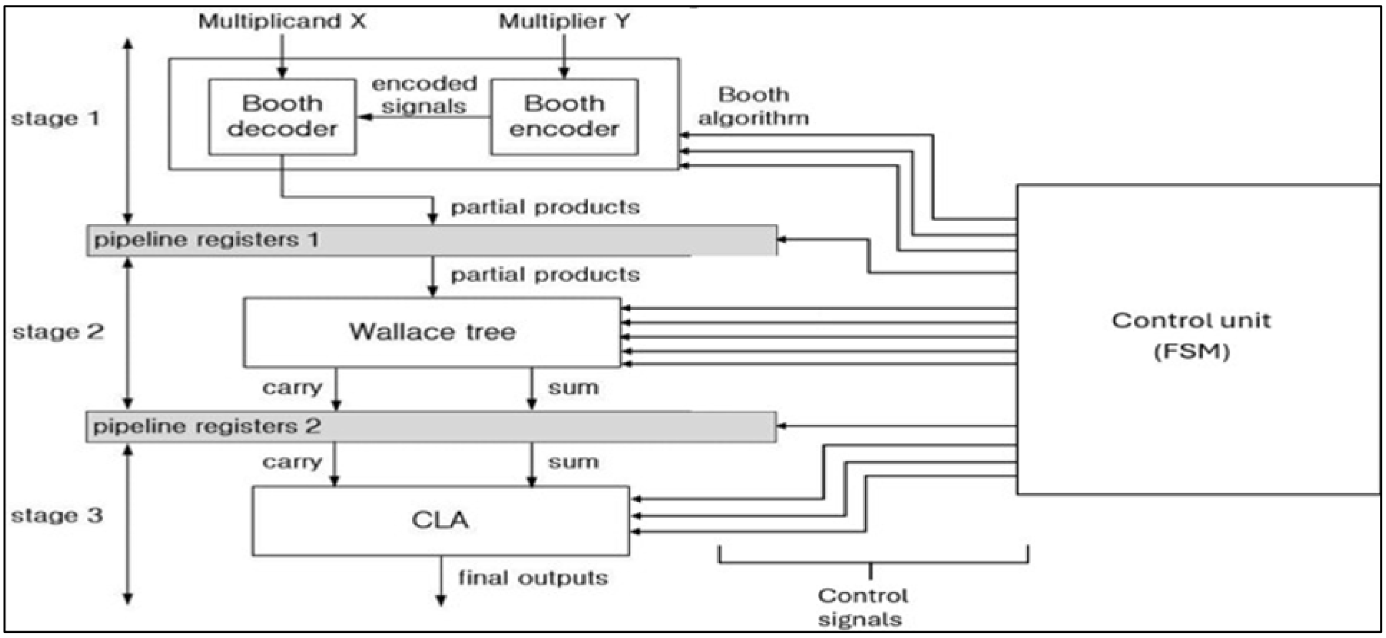


Fig 3 Modified Booth Algorithm Block

- *Radix-4 Booth Encoding:*

Group bits in sets of three to reduce partial products, with each group encoding a multiplier factor (0, ±1, ±2).

- *Partial Product Generation:*

Generating partial products by shifting and adjusting the multiplicand based on Booth encoding.

- *Wallace Tree Reduction:*
  Use of Wallace tree to parallelly reduce partial products to a minimum number of rows. Each stage reduces the number of rows of bits by grouping three bits at the same position into a sum and carry.The process repeats until only two rows of bits remain, representing the final sum and carry.

- *Pipeline Stages:*
  Insert pipeline stages to enhance throughput, allowing multiple operations simultaneously.

- *Final Addition:*
  Sum the final two rows with a fast adder to obtain the final product.

- *FSM:*
  In a MOORE FSM outputs are determined only on the current state.The output remains constant as long the FSM stasys in the same state.

IV. ALGORITHM

➢ *Booth's Multiplication Algorithm*

Booth's algorithm is used to minimize the number of partial products by encoding the multiplier.

