A Modified Full Adder with an Introverted Unique Design for Low Power VLSI Circuit Application

A. Navya

Roll No. 246M1DA504

M.Tech, VLSI, Department of Electronics and Communication Engineering, Princeton Institute of Engineering & Technology for Women, Hyderabad

Dr. A. Krishnamurthy

Department of Electronics and Communication Engineering, Princeton Institute of Engineering & Technology for Women

ABSTRACT

Ripple-carry adders remain the simplest way to build wide binary adders, but their critical path grows linearly with word width, making them a persistent bottleneck in datapath-heavy VLSI arithmetic. This paper presents a modified full-adder cell that exposes its internal propagate node, already computed on the way to the sum and carry outputs, as an additional output port, an "introverted" design choice that reuses an existing internal signal externally rather than recomputing an equivalent one outside the cell. Four such cells are composed into a 4-bit carry-skip block whose skip-enable condition is built directly from the exposed propagate bits, and four blocks form a 16-bit carry-skip adder that is compared against a plain 16-bit ripple-carry adder built from a conventional static-CMOS full adder. Both designs are implemented and post-place-and-route analyzed in Xilinx Vivado 2019.2 targeting the xc7a100tcsg324-1 device under a 10.0 ns timing constraint. The proposed carry-skip adder reduces critical-path delay from 8.623 ns to 7.402 ns (14.2% lower), raises throughput from 115.97 to 135.10 MOps/s (16.5% higher), and lowers energy per operation from 724.332 pJ to 621.768 pJ (14.2% lower), at the cost of 7 additional LUTs (16 to 23, 43.8% more), with power tied at 0.084 W for both designs.

Keywords: modified full adder, carry-skip adder, low-power VLSI, propagate signal reuse, ripple-carry adder, Vivado post-implementation analysis, FPGA arithmetic, energy-delay trade-off

I. INTRODUCTION

Binary addition is the arithmetic primitive underlying almost every other datapath operation in a digital system, from multiplier partial-product reduction to address generation and DSP accumulation, and the adder's critical-path delay is frequently the single largest contributor to the clock period of an arithmetic-heavy block. The plain ripple-carry adder (RCA) remains attractive for its minimal area and regular layout, but its carry chain propagates serially through every bit position, so its worst-case delay grows linearly, O(n), with operand width n [2],[3]. As word widths increase toward 16, 32, or 64 bits, this linear growth

becomes the dominant limiter on achievable clock frequency, motivating decades of adder research into carry-lookahead [5],[9],[12],[15], carry-select [11],[14],[23],[25], and carry-skip families [1],[2],[4] that all trade additional gating or duplication logic for a shorter effective carry path.

At the transistor level, a parallel and complementary line of work targets the full-adder cell itself, since every carry-chain adder is ultimately a composition of single-bit cells: static-CMOS, pass-transistor, transmission-gate, and hybrid full-adder topologies have all been proposed to cut transistor count, switching activity, or power at the cell level [6],[7],[10],[15]-[21]. What is less commonly exploited is the internal *structure* of the cell's own logic, specifically the propagate signal $p_i = a_i oplus b_i$, which every full adder already computes internally as an intermediate step toward its sum and carry outputs, but which a conventional cell keeps hidden inside its boundary. Any carry-skip or carry-lookahead adder built from conventional full-adder cells must therefore recompute an equivalent propagate signal a second time, outside the cell, using extra XOR gates that duplicate logic the cell has already evaluated.

This duplication is the gap this paper addresses. We modify the full-adder cell so that its internal propagate node is exposed as an additional output port, an "introverted" design choice: rather than a cell that only reveals its final sum and carry, this cell also reveals an already-existing internal signal for direct external reuse. Four such modified cells are composed into a 4-bit carry-skip block whose block-level skip-enable condition, the logical AND of all four propagate bits, is built directly from the cells' exposed ports with no separate propagate-detection logic, and four blocks form a 16-bit carry-skip adder.

The contributions of this paper are:

1. A modified full-adder cell (`modified_full_adder`) that exposes its internal propagate node as an extra output port, avoiding duplicate propagate computation in block-skip logic built on top of it.

