Nalanda Library

Aryan Arora

Similarity report

Amrita

Indian Institute of Technology Ropar

Document Details

Submission ID
trn:oid:::1:3427766399

Submission Date
Nov 30, 2025, 9:33 AM GMT+5:30

Download Date
Nov 30, 2025, 9:36 AM GMT+5:30

File Name
Booth_Multiplier_1161_1349_1370.pdf

File Size
1.6 MB

6 Pages

2,224 Words

11,622 Characters

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

0 AI-generated only 0%
Likely AI-generated text from a large-language model.

0 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (i.e., our AI models may produce either false positive results or false negative results), so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

Implementation of 8×8 Modified Low Power Booth Encoded Multiplier Using Kogge–Stone Adder

Aryan Arora (2022EEB1161), Srijan Kumar Kar (2022EEB1349), Arjun Rana (2022EEB1370)
Department of Electrical Engineering IIT Ropar

***Abstract*—This project implements an 8×8 Booth encoded multiplier using a Kogge–Stone adder in Cadence Virtuoso. The design includes schematic creation, layout implementation, functional verification. The Booth algorithm and Modified Radix-4 Booth encoding are used to generate efficient partial products which are summed using fast parallel-prefix adders.**

***Index Terms*—Modified Booth Algorithm, Booth Decoder, Booth Encoder, Kogge–Stone Adder, Cadence Virtuoso**

I. Introduction

Multiplication is an essential arithmetic operation in microprocessors and digital signal processors. In this project, we implement an 8×8 Booth Encoded Multiplier using Kogge–Stone adder as the final summation stage (*which is a novel improvement over the conventional adder in the reference paper*). All building blocks—gates, encoders, decoders, and adders—were designed from transistor level using gpdk180 library.

The multiplier takes two *8-bit signed numbers* in **two's complement format** and produces a *16-bit output*.

II. Theory

A. Number Representation

Binary numbers assign positional weights of 2^n to each bit. In 2's complement representation used for signed numbers,MSB is assigned a negative weight of -2^{n-1} (for an n-bit number). This negative weighting of the MSB helps represent both positive and negative integers, while arithmetic operation can still be performed using standard binary arithmetic rules without extra circuitry for sign handling.

B. Booth's Algorithm

Booth's algorithm efficiently multiplies two signed two's complement numbers by examining transitions between adjacent bits of the multiplier. A dummy bit 0 is appended to the LSB.

The algorithm inspects pairs (x_i, x_{i-1}) and performs operations: This reduces the total number of arithmetic operations, especially for long runs of 1s or 0s. Booth's multiplication algorithm reduces the number of addition/subtraction operations by examining two bits of the multiplier at a time (multiplier and an appended dummy 0). Based on pairs 00, 01, 10, or 11, the algorithm either adds, subtracts, or performs no operation. This reduces switching activity and speeds computation.

Input	Operation
00	No operation
01	Add multiplicand
10	Subtract multiplicand
11	No operation

TABLE I
Input to Operation Mapping

Steps:

- Add dummy 0 to multiplier LSB.
- Inspect two bits at a time: 00 (NOP), 01 (add), 10 (subtract), 11 (NOP).
- Shift multiplicand left after each step.
- Continue until multiplier bits exhausted.

C. Modified Radix-4 Booth Encoding ***(Our Novel Innovation for 8 bit Multiplication)***

Radix-4 Booth multiplication further reduces the number of addition/subtraction operations by grouping the multiplier bits into **sets of three**: (P_2, P_1, P_0). This allows the algorithm to shift the multiplicand by 2 bits at every step (equivalent to multiplying by 4), **effectively halving the number of partial products**.

However, standard Booth encoding cannot directly support radix-4 for all bit patterns due to two corner cases:
1. Multiplier pattern **00011** – shifting by 2 causes the algorithm to skip the required +Y addition.
2. Multiplier pattern **11100** – shifting by 2 would skip the required -Y subtraction.