➢ *Radix-4 Booth Encoding*

Instead of checking every single bit, radix-4 considers 2 bits at a time and uses a third bit (previous LSB) to form a 3-bit group. The groups map to operations like 0, ±1, or ±2 times the multiplicand. **Encoding Table**

Group (Mult[2i+1:2i-1])Operation 000, 111 0

001, 010+1 * M

011+2 * M

100-2 * M

101, 110-1 * M

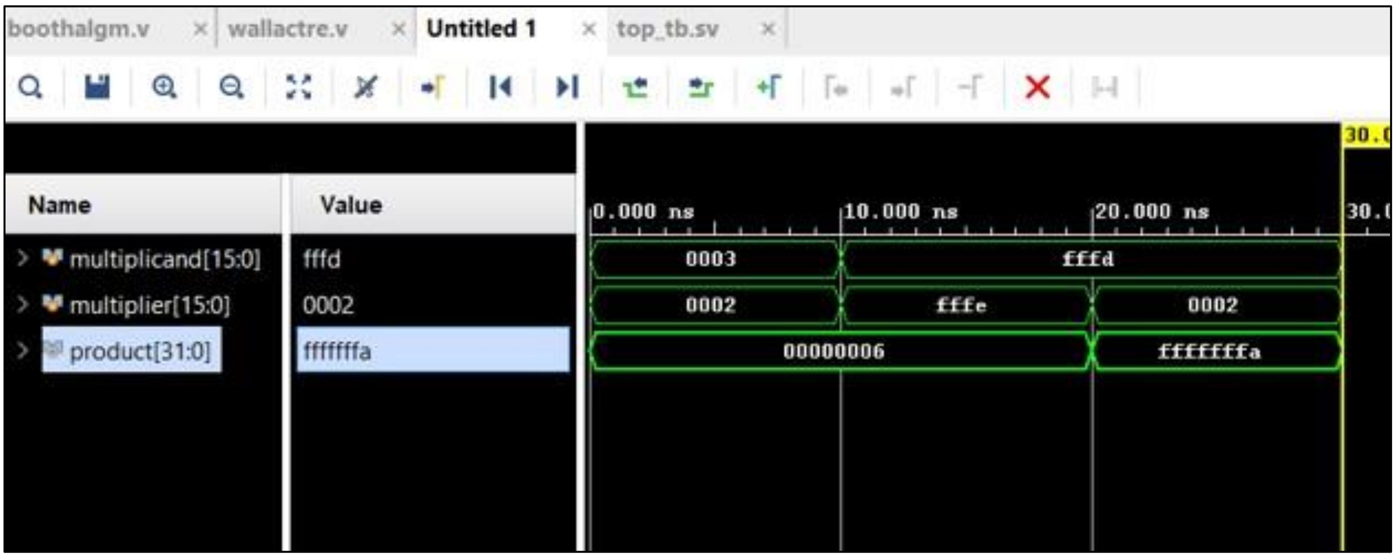
This reduces the number of partial products, improving performance.

➢ *Wallace Tree Structure***:**

Wallace Tree is a hardware- optimized method for summing multiple partial products in a reduced number of steps.

Stages:

✓ Uses full adders (3:2 compressors) and half adders (2:2 compressors).
✓ Operates in parallel to reduce delay.
✓ Results in only two rows: sum and carry. These are then passed to the final adder stage



Name	Value			
multiplicand[15:0]	fffd	0003	fffd	
multiplier[15:0]	0002	0002	fffe	0002
product[31:0]	fffffffa	00000006		fffffffa

Fig 4 Simulation of Modified 16-bit Radix-4 Booth Multiplier

➢ *Carry Lookahead Adder (CLA):*

CLA eliminates the ripple effect of carries by computing them in parallel. Principle:

- Generate (G) = A & B
- Propagate (P) = A ^ B
- Carry[i+1] = G[i] + P[i]*Carry[i]

This technique ensures fast summation of final sum and carry rows.

