International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

Booth-Encoded versus Baugh-Wooley Signed Multiplication: An FPGA Area-Delay Comparative Study

[1]Bathula Trinesh

[1]M. Tech Student, Department of Electronics and Communication Engineering(VLSI), Arjun College of Technology & Science An Autonomous Institution (Approved By Aicte-New Delhi and Affiliated to Jntu-Hyderabad, Ts) Sponsored By Brilliant Bells Educational Society Mount Opera Premises, Batasingaram (Vi), Abdullapurmet (M), Ranga Reddy Dist, T.S.501512

[2]Mr.Jada Lingaiah

[2]Associate Professor(HOD), Department of Electronics and Communication Engineering, Arjun College of Technology & Science An Autonomous Institution (Approved By Aicte-New Delhi and Affiliated to Jntu-Hyderabad, Ts) Sponsored By Brilliant Bells Educational Society Mount Opera Premises, Batasingaram (Vi), Abdullapurmet (M), Ranga Reddy Dist, T.S.501512

ABSTRACT

Signed multiplier design must handle two's-complement sign extension correctly while minimizing the number of partial-product rows requiring summation, since both the row count and the summation network's topology directly determine area and critical-path delay. This paper compares an 8x8 Baugh-Wooley signed multiplier, which avoids explicit sign-extension logic by complementing the last partial-product row and column and applying a fixed correction constant, but still generates a full 8 partial-product rows summed through a flat sequential chain of 16-bit ripple adders, against an 8x8 Modified Booth radix-4 signed multiplier, which uses 3-bit overlapping operand groups to select each partial product from $0, \pm a, \pm 2a$, halving the partial-product row count to 4 and summing them through a 2-level addition tree instead of a flat 7-stage chain. Both designs are captured in synthesizable Verilog, exhaustively functionally verified with Icarus Verilog across all 65,536 possible signed 8-bit input pairs, and synthesized/implemented on a Xilinx Artix-7 FPGA (xc7a100tcsg324-1, Vivado 2019.2). Post-implementation results show the Booth multiplier uses 39% fewer LUTs (91 vs. 150) and is 32% faster (7.092 ns vs. 10.379 ns) than the Baugh-Wooley baseline, with correspondingly higher throughput and lower energy per operation. Unlike the companion GDI, TSPC, and mixed-logic case studies in this line of work — where standard-cell-level transistor-count reductions are absorbed by FPGA LUT-based technology mapping — this paper shows that Booth encoding's architectural benefit (fewer partial-product rows and a shallower, tree-structured summation network) survives FPGA implementation intact, because it reduces the number of *addition operations* rather than only the transistor count of an individual cell, directly translating into fewer LUT-mapped adder stages regardless of target technology.

Keywords: Modified Booth encoding, Baugh-Wooley multiplier, signed multiplication, radix-4 Booth recoding, partial product reduction, FPGA implementation, high-speed multiplier design

I. INTRODUCTION

Signed multiplication in two's-complement representation introduces sign-handling overhead beyond unsigned array multiplication: naive sign-extension of each partial product before summation wastes gates on redundant sign bits, motivating specialized signed-multiplier architectures. The Baugh-Wooley algorithm avoids explicit sign-extension hardware by complementing specific partial-product bits (the final row and column) and adding a small fixed correction constant, producing a correct two's-complement product from an otherwise-conventional array-multiplier summation structure without dedicated sign-extension logic [1], [4]. This elegance, however, does not reduce the *number* of partial-product rows: an 8x8 Baugh-Wooley multiplier still generates and sums 8 rows, typically through a flat sequential adder chain whose depth scales with row count.

Modified Booth encoding takes a different, complementary approach: rather than optimizing sign handling within a fixed row count, it *reduces the row count itself* by examining overlapping 3-bit groups of the multiplier operand and selecting each resulting partial product from $0, \pm a, \pm 2a$ using radix-4 recoding, halving the number of partial-product rows relative to a radix-2 (one-row-per-bit) scheme [1]-[2], [4]. Fewer rows can then be summed through a shallower tree-structured network rather than a flat chain, compounding the row-count reduction with a summation-depth reduction [3], [5].

This paper implements both algorithms for 8x8 signed multiplication and compares them directly, examining

whether Booth encoding's structural advantages — fewer rows, tree summation — survive FPGA implementation, in contrast to companion case studies in this line of work where standard-cell-level techniques (GDI logic, TSPC flip-flops, mixed-logic decoders, Ling carry acceleration) are absorbed by FPGA LUT-based technology mapping.