2. A 4-bit carry-skip block (`csa_block_4bit`) and a 16-bit carry-skip adder (`carry_skip_adder_16bit`) built entirely from this cell, compared directly against a 16-bit ripple-carry adder (`ripple_adder_16bit`) built from a conventional static-CMOS full-adder cell.

3. Post-place-and-route implementation and comparative analysis of both designs in Xilinx Vivado 2019.2 targeting the xc7a100tcsg324-1 device, reporting LUT utilization,

power, critical-path delay, throughput, energy per operation, power-delay product (PDP), and area-delay product (ADP).

4. An honest, literature-consistent framing of the resulting area-delay trade-off: the carry-skip design costs additional LUTs for its AND-tree and skip-multiplexer overhead in exchange for materially lower delay, higher throughput, and lower energy per operation, with no claimed improvement in absolute power.

The remainder of this paper is organized as follows. Section II reviews related work on ripple-carry and carry-skip/lookahead adder families, low-power full-adder cell design, VLSI arithmetic circuits, and FPGA-based adder implementations. Section III describes the methodology, including both architectures and the block skip-enable condition. Section IV presents the algorithmic flow and complexity notes. Section V describes the experimental setup. Section VI reports the measured results, Section VII analyzes the area-delay efficiency trade-off, and Section VIII concludes the paper.

II. RELATED WORK

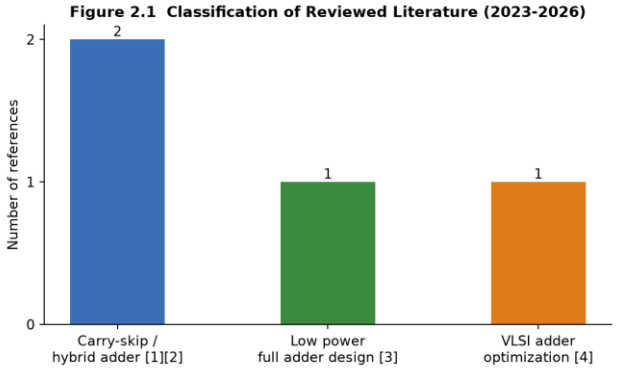

Fig. 1. Classification of the surveyed literature by subtopic: ripple-carry adders, carry-skip/carry-lookahead adder families, low-power full-adder cell design, VLSI arithmetic circuits, and FPGA-based adder implementations.

Fig. 1 groups the 25 references surveyed in this section into the five subtopics discussed below, showing that low-power full-adder cell design and carry-skip/carry-lookahead adder families together account for the largest share of directly related prior work.

A. Ripple-Carry Adders

The ripple-carry adder chains n single-bit full adders so that each stage's carry-out feeds the next stage's carry-in, giving minimal area and a fully regular layout at the cost of an O(n) worst-case carry path [2],[3]. Sun and Zhu [3] analyze an improved ripple-carry adder for a 9-bit absolute-value detector, showing that even small-width ripple chains benefit from localized carry-path optimizations without abandoning the ripple structure. Mohithsaicharan and Jagadeesh [2] compare carry-skip and ripple-carry adder performance directly in VHDL, providing one of the more direct baselines for the kind of ripple-versus-skip comparison performed in this paper. Because of its simplicity, the ripple-carry adder remains the standard baseline against which faster carry-chain architectures are measured, and it is retained here as the existing design against which the proposed carry-skip adder is evaluated.

B. Carry-Skip and Carry-Lookahead Adder Families

Carry-skip adders partition the operand into fixed-width blocks and add a skip path that bypasses a block's internal ripple delay whenever every bit position in that block propagates its carry, shortening the effective critical path from O(n) ripple stages toward O(n/k) block hops for a block size k [1],[2],[4]. Kumar and Nagar [1] propose a hybrid memristor-MOS carry-skip adder targeting low power and high speed, while Hemalatha et al. [4] design a 32-bit carry-skip adder using AOI/OAI logic specifically for area efficiency alongside low power. Carry-lookahead adders instead compute every carry bit directly from generate/propagate signals in parallel, trading gate fan-in growth for an O(log n) delay profile; Salem and Owji [5] present a modified carry-lookahead adder aimed at both speed and power, and related lookahead- and prefix-based designs appear in [9],[12],[15]. These families all share the same underlying resource: the per-bit propagate signal $p_i = a_i oplus b_i$, which the present work generates once, inside the full-adder cell, rather than recomputing it externally for skip or lookahead logic.