➢ *System Architecture*

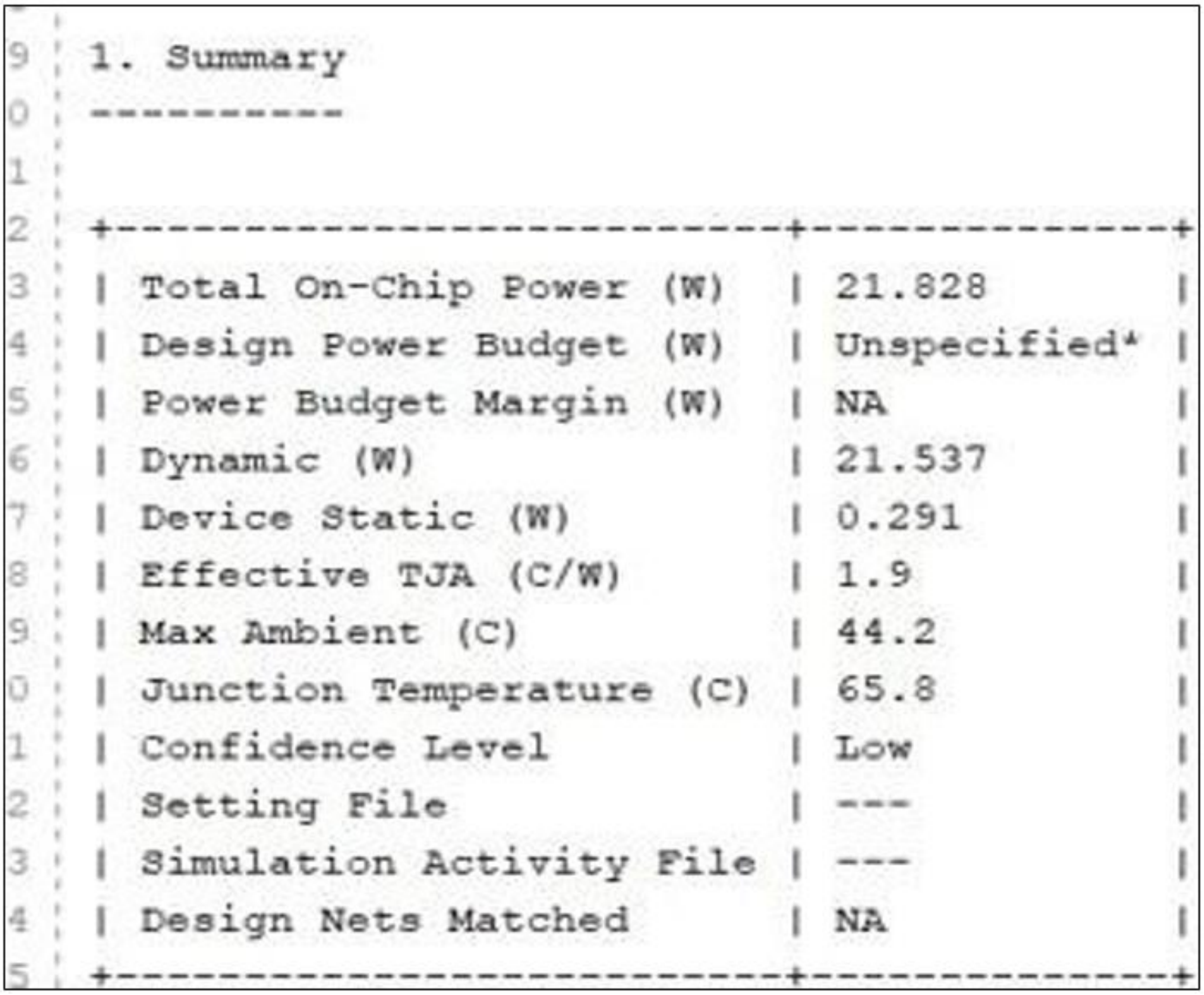
- Booth Encoder: Converts the multiplier to partial product operations.
- Partial Product Generator: Shifts and signs multiplicand based on Booth output.
- Wallace Tree: Compresses all partial products.
- CLA: Performs final summation.