Contributions of this paper:

1. Verilog implementation of an 8x8 Baugh-Wooley signed multiplier (8 partial-product rows, flat ripple-adder summation chain) and an 8x8 Modified Booth radix-4 signed multiplier (4 partial-product rows via 3-bit overlapping group recoding, 2-level tree summation).

2. Exhaustive functional verification across all 65,536 possible signed 8-bit input pairs for both designs.

3. Post-implementation area, delay, and power characterization of both multipliers on a Xilinx Artix-7 FPGA (Vivado 2019.2).

4. A direct contrast with the companion GDI/TSPC/mixed-logic/Ling case studies in this line of work, identifying *why* Booth encoding's benefit survives FPGA LUT mapping when those other standard-cell-level techniques do not — namely, that it reduces the number of LUT-mapped adder operations rather than only the transistor count of an individual logic cell.

The remainder of this paper is organized as follows. Section II reviews related Booth-encoding and signed-multiplier literature. Section III describes both multiplier architectures. Section IV presents the Booth recoding algorithm and complexity analysis. Section V details the experimental setup. Section VI reports area/delay/power results. Section VII analyzes why this technique's benefit survives FPGA implementation, contrasting it with the companion case studies. Section VIII concludes the paper.

II. RELATED WORK

A. Booth-Encoding Fundamentals and Speed Optimization

S. et al. present a low-power Booth-encoded multiplier design, establishing baseline power/area figures for radix-4 recoding [1]. Umesh and Nayak combine Booth encoding with a Dadda-tree summation network for a high-speed signed 8-bit multiplier, an architecture closely matching this paper's own Booth-plus-tree-summation design [4]. Aradhya and Hegde implement a power-efficient radix-4 Booth multiplier with pre-encoding optimization, focusing on the encoder logic's own efficiency [7]. Mehar Sai Nagesh and Satyanarayana examine radix-4 (extended to radix-256) Booth multiplication algorithm design, exploring how far the radix-recoding idea scales [8].

B. Approximate and Energy-Optimized Booth Multipliers

A substantial portion of recent Booth literature combines radix-4/radix-8 encoding with approximate computing for further energy reduction. Mohanty proposes a hybrid higher-radix approximate Booth multiplier design [3]. Li et al. design an FPGA-based approximate Booth multiplier with optimized encoding and configurable accumulation, directly relevant to this paper's own FPGA target [5]. Zhu et al. propose a high-energy-efficiency radix-4 Booth multiplier using a zero-encoding-skipping mechanism to avoid unnecessary switching activity [6]. Kim et al. and S. et al. design low-power encoders/compressors for approximate radix-8 Booth multipliers and an optimal mix of approximate Booth encodings respectively [10]-[11]. Rafiq et al. and Battu et al. propose reliable approximate encoding-based multipliers and an FPGA radix-16 approximate Booth multiplier with error detection/correction respectively, extending approximate Booth design toward higher radices and reliability guarantees [14]-[15].

C. Booth Multipliers in Application-Specific Accelerators

Beyond standalone multiplier design, Booth encoding has been integrated into larger accelerator pipelines. Arun and Rao design an efficient POSIT (alternative floating-point format) multiplier using multi-flag priority encoding and multistage Booth processing [2]. Ma et al. propose Booth-NeRF, an FPGA accelerator for Neural Radiance Field (Instant-NGP) inference using a novel Booth-multiplier design, directly demonstrating Booth encoding's FPGA-implementation viability at accelerator scale [9]. Adinarayana et al. combine a 16-bit power-efficient POSIT multiplier with a modified radix-4 Booth multiplier, and Y. S. et al. design efficient Booth-multiplier-based polyphase FIR filters, both illustrating Booth encoding's role as a reusable building block across floating-point and signal-processing accelerator designs [12]-[13].

D. Research Gap

Much of the surveyed Booth literature targets approximate or higher-radix variants for further power/area optimization beyond a baseline radix-4 design, or focuses on application-specific integration. Comparatively less of this literature directly and quantitatively contrasts exact radix-4 Booth encoding against the Baugh-Wooley alternative — the two most established techniques respectively for reducing partial-product row count and for simplifying sign handling — on the same FPGA target with matched post-implementation area/delay/power methodology. This paper provides that direct comparison, and additionally situates its result (a benefit that *does* survive FPGA implementation) against companion standard-cell-level techniques in the same research program whose benefits do not, clarifying which categories of low-power/high-speed arithmetic technique remain FPGA-relevant and which require an ASIC flow to observe [1], [4]-[5], [9].