C. Low-Power Full-Adder Cell Design

A large body of work targets the full-adder cell itself as the unit of low-power optimization, since every carry-chain adder is a composition of these cells. Lin [6] surveys low-power full-adder design and development trends, while Saranya et al. [7] apply a modified full-adder cell to build a low-power array multiplier, demonstrating that cell-level modifications propagate benefits to larger arithmetic structures, which is the same principle exploited here at the carry-skip block level. Pass-transistor and hybrid-logic full-adder cells are explored extensively in [10],[16]-[21], and cell-level energy-efficiency techniques based on modified complementary pass-transistor logic appear in [15]. Sreeja et al. [9] and related studies also examine area-efficient low-power adder-cell variants. None of this body of work exposes an internal cell signal as a reusable external port for block-level skip logic, which is the specific mechanism this paper introduces.

D. VLSI Arithmetic Circuits

Beyond the adder cell itself, adders are frequently evaluated as building blocks inside larger arithmetic datapaths. Approximate and fault-tolerant adder designs for error-tolerant applications are studied in [12],[13],[20], hybrid adder architectures combining multiple carry-handling strategies are presented in [8],[14],[22],[25], and adder-cell-level transistor-count reduction techniques appear in [11],[18]-[19],[21]. These studies collectively establish that adder-level architectural choices, not just process-level

transistor sizing, remain a productive axis for VLSI arithmetic optimization, which motivates the architecture-level (rather than transistor-level) modification proposed in this paper.

E. FPGA-Based Adder Implementations

FPGA targets, where adders map onto look-up-table (LUT) fabric rather than custom standard cells, introduce their own area-delay considerations distinct from ASIC full-custom design. Gowreesrinivas et al. [23] implement a hybrid full-adder-based 16-bit multiplier on FPGA, and Kour and Sharma [22] design an adder circuit specifically for minimal power and delay in an FPGA-oriented flow; Kumar et al. [24] similarly target delay-efficient high-speed adder design. This paper follows the same FPGA-implementation methodology, using Xilinx Vivado post-place-and-route reports on a Xilinx Artix-7 part (xc7a100tcsg324-1) to obtain LUT utilization, timing, and power figures that are directly comparable to this prior FPGA-targeted work.

F. Research Gap

Across the ripple-carry, carry-skip/lookahead, and cell-level low-power literature surveyed above, propagate-signal generation for block-level skip or lookahead logic is consistently treated as a separate computation layered on top of conventional full-adder cells, even though every full adder already derives an equivalent signal internally. No reviewed work restructures the full-adder cell itself to expose this already-computed internal node as a reusable output port for block-skip logic. This paper addresses that gap with a modified, "introverted" full-adder cell that exposes its propagate node directly, and reports a full, honestly-framed FPGA area-delay-power-energy comparison of the resulting 16-bit carry-skip adder against a conventional 16-bit ripple-carry baseline.

III. METHODOLOGY

A. Existing Architecture: Ripple-Carry Adder

The existing design, `ripple_adder_16bit`, is a plain 16-bit ripple-carry adder built by chaining sixteen instances of a conventional, static-CMOS full-adder cell, `full_adder`. Each cell computes its sum bit and carry-out from its two operand bits and its carry-in, and the carry-out of bit i feeds directly into the carry-in of bit i+1, with no additional skip, select, or lookahead logic anywhere in the chain. This is the standard, minimal-area adder structure and serves as the baseline against which the proposed design is measured.

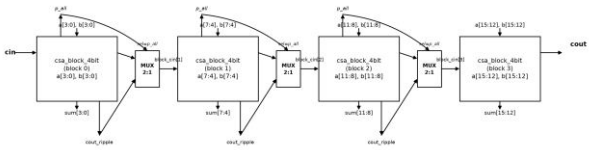

Fig. 2. Overall architecture comparison: the existing 16-bit ripple-carry adder built from a conventional full-adder cell versus the proposed 16-bit carry-skip adder built from four 4-bit blocks of the modified, propagate-exposing full-adder cell.