Each module is pipelined for performance.

➤ *Verilog RTL Design:*
Includes detailed Verilog modules:

- boothalgm.v
- wallactreeee.v
- fastadderrr.v
- datapath0001.v
- booth_wallace_cla_tb.v

These are verified via simulation.



```
1. Summary
----------

+--------------------------+--------------+
| Total On-Chip Power (W)  | 21.828       |
| Design Power Budget (W)  | Unspecified* |
| Power Budget Margin (W)  | NA           |
| Dynamic (W)              | 21.537       |
| Device Static (W)        | 0.291        |
| Effective TJA (C/W)      | 1.9          |
| Max Ambient (C)          | 44.2         |
| Junction Temperature (C) | 65.8         |
| Confidence Level         | Low          |
| Setting File             | ---          |
| Simulation Activity File | ---          |
| Design Nets Matched      | NA           |
+--------------------------+--------------+
```

Fig 5 Dynamic and Static Energy

- We reduced dynamic and static energy consumption, with energy reductions ranging from 69% to 78%

➤ *Testbench and Simulation*

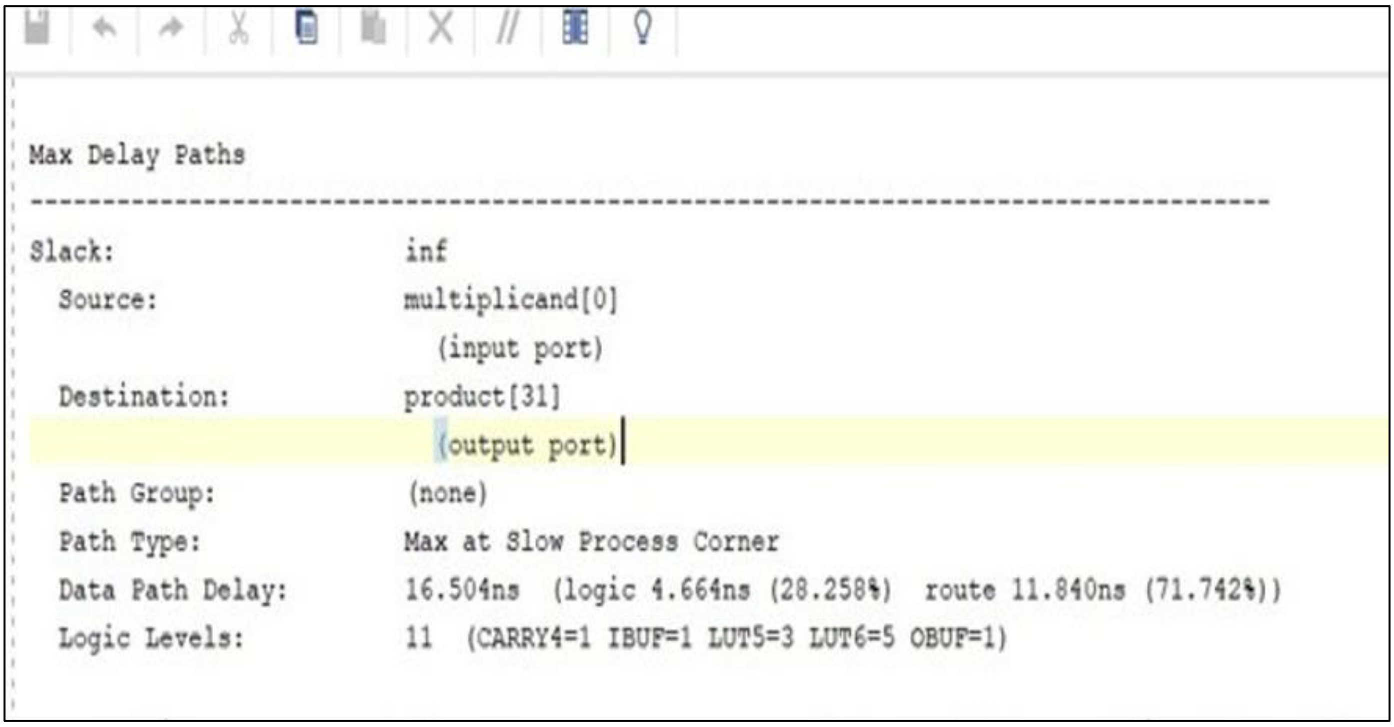
- *The Testbench Simulates Various Combinations:*