To solve this, Modified Booth encoding redefines the operations for patterns 011 and 100. Instead of a no output followed by ±Y, it performs a single ±2Y operation.
The complete radix-4 decision table becomes:

The full algorithm is:

1) Append a dummy zero at LSB of multiplier.
2) Initialize partial product = 0.
3) Inspect the lowest 3 bits and select the operation from the table above.
4) Add/subtract 0, Y, 2Y, -Y, or -2Y accordingly.
5) Shift multiplicand left by 2 (= multiply by 4).
6) Shift multiplier right by 2.
7) Repeat until all bits are processed.

Unlike the implementation in reference research paper

Input	Output
000	0
001	+Y
010	+Y
011	+2Y
100	-2Y
101	-Y
110	-Y
111	0

TABLE II
INPUT TO OUTPUT MAPPING

which is limited to a 4×4 multiplier, our design extends the implementation to an 8×8 multiplication architecture. Using this method, the number of additions/subtractions for 8×8 multiplication reduces to at most 4, compared to higher counts in Booth radix-2 of reference paper.

III. Block Diagram

The 8×8 Booth multiplier architecture processes the multiplier bits in overlapping triplets to generate four partial products. These components interact as follows:

- The multiplier $X[7:0]$ is divided into four 3-bit groups: $(X_1, X_0, 0)$, (X_3, X_2, X_1), (X_5, X_4, X_3), and (X_7, X_6, X_5).
- Each group is encoded by a Booth encoder, producing signals S (select Y), D (select $2Y$), and N (negation control).
- A Booth decoder uses S, D, and N to choose one of five values: 0, Y, $2Y$, $-Y$, or $-2Y$.
- The outputs of the four decoders form partial products of width 9 bits.
- These partial products are shifted according to their significance and summed via 16-bit adders.

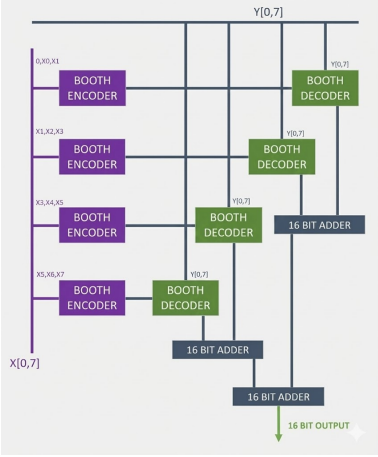

Fig. 1. Block Diagram of Booth Multiplier

This structured pipeline ensures efficient multiplication with reduced partial product count due to Radix-4 Booth encoding.

IV. Basic Gates Used

Four logic gates were implemented using MOSFETs from gpdk180:

- NOT gate
- NAND gate
- NOR gate
- XOR gate

Circuit schematics and layouts were created for each gate.

V. Booth Encoder

The Booth encoder processes a 3-bit group (P_2, P_1, P_0) from the multiplier and converts it into three control outputs: S, D, and N. These determine both magnitude and sign of the selected partial product.

- S selects whether the multiplicand Y is used.
- D selects whether $2Y$ (left shift by 1) is used.
- N determines the sign: 0 for positive, 1 for negative.

This reduces multiple multiplier bits into a single encoded operation. The full truth table describes all eight input combinations and their corresponding encoded operations.

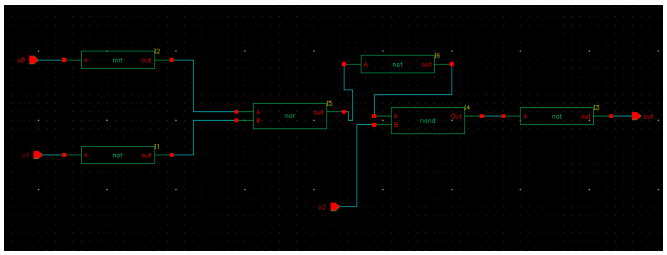

Fig. 2. Schematic for generation of N.