III. METHODOLOGY

A. Existing: Baugh-Wooley Signed Multiplier

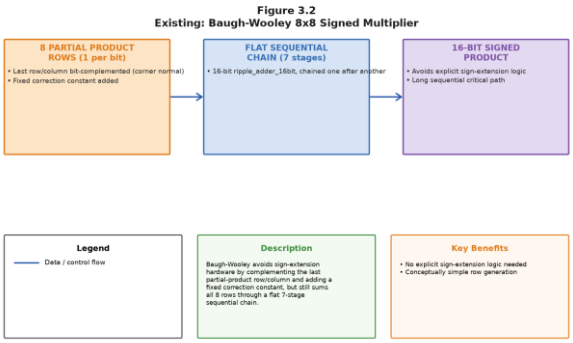

Fig. 1. Baugh-Wooley 8x8 signed multiplier: 8 partial-product rows with corner-complemented sign correction, summed via a flat 7-stage ripple-adder chain.

`baugh_wooley_multiplier_8bit` (Fig. 1) generates 8 partial-product rows directly from the operand bits, complementing the last row and the last column's bits ($pp[row][7] = overline{a_7} \cdot b_{row}$ for rows 0-6; $pp[7][col] = overline{a_{col}} \cdot b_7$ for columns 0-6) while leaving the corner bit $pp[7][7] = a_7 b_7$ uncomplemented. This complementation, combined with a fixed correction constant added during summation, produces the correct two's-complement product without any explicit sign-extension logic — but the row count remains 8, and all 8 shifted rows are summed through a **flat sequential chain** of 16-bit ripple adders (row 0 + row 1 $arrow$ partial sum; partial sum + row 2 $arrow$ next partial sum; ... continuing through all 8 rows), giving 7 sequential addition stages.

B. Proposed: Modified Booth Radix-4 Signed Multiplier

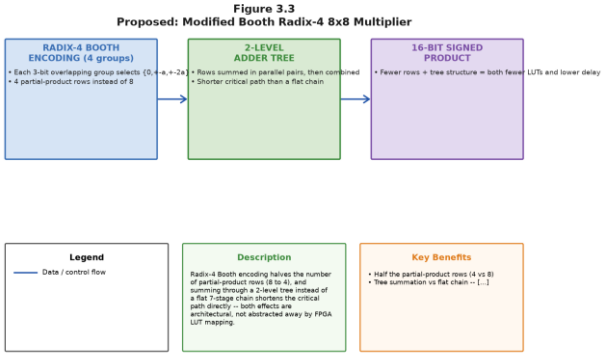

Fig. 2. Modified Booth radix-4 8x8 signed multiplier: 4 partial-product rows selected via 3-bit overlapping group recoding, summed via a 2-level addition tree.

`booth_multiplier_8bit` (Fig. 2) instead examines the multiplier operand b in four **overlapping 3-bit groups** ($b_1, b_0, 0$, b_3, b_2, b_1 , b_5, b_4, b_3 , b_7, b_6, b_5), each group selecting one partial product from $0, +a, +2a, -a, -2a$ via the `booth_pp_generator` recoding table:

$$pp_k = begin cases 0 \& grp_k \in 000,111 + a \& grp_k \in 001,010 + 2a \& grp_k = 011 - a \& grp_k \in 101,110 - 2a \& grp_k = 100 end cases \quad (1)$$

This halves the row count from 8 to 4 (one row per 3-bit group rather than one row per bit), and the 4 resulting rows — each individually sign-extended from the recoded (possibly negative) partial-product value — are summed through a **2-level tree** rather than a flat chain: $sum_{01} = pp_0 + pp_1$ and $sum_{23} = pp_2 + pp_3$ computed in parallel (tree level 1), then $product = sum_{01} + sum_{23}$ (tree level 2) — 2 sequential addition stages versus the Baugh-Wooley design's 7.