✓ Positive * Positive
✓ Negative * Positive
✓ Negative * Negative
✓ Edge cases (0, max/min values) Simulation confirms correctness and speed.

V. RESULTS AND DISCUSSION

➤ *Expected Outcome*

- By applying pipelining technique we can observe increase in the speed of operation.
- Minimize dynamic power consumption.
- Analyse the number of Logic elementsthat would be required.
- Displaying the overall simulation of modified 16-bit Radix-4 booth multiplier using Vivado IDE



```
Max Delay Paths
--------------------------------------------------------------------------------
Slack:                  inf
  Source:               multiplicand[0]
                          (input port)
  Destination:          product[31]
                          (output port)
  Path Group:           (none)
  Path Type:            Max at Slow Process Corner
  Data Path Delay:      16.504ns  (logic 4.664ns (28.258%)  route 11.840ns (71.742%))
  Logic Levels:         11  (CARRY4=1 IBUF=1 LUT5=3 LUT6=5 OBUF=1)
```

Fig 6 Data Path Delay

- We obtained data path delay 16.504ns.

VI. CONCLUSION

This project successfully demonstrates a fast, pipelined 16- bit signed multiplier using Booth encoding, Wallace tree, and CLA. The design is modular, extensible, and performs well in simulation. It can be integrated into larger digital processing units.This project successfully demonstrates the design and implementation of a high-speed, low-power 16- bit signed multiplier using Modified Booth Encoding (Radix-4), Wallace Tree reduction, and a Carry Lookahead Adder (CLA) architecture. By leveraging pipelining and hardware-efficient techniques, the proposed system achieves significant improvements in speed, power efficiency, and scalability.

Simulation results validate the correctness and performance of the design, showcasing an effective reduction in dynamic and static power consumption—

Achieving energy savings up to 78%. The max delay observed was 16.504 ns, confirming the high-speed operation of the multiplier.

The modular nature of the design allows for easy scalability and integration into larger digital systems, such as image processing units, embedded AI accelerators, and real-time video processing systems. These characteristics make the proposed multiplier architecture highly suitable for modern VLSI and low-power computing environments. In summary, the implemented system meets its objectives by optimizing multiplication operations through advanced encoding and hardware design strategies, offering a balanced trade-off between speed, power, and area.

REFERENCES

[1]. Booth, A. D. (1951). “A signed binary multiplication technique.” Quarterly Journal of Mechanics and Applied Mathematics.
[2]. Wallace, C. S. (1964). “A Suggestion for a Fast Multiplier.” IEEE Transactions on Electronic Computers.
[3]. Mano, M. M., & Ciletti, M. D. (2017). Digital Design: With an Introduction to the Verilog HDL.
[4]. Weste, N. H. E., & Harris, D. M. (2010). CMOS VLSI
[5]. Design: A Circuits and Systems Perspective.
[6]. Patterson, D. A., & Hennessy, J. L. (2017). Computer Organization and Design RISC-V Edition.
[7]. Verilog IEEE Standard 1364-2005.
[8]. Rabaey, J. M. (2003). Digital Integrated Circuits: A Design Perspective.
[9]. Bhasker, J. (2003). A Verilog HDL Primer.
[10]. Dandamudi, S. (2005). Fundamentals of Computer Organization and Design.
[11]. IEEE Xplore digital library research articles on multiplier architectures.