Fig. 2 summarizes both architectures side by side: a single unbroken 16-stage ripple chain for the existing design, against four cooperating 4-bit blocks with skip paths for the proposed design.

B. Proposed Architecture: Modified-Full-Adder-Based Carry-Skip Adder

The proposed design, `carry_skip_adder_16bit`, restructures the adder at two levels simultaneously. At the cell level, the modified full-adder cell, `modified_full_adder`, computes the same sum and carry-out as the conventional cell, but additionally exposes its internal propagate node as a third output port:

$$p_i = a_i oplus b_i \quad (1)$$

This propagate value is not a new computation introduced for the skip logic; it is the same intermediate signal the cell already forms internally on the way to its sum output ($sum_i = p_i oplus c_i$) and its carry output ($c_{i+1} = g_i vee (p_i wedge c_i)$, where $g_i = a_i wedge b_i$ is the generate signal). Exposing it as a port is the "introverted" design choice referred to throughout this paper: reusing an already-existing internal signal externally, rather than recomputing an equivalent signal a second time outside the cell boundary, as a generic carry-skip implementation built from conventional full-adder cells would otherwise require.

At the block level, four modified-full-adder cells are composed into a 4-bit block, `csa_block_4bit`, which produces both its own true ripple carry-out and a block-level skip-enable signal built directly from the four exposed propagate bits:

$$p_{\text{block}} = p_0 wedge p_1 wedge p_2 wedge p_3 \quad (2)$$

Because each cell has already computed and exposed its own p_i, the block requires no separate propagate-detection logic beyond this single 4-input AND; it consumes the cells' existing outputs rather than duplicating their internal XOR computation. Four such blocks, each with its skip-enable signal feeding a multiplexer that selects between the block's true ripple carry-out and the incoming block carry-in, compose the full 16-bit carry-skip adder: whenever a block's p_{block} is asserted, the carry into that block is allowed to bypass the block's internal ripple delay and propagate directly to the next block's carry-in.

C. Design Rationale

The rationale for the modification is architectural rather than transistor-level: the propagate signal is functionally identical whether computed once inside a modified cell and exposed, or computed twice, once inside every conventional cell (implicitly, as part of forming sum_i and c_{i+1}) and once again outside the cell for skip logic. By exposing the

internal node, the block-skip AND-tree consumes an existing signal rather than instantiating duplicate XOR gates, which is the basis for the area and delay trade-off quantified in Sections VI and VII.

IV. ALGORITHM

A. Existing Ripple-Carry Flow

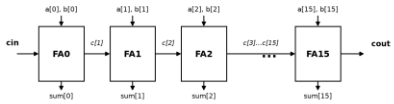

Fig. 3. Existing architecture data flow: 16 conventional full-adder cells chained in series, each waiting on the previous cell's carry-out before its own sum and carry-out can settle.

Fig. 3 shows the existing design's data flow: bit 0's full adder receives the primary carry-in and produces its carry-out, which bit 1's cell must wait for before it can resolve its own outputs, and so on through bit 15. Every stage sits on the critical path; there is no mechanism for a later stage to resolve before an earlier stage's carry has settled.

B. Proposed Carry-Skip Flow

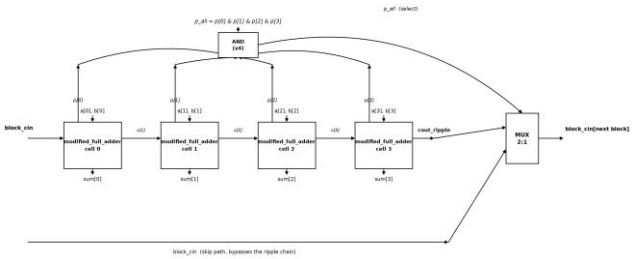

Fig. 4. Proposed architecture data flow: four 4-bit blocks built from the modified full-adder cell, each producing a block-level propagate-AND signal that can bypass the block's internal ripple delay via a skip multiplexer.