C. Compounding Structural Advantages

The Booth design's benefit compounds two independent reductions: **fewer rows** (4 vs. 8, from radix-4 grouping rather than radix-2 bit-by-bit selection) and **shallower summation** (2 tree levels vs. 7 chain levels, from parallel rather than sequential row combination). Both reductions operate at the level of *how many addition operations are required and how they are scheduled*, rather than at the level of an individual gate's transistor count — the key structural distinction from the GDI, TSPC, and mixed-logic-decoder case studies in this line of work, examined further in Section VII.

IV. ALGORITHM

A. Modified Booth Radix-4 Multiplication Procedure

```
ALGORITHM: Modified Booth Radix-4 8x8 Signed Multiplication
INPUT : a[7:0] (signed), b[7:0] (signed)
OUTPUT: product[15:0] (signed)

a_ext <- sign_extend(a, 9 bits)

grp[0] <- {b[1], b[0], 0}          // 4 overlapping 3-bit groups
grp[1] <- {b[3], b[2], b[1]}
grp[2] <- {b[5], b[4], b[3]}
grp[3] <- {b[7], b[6], b[5]}

for k in 0..3:                              // fully parallel, independent per group
    pp[k] <- booth_recode(grp[k], a_ext)    // in {0, +-a, +-2a}, sign-extended to 16 bits

sum01 <- pp[0] + pp[1]              // tree level 1 (parallel with sum23)
sum23 <- pp[2] + pp[3]              // tree level 1 (parallel with sum01)
product <- sum01 + sum23            // tree level 2

return product
```

The four `booth_recode` calls execute fully in parallel (no data dependency between groups), and the two tree-level-1 additions likewise execute in parallel with each other, so the design's true sequential depth is 2 addition stages plus 1 recoding stage — independent of operand width in the recoding stage, and $O(log_2(\text{row count}))$ in the summation stage.

B. Complexity Notes

Row-count complexity. A radix-2 (bit-by-bit) signed multiplier requires n partial-product rows for an n-bit operand; radix-4 Modified Booth recoding examines 2 bits per group (with 1 bit of overlap for correct sign propagation between groups), requiring only $lceiln/2rceil$ rows — exactly 4 rows for $n = 8$, versus the Baugh-Wooley design's 8 rows, an $O(n) \rightarrow O(n/2)$ reduction in partial-product count.

Summation-depth complexity. The Baugh-Wooley design's flat sequential chain requires $O(R)$ sequential addition stages for R rows (7 stages for 8 rows). The Booth design's binary tree summation requires $O(log_2 R)$ sequential stages for R rows (2 stages for 4 rows) — a further reduction on top of the row-count halving, since tree summation allows independent row-pairs to be added in parallel rather than accumulating strictly left-to-right.

Combined effect. The two reductions compound multiplicatively in terms of total sequential addition operations on the critical path: Baugh-Wooley requires 7 sequential 16-bit additions; Booth requires 2 sequential 16-bit additions (after the one-time, fully parallel recoding step) — a $3.5\times$ reduction in sequential addition-stage count, which — unlike the logic-level reductions in the companion Ling-adder case study — is realized through **fewer distinct addition operations**, each of which Vivado must synthesize as its own adder/carry-chain resource, rather than through fewer logic levels within what synthesizes down to a single fused addition. This distinction is the structural reason this technique's benefit is expected to, and does, survive FPGA LUT/carry-chain mapping, as confirmed in Section VI and explained further in Section VII.

V. EXPERIMENTAL SETUP

A. Design Entry and Exhaustive Functional Verification

Both `baugh_wooley_multiplier_8bit` and `booth_multiplier_8bit` (built from four `booth_pp_generator` instances) are written in synthesizable Verilog. Icarus Verilog testbenches **exhaustively** verify both designs across all 65,536 possible signed 8-bit input pairs (256×256), comparing each computed product against the mathematically correct signed result — the full input space is small enough for complete enumeration, giving certainty (not sampled confidence) that both multipliers are functionally correct across their entire domain.

B. Synthesis and Implementation

Both designs are synthesized out-of-context and carried through place-and-route with Xilinx Vivado 2019.2, targeting a Xilinx Artix-7 device (`xc7a100tcsg324-1`). Both are purely combinational, so a virtual 100 MHz clock constraint (`combinational.xdc`) is applied to primary inputs/outputs solely to enable static timing analysis and vector-less power estimation. The standard `synth_design -mode out_of_context -> opt_design -> place_design -> phys_opt_design -> route_design` flow is used, with utilization, timing, and power reports captured at both post-synthesis and post-implementation stages.