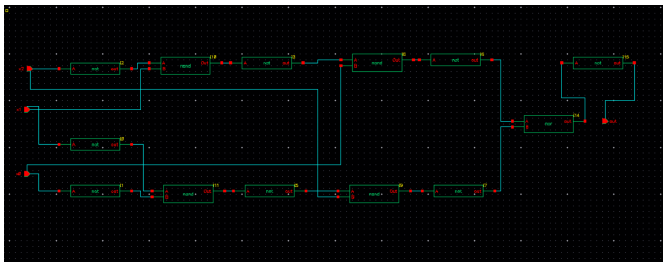

Fig. 3. Schematic for generation of D

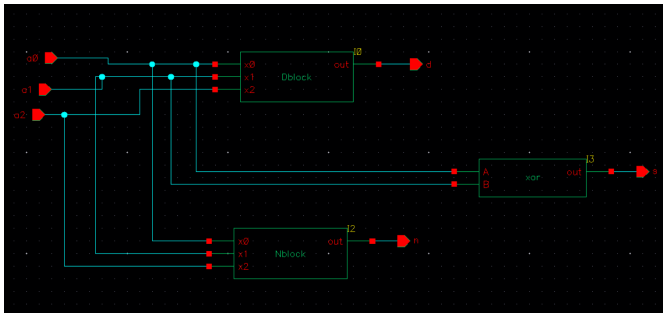

Fig. 4. Schematic of 1 encoder giving D,S,N.

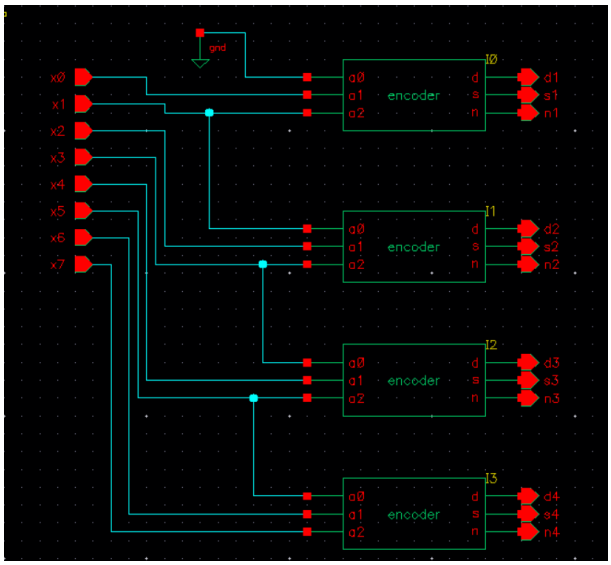

Fig. 5. Schematic of encoder block with 4 encoders.

VI. Booth Decoder

The Booth decoder receives the control signals S, D, and N from the Booth encoder and *produces the corresponding partial product value*. The decoder implements two orthogonal functions: magnitude selection (0, Y, 2Y) and sign selection (positive or negative).

Magnitude generation:

- If S = 1 and D = 0,then output = Y (pass-through of the multiplicand bits).
- If D = 1, then output = 2Y (left shift multiplicand by 1; LSB = 0).
- If S = 0 and D = 0 → output = 0.

Sign generation: If the N flag indicates a negative partial product (N=1), the decoder must produce the two's complement of the generated magnitude. The two's complement process is implemented in two stages:

- 1's complement: bitwise XOR magnitude with logic 1.
- Add 1 using a ***9-bit Kogge–Stone adder*** *which is discussed in upcoming section in great detail.* It is configured to add a single-bit input of value 1 at the appropriate LSB position for the 9-bit vector.

Implementation notes:

- The multiplicand Y is distributed to all decoder blocks to allow parallel generation of each partial product.
- Each decoder outputs a 9-bit vector (8 bits of magnitude *plus an extra sign/extension bit*) to facilitate subsequent addition and sign extension.
- Careful sign-extension and alignment are necessary when shifting partial products for final summation.

The decoder selects between Y, $2Y$, $-Y$, $-2Y$ using S, D, N. Two operations:

- Multiplication by 1 or 2 using shift logic.
- Negation using 2's complement (1's complement + 1).

The 1's complement is generated using XOR with constant 1. The 2's complement is generated using the 9-bit KS adder.

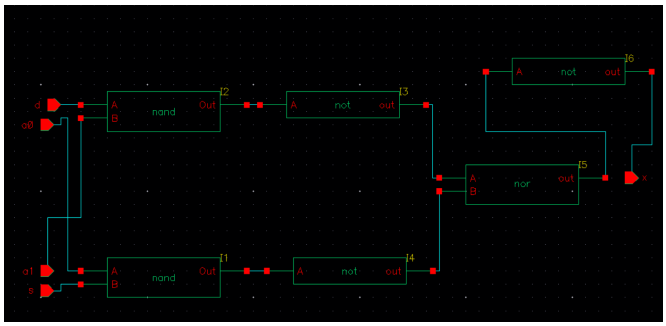

Fig. 6. Schematic of the Initial Block of Decoder.