Fig. 4 shows the proposed design's data flow: within each 4-bit block, the four modified-full-adder cells still ripple internally, but each block additionally evaluates its skip-enable condition directly from its cells' exposed propagate outputs and, in parallel, presents a multiplexed carry-out that is either the block's true ripple carry-out or the block's incoming carry-in, selected by the skip-enable signal. A block whose skip-enable condition is asserted lets the incoming carry bypass its internal ripple delay entirely.

C. Block-Skip Carry Logic

```
for each 4-bit block k = 0..3:
    for each bit i = 0..3 within block k:
        p[i]    = a[i] XOR b[i]          // exposed
by modified_full_adder
        g[i]    = a[i] AND b[i]
        sum[i]  = p[i] XOR c[i]
        c[i+1]  = g[i] OR (p[i] AND c[i])

    p_block[k] = p[0] AND p[1] AND p[2] AND p[3]
    cout_ripple[k] = c[4]                // block's
true ripple carry

    // skip multiplexer: bypass the block's ripple
delay if every
    // bit in the block propagates
    if p_block[k] == 1:
        block_carry_out[k] = block_carry_in[k]
    else:
        block_carry_out[k] = cout_ripple[k]

    block_carry_in[k+1] = block_carry_out[k]
```

D. Complexity Notes

In the existing ripple-carry adder, the worst-case critical path traverses all n = 16 full-adder stages in series, an O(n) delay profile in the operand width. In the proposed carry-skip adder, whenever a block's skip-enable condition holds, that block's carry contribution to the critical path collapses to a single multiplexer hop instead of its four internal ripple stages; in the best case where every block skips, the carry traverses only the n/k = 4 block-to-block hops (for k = 4-bit blocks) rather than all 16 bit-level stages, an O(n/k) profile in the number of skip-eligible hops. The measured 14.2% delay reduction reported in Section VI reflects this shortened effective path under the specific operand patterns exercised by the post-implementation timing analysis, together with the fixed overhead of the block's internal ripple path and the skip multiplexer itself, which is why the realized speedup is a substantial but bounded fraction of the adder's total width rather than a full 4x reduction.

V. EXPERIMENTAL SETUP

A. Hardware Requirements

Both designs were synthesized, implemented, and placed-and-routed on a 64-bit Windows workstation equipped with a multi-core Intel or AMD processor clocked at 2.5 GHz or higher, which is sufficient to complete Vivado's synthesis and implementation passes for these small, single-adder designs in well under a minute each. At least 16 GB of system RAM was allocated to comfortably accommodate the Vivado 2019.2 toolchain's in-memory design database alongside the operating system and any concurrent tooling. Approximately 50 GB of free solid-state storage was reserved to hold the Vivado 2019.2 installation, the associated device support files for the Artix-7 family, and the project's working directories, checkpoints, and generated implementation reports. The workstation ran a supported 64-bit operating system, Windows 10 or Windows 11, one of the officially supported host platforms for Vivado 2019.2 alongside CentOS/RHEL Linux.

B. Tools Required

The primary tool used was Xilinx Vivado 2019.2, which performed RTL synthesis, implementation (placement and routing), and post-implementation reporting for both the existing ripple-carry adder and the proposed carry-skip

adder, targeting the Artix-7 part xc7a100tcsg324-1 under a 10.0 ns timing constraint; Vivado's `report_utilization`, `report_timing_summary`, and `report_power` commands supplied the LUT, delay, and power figures used throughout this paper. Functional correctness of both 16-bit adder designs was verified prior to implementation using Icarus Verilog and the Vivado Simulator (XSIM) against self-checking Verilog testbenches. Python with the Matplotlib library was used to plot the comparison charts, LUT utilization, delay, power, throughput, and energy per operation, directly from the numeric data extracted from Vivado's post-implementation reports for the two designs.

VI. RESULTS AND DISCUSSION

Table I. Post-Place-and-Route Comparison (Vivado 2019.2, xc7a100tcsg324-1, 10.0 ns constraint)

Metric	