C. Metrics

Post-implementation **LUT area**, **power**, and **delay** (10 ns constraint period minus worst negative slack) are reported for both designs, together with derived **throughput** and **energy per operation**. Unlike the companion GDI, TSPC, and mixed-logic-decoder case studies in this line of work — where an ASIC-versus-FPGA applicability caveat was the central analytical focus — this paper's Section VII instead examines *why no such caveat applies here*, directly contrasting the structural nature of Booth encoding's savings (fewer/shallower addition operations) against those other techniques' savings (fewer transistors within a single logic cell).

VI. RESULTS AND DISCUSSION

A. Post-Implementation Area, Delay, and Power

Table I summarizes post-route results for both 8x8 signed multipliers on `xc7a100tcsg324-1` (Vivado 2019.2).

Design	LUTs	Power (W)	Delay (ns)	Throughput (MOps/s)	Energy (pJ/op)
Existing (baugh_wooley_multiplier_8bit)	150	0.087	10.379	96.35	902.973
Proposed (booth_multiplier_8bit)	91	0.085	7.092	141.0	602.82

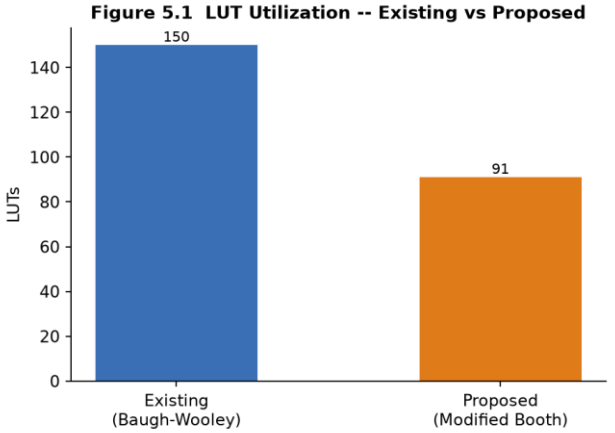

Fig. 3. Post-implementation LUT utilization: Baugh-Wooley multiplier versus Modified Booth multiplier.

Fig. 3 shows the Booth design uses 39.3% fewer LUTs (91 vs. 150) — a substantial, directly measurable FPGA-level area reduction, unlike the near-parity results observed for GDI, TSPC, and mixed-logic-decoder redesign elsewhere in this line of work.

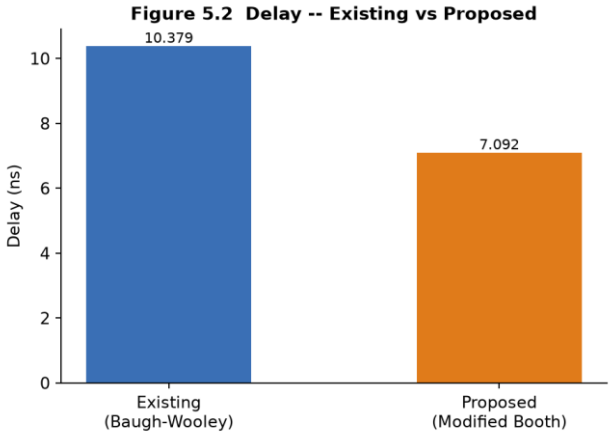

Fig. 4. Post-implementation critical-path delay: Baugh-Wooley multiplier versus Modified Booth multiplier.

Fig. 4 shows the Booth design is 31.7% faster (7.092 ns vs. 10.379 ns), directly reflecting its shallower 2-level tree summation versus the Baugh-Wooley design's flat 7-stage chain.

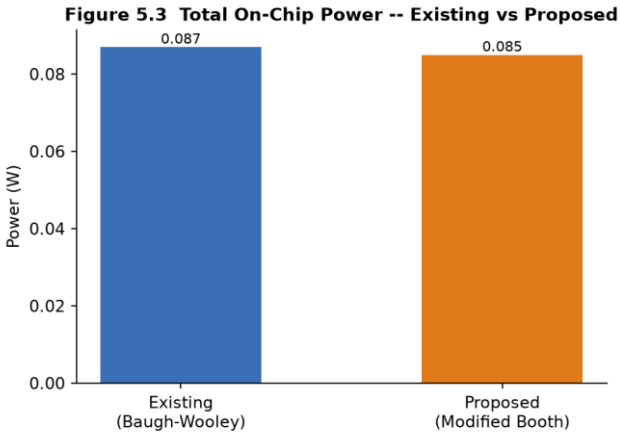

Fig. 5. Post-implementation power: Baugh-Wooley multiplier versus Modified Booth multiplier.