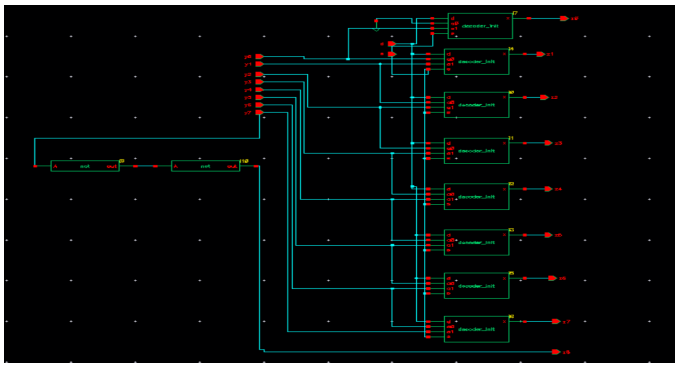

Fig. 7. Layout of bit decoder.

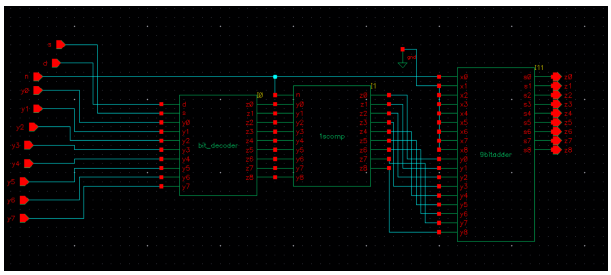

Fig. 8. Schematic of the completed Decoder.

VII. Kogge–Stone Adder

A. Preprocessing

Generate (G) and Propagate (P):

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

B. Carry Generation

$$PP_{i:j} = P_{i:k+1}\, P_{k:j}$$
$$GG_{i:j} = G_{i:k} + P_{i:k+1} G_{k:j}$$

C. Sum Generation

$$C_{i-1} = G_i + P_i C_{in}$$
$$S_i = P_i \oplus C_{i-1}$$

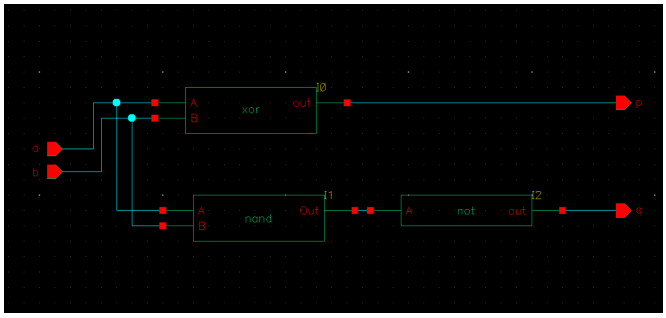

Fig. 9. Schematic of Preprocessing Stage.

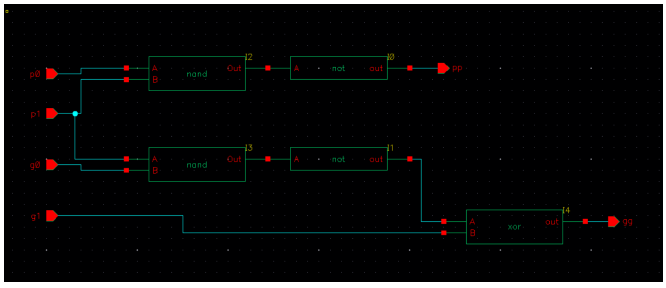

Fig. 10. Schematic of Carry Generating Stage.

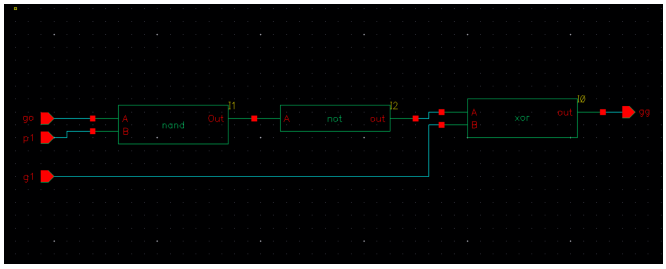

Fig. 11. Schematic of Post processing Stage.