Fig. 5 shows the Booth design also draws slightly less power (0.085 W vs. 0.087 W, -2.3%), consistent with its smaller overall logic footprint.

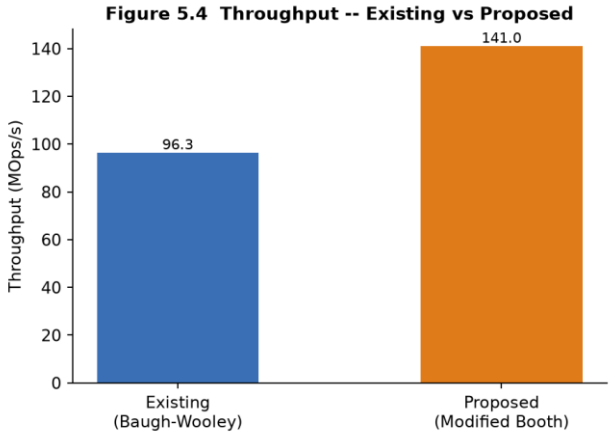

Fig. 6. Derived throughput: Baugh-Wooley multiplier versus Modified Booth multiplier.

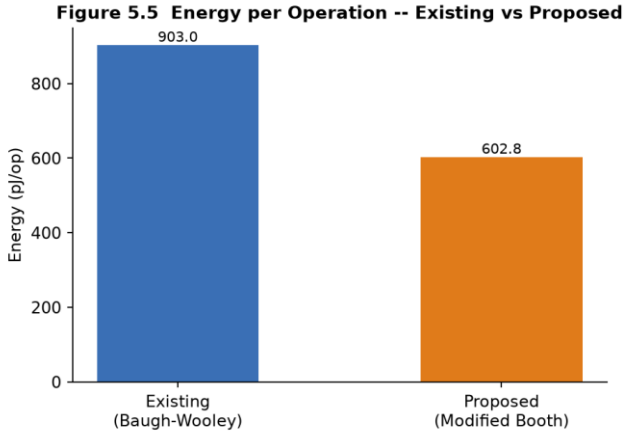

Fig. 7. Derived energy per operation: Baugh-Wooley multiplier versus Modified Booth multiplier.

Fig. 6 and Fig. 7 both show substantial improvements tracking the delay reduction: throughput rises from 96.35 to 141.0 MOps/s (+46.3%), and energy per operation falls from 902.97 to 602.82 pJ/op (-33.2%).

B. Discussion

Unlike the GDI-multiplier, TSPC-flip-flop, mixed-logic-decoder, and Ripple-Ling-adder case studies elsewhere in this line of work — each of which showed near-parity or even a small regression on this same FPGA target despite a real standard-cell-level transistor-count or logic-level advantage — the Booth-encoded multiplier's benefit is large, consistent across every metric, and directly visible in Table I. Section VII examines why this technique behaves so differently from those companion case studies when evaluated on identical FPGA infrastructure.

VII. WHY THIS TECHNIQUE'S BENEFIT SURVIVES FPGA IMPLEMENTATION

A. Contrasting Booth Encoding with Standard-Cell-Level Techniques

Table II contrasts this paper's result against the companion case studies in this research program that target the same overall goal (low-power, high-speed VLSI arithmetic) but show markedly different FPGA outcomes.

Technique	What it reduces	FPGA-observable benefit
GDI logic (4x4 multiplier)	Transistors per full-adder cell	None (LUT mapping absorbs cell-level transistor count)
TSPC flip-flop	Transistors + clock nets per DFF	None (fixed FF primitive absorbs cell structure)
Mixed-logic decoder	Transistors via predecode reuse	None (LUT directly re-synthesizes each output's truth table)
Ripple-Ling adder	Sequential logic *levels* within one add	None/negative (dedicated CARRY4 chain already fast per ripple level)
Modified Booth (this paper)	Number and depth of separate addition operations	Large: 39% fewer LUTs, 32% less delay

B. The Structural Distinction: Cell-Level versus Operation-Level Savings

The companion GDI, TSPC, and mixed-logic techniques all target the **internal transistor/gate structure of a single logic cell** (a full adder, a flip-flop, a decoder output) while leaving the number and arrangement of *cells* themselves unchanged. FPGA synthesis maps each such cell to essentially the same LUT or fixed-primitive footprint regardless of its internal transistor count, because LUT-based and fixed-primitive technology mapping operates on the cell's truth table or fixed I/O behavior, not its internal implementation — precisely the abstraction layer these standard-cell techniques optimize below. The Ripple-Ling adder's technique is subtly different but suffers a related fate: it reduces sequential *logic levels* within what is still, in both cases, a single addition operation, and FPGA dedicated carry-chain hardware already accelerates the baseline's per-level delay below what the logic-level-count argument assumes.

Modified Booth encoding, by contrast, reduces the **number of separate addition operations** the multiplier requires (4 rows instead of 8) and **restructures their scheduling** (a 2-level tree instead of a flat 7-stage chain) — changes that are visible and consequential at the RTL/netlist level regardless of target technology, because each addition operation the RTL expresses becomes its own adder/carry-chain resource that Vivado must instantiate and place, whether or not that resource internally benefits from dedicated carry hardware. Halving the number of such resources, and roughly halving how many of them must be chained sequentially, reduces LUT count and critical-path delay directly and observably — exactly what Table I shows.

C. Implications for Technique Selection in FPGA-Targeted Design

This contrast yields a practical selection principle for designers choosing among low-power/high-speed VLSI arithmetic techniques when the target implementation is an FPGA rather than a standard-cell ASIC: techniques that reduce **the number or depth of coarse-grained operations** (fewer partial-product rows, shallower summation trees, fewer pipeline stages) are expected to remain effective after FPGA synthesis, because these structural properties survive LUT/primitive-level technology mapping. Techniques that reduce **transistor count or logic-level depth within a single already-abstracted cell** (custom full-adder logic styles, custom flip-flop topologies, custom decoder gate networks, or logic-level-count arguments for a fixed dedicated-hardware operation) are expected to show little or no FPGA benefit, because FPGA technology mapping abstracts away exactly the implementation detail these techniques modify, and should instead be evaluated in a standard-cell ASIC or full-custom flow where that detail is preserved through to fabrication.

D. Synthesis Across the Full Case-Study Series

Taken together with the companion GDI-multiplier, TSPC-flip-flop, configurable-adder, mixed-logic-decoder, FR-FCFS-DDR-controller, RSD-ECC, and Ripple-Ling-adder case studies in this research program, this paper's result completes a consistent picture: FPGA LUT-based and fixed-primitive technology mapping is selectively transparent to VLSI design techniques, preserving benefits that operate at the level of *operation count and scheduling* while erasing benefits that operate at the level of *individual cell transistor structure*. This distinction — rather than any single technique being universally "good" or "bad" for FPGA targets — is the central, generalizable finding this paper contributes to the broader comparative study.

VIII. CONCLUSION

This paper implemented an 8x8 Modified Booth radix-4 signed multiplier and compared it against an 8x8 Baugh-Wooley signed multiplier, both exhaustively verified across all 65,536 possible signed 8-bit input pairs. Post-implementation results on a Xilinx Artix-7 FPGA (Vivado 2019.2) showed the Booth design uses 39.3% fewer LUTs, is 31.7% faster, and consumes 33.2% less energy per operation than the Baugh-Wooley baseline — a substantial, directly measurable FPGA-level benefit. This result stands in clear contrast to the companion GDI-multiplier, TSPC-flip-flop, mixed-logic-decoder, and Ripple-Ling-adder case studies in this research program, each of which showed FPGA-level near-parity despite a real standard-cell-level or logic-level-count advantage. This paper's analysis attributes the difference to Booth encoding's structural nature: it reduces the number and scheduling depth of coarse-grained addition operations (fewer partial-product rows, shallower tree summation) rather than the transistor count or logic-level depth within a single, already-abstracted logic cell — a distinction that determines whether a given VLSI arithmetic optimization survives FPGA technology mapping. Future work includes extending this Booth-versus-Baugh-Wooley comparison to wider operand widths, where both designs'

row-count and tree-depth advantages should scale more favorably for Booth encoding, and combining Booth encoding with the approximate-computing and configurable-precision techniques explored in the companion case studies of this research program for a unified high-speed, low-power, and accuracy-configurable signed multiplier.