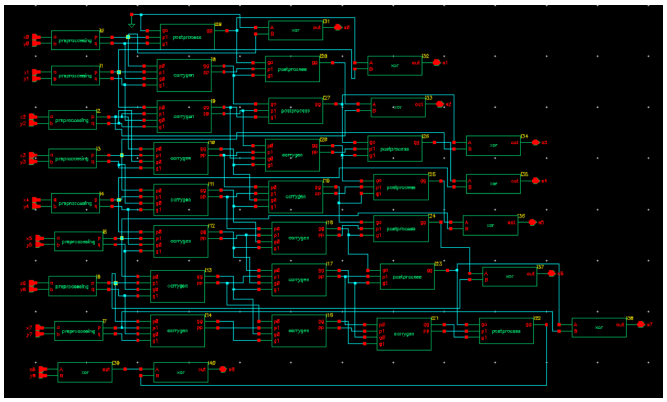

Fig. 12. Schematic of the 9-bit Kogge–Stone adder.

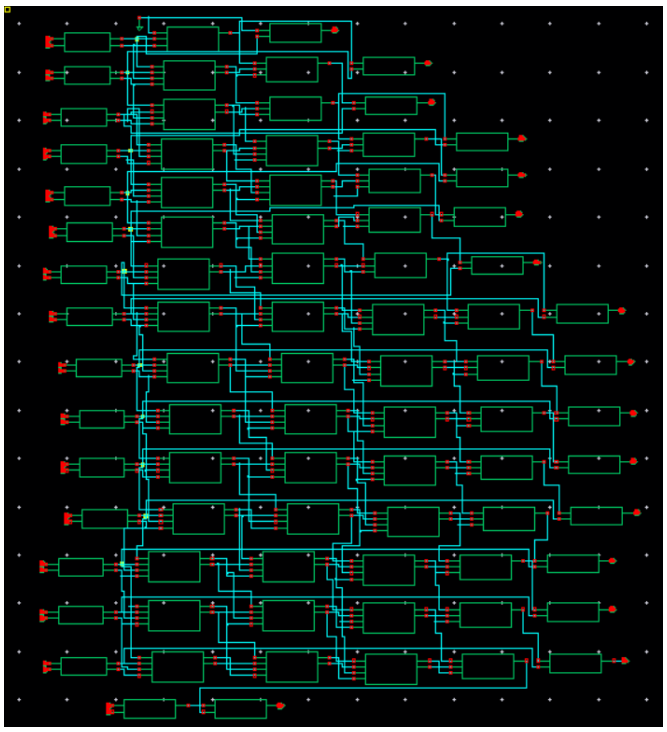

Fig. 13. Schematic of the 16-bit Kogge–Stone adder.

VIII. Final Multiplier Design

After decoding, the four 9-bit partial-product vectors p, q, r, and v must be aligned and combined. Each partial product corresponds to a particular bit-weight window in the final 16-bit result and therefore must be shifted appropriately before summation.

Alignment example:

- p contributes to bits [0..8]
- q contributes shifted by 2: bits [2..10]
- r contributes shifted by 4: bits [4..12]
- v contributes shifted by 6: bits [6..14]

Addition Sequence:

1) Use a 16-bit adder to compute $S_1 = p + (q \ll 2)$.

2) Use a 16-bit adder to compute $S_2 = (r \ll 4) + (v \ll 6)$.
3) Final 16-bit adder computes P = S1 + S2.

Bit-extension and sign handling: Partial products may be negative (two's complement) and require proper sign-extension to 16 bits before addition. The decoder outputs 9-bit two's-complement values; these should be extended to 16 bits by replicating the MSB (sign bit) across the high-order bits of the 16-bit adder inputs.

Verification: Functional verification involves verifying each block and then the integrated design with vectors covering corner cases:

- Positive × Positive
- Positive × Negative
- Negative × Positive
- Negative × Negative
- Run-length corner cases that stress Booth encoding (e.g., 00011111, 11100000)

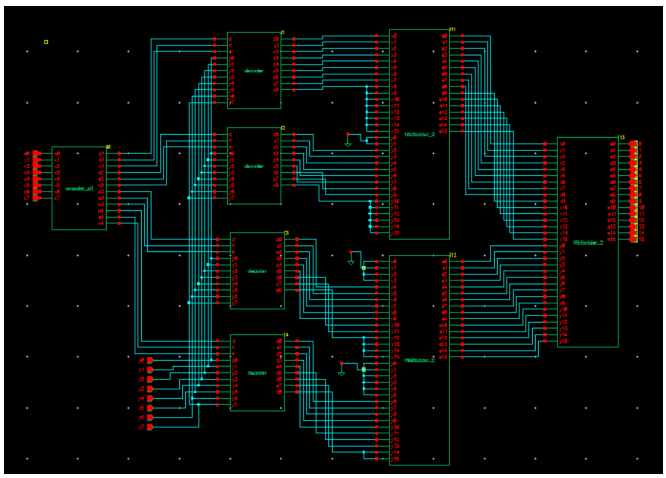

Fig. 14. Schematic of the final booth multiplier.

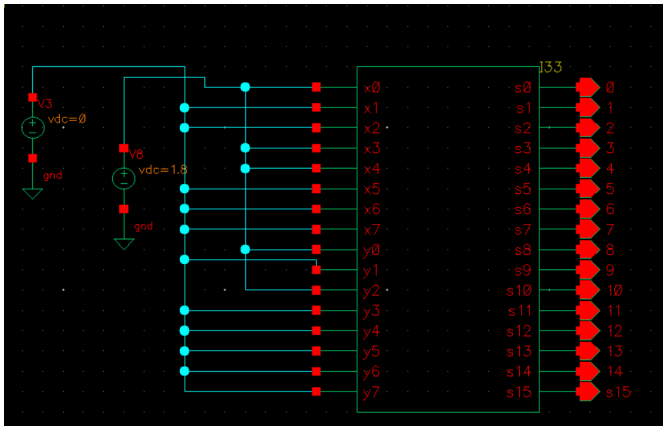

Fig. 15. Schematic of testing the given input.

The Booth decoder produces four 9-bit partial products: p, q, r, v. These are aligned and added using two 16-bit adders, and their result added by another 16-bit adder to produce the final 16-bit result $P[15:0]$.
Test case:

- $X = 00011001_2 = 25$
- $Y = 00000101_2 = 5$
- Output $= 0000000001111101_2 = 125$ (correct)

IX. Simulation Results

Simulation was performed at multiple abstraction levels:

- Gate-level functional verification for each logic gate (NOT, NAND, NOR, XOR).
- Block-level schematic simulation for encoders, decoders, and adders.
- Full-chip transient simulation for the integrated multiplier.

Representative outcomes:
Note: Output Logic 1 is shown by approximately 1.79999999V and Logic 0 is represented as approximately 4.716170nV with a ripple in picovolts.

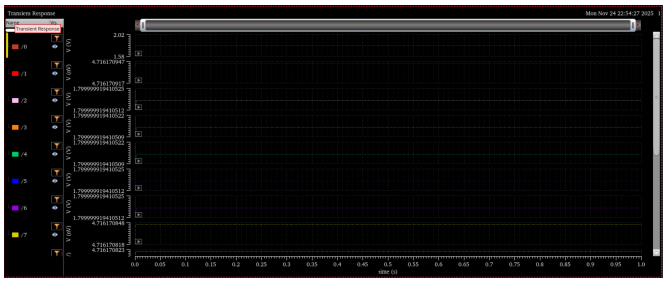

Fig. 16. Results of multiplication (1st 8 bits of 16 bit output).

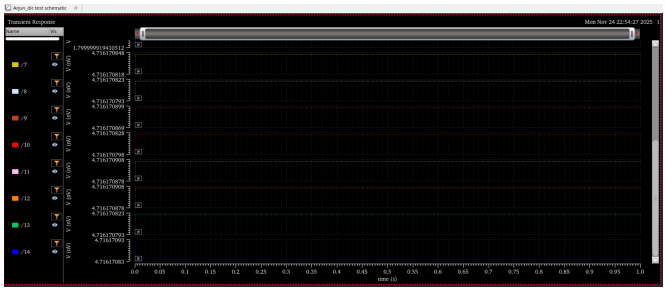

Fig. 17. Results of multiplication (Last 8 bits of 16 bit output).