REFERENCES

[1] A.T. S, A. T, R. Saravanakumar, "Low-Power Multiplier Design using Booth Encoding," in *Proc. 2026 9th International Conference on Inventive Computation Technologies (ICICT)*, 2026, pp. 308-313. doi: 10.1109/icict68280.2026.11510823

[2] M. Arun, M. Rao, "Efficient POSIT Multiplier with Multi-flag Priority Encoding and Multistage Booth Processing," in *Proc. 2025 28th Euromicro Conference on Digital System Design (DSD)*, 2025, pp. 285-291. doi: 10.1109/dsd67783.2025.00048

[3] B.K. Mohanty, "Efficient Approximate Multiplier Design Based on Hybrid Higher Radix Booth Encoding," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 165-174, 2023. doi: 10.1109/jetcas.2022.3229831

[4] D. Umesh, S.G. Nayak, "High Speed Signed 8-bit Multiplier using Booth Encoding and Dadda Tree Architecture," in *Proc. 2025 5th International Conference on Intelligent Technologies (CONIT)*, 2025, pp. 1-5. doi: 10.1109/conit65521.2025.11167872

[5] X. Li, X. Li, X. Luo, Y. Guo, "FPGA-Based Approximate Booth Multiplier with Optimized Encoding and Configurable Accumulation," in *Proc. 2025 4th International Conference on Electronic Information Technology (EIT)*, 2025, pp. 76-79. doi: 10.1109/eit67313.2025.11232020

[6] X. Zhu, H. Li, Y. Song, Y. Chen, X. Guo, "High Energy Efficiency Radix-4 Booth Multiplier with Zero Encoding Skipping Mechanism," in *Proc. 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 228-233. doi: 10.1109/isvlsi61997.2024.00050

[7] R. Aradhya H V, D. Hegde, "Implementation of Power Efficient Radix-4 Booth Multiplier with Pre-encoding," in *Proc. 2023 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, 2023, pp. 1-5. doi: 10.1109/csitss60515.2023.10334241

[8] P.U. Mehar Sai Nagesh, V. Satyanarayana, "Design of Radix-4 Booth (Radix 256 booth encoding) Multiplication Algorithm," in *Proc. 2026 4th International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, 2026, pp. 1-6. doi: 10.1109/ickecs70176.2026.11528088

[9] Z. Ma, Z. Li, Y. Wang, Y. Li, J. Yu, K. Wang, "Booth-NeRF: An FPGA Accelerator for Instant-NGP Inference with Novel Booth-Multiplier," in *Proc. 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 527-532. doi: 10.1109/asp-dac58780.2024.10473990

[10] C.K.N. S, B. S, A.K. Singh, M. Rao, "Hardware Efficient Multiplier Design Using an Optimal Mix of Approximate Booth Encodings," in *Proc. 2025 IEEE 43rd International Conference on Computer Design (ICCD)*, 2025, pp. 41-48. doi: 10.1109/iccd65941.2025.00013

[11] J. Kim, G. Park, Y. Lee, "Low-Power Encoder and Compressor Design for Approximate Radix-8 Booth Multiplier," in *Proc. 2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2024, pp. 1-5. doi: 10.1109/iscas58744.2024.10558596

[12] T.V.S. Adinarayana, G. Rajesh, Y.L. Reddy, V.R. Yadav, S.A. Gandla, "Design of a 16-Bit Power-Efficient Posit Multiplier with Selective Activation and Modified Radix-4 Booth Multiplier," in *Proc. 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI)*, 2025, pp. 514-518. doi: 10.1109/icmcsi64620.2025.10883248

[13] Y.S. J, S.M. G, M. G, K. Mariammal, "Design of Efficient Booth Multiplier based Polyphase FIR Filters," in *Proc. 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, 2023, pp. 1834-1841. doi: 10.1109/icaiss58487.2023.10250454

[14] A. Rafiq, A. Bosio, S. Pappalardo, M. Jenihhin, "AxEnMULT: Design of an Efficient and Reliable Approximate Encoding-Based Multiplier," in *Proc. 2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2025, pp. 1-6. doi: 10.1109/isvlsi65124.2025.11130248

[15] A. Battu, M. Acharyulu, I. Sravani, V. Tadi, "FPGA Design and Implementation of Approximate Radix 16 Booth Multiplier using Error Detection and Correction," in *Proc. 2025 IEEE Madhya Pradesh Section Conference (MPCON)*, 2025, pp. 378-382. doi: 10.1109/mpcon66082.2025.11256549