- Test vector: $X = 00011001_2 = 25_{10}$& $Y = 00000101_2 = 5_{10}$, Output $= 0000000001111101_2 = 125_{10}$.
- Timing waveforms show correct sequencing of Booth encoding, partial product generation, and final addition.
- Observed small output ripple due to wiring capacitance. The ripple has magnitude in pico-volts & does not affect logic correctness.

- Kogge–Stone adders provided fast carry propagation and enabled the final adder tree to meet timing constraints for the chosen technology node.

Power Consumption Results:

Average Power in mW for Different 4-bit Multiplier and Multiplicand Inputs

Input (4-bit Multiplier × Multiplicand)	Booth Multiplier in Paper	Our Modified Booth Multiplier
00001111 × 00001101	0.431	0.211
00000101 × 00001101	0.407	0.198
00000101 × 00001010	0.417	0.197

As the reference paper was based on 4X4 bit multiplication so to test modified 8X8 bit multiplier, we have taken first 4 MSBs as zero (i.e. 0000) for fair comparision.

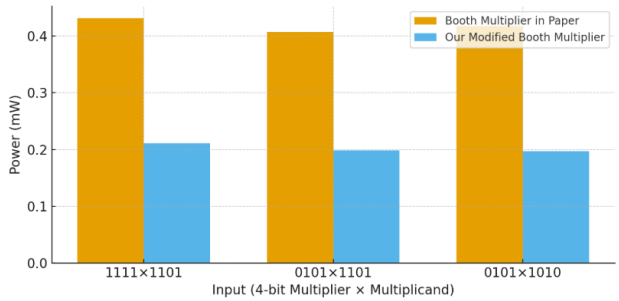

Fig. 18. Test case waveform

X. SUMMARY OF MODIFICATIONS & INNOVATIONS

- We made a 8X8 bit adder instead of 4X4 bit adder represented in reference paper supporting larger numeric range needed for practical embedded designs, DSP kits,etc.
- We replaced the conventional 16 bit adder which no doublt requires lesser no of gates (and hence lasser layout area) but is *slower and consumes more power* as compared to our propsed Kogge-Stone 16 bit adder. Our circuit consumers less than 50% of power on average as compared to that booth multiplier in refrence research paper.
- We extended the implementation of this multiplier to *signed numbers.* In the reference paper by A.S.Prabhu & V.Elakya, the multiplication is restriced to positive pair of numbers but our Modified algorithm is capable of multiplying positive as well as negative pair of numbers *(i.e. + with +,+ with -,- with + & - with -)*

XI. SCOPE FOR IMPROVEMENT

- The number of MOSFETs in the basic logic gates can be reduced. For example,using XOR/XNOR Topology, the encoder can be redesigned to use fewer transistors saving further power and enhancing speed.
- A small ripple has been observed at all the 16 bits of output of the testcase, in nano-volt range. This is mainly due to wiring capacitances, which could be reduced with more time dedicated towards these effects.
- This Modifies Booth Multiplier is a relatively large circuit, and some internal noise will obviously be generated within it. With more time, the design could be improved to achieve little higher noise immunity.

XII. REFERENCES

- Tushar V.More, Dr.R.V.Kshirsagar, “Design of Low Power Column Bypass Multiplier using FPGA” IEEE journal of solid-state, circuits, vol 31,pp 1535-1546,July 2011.
- Rao P.V and Cyril Prasanna Raj and S. Ravi, “VLSI Design and Analysis of Multipliers for Low Power”, Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2009.
- Ohban.J, V.G. Moshnyaga, and K. Inoue, “Multiplier energy reduction through bypassing of partial products,” Asia-Pacific Conf.on Circuits and Systems. vol.2, pp. 13-17, 2002.
- CMOS VLSI Design – Neil H.E. Weste, David Harris
- Fundamentals Of Digital Logic With Verilog Design – Stephen Brown, Zvonko Vranesic
- Wikipedia articles on multiplier, Booth algorithm, logic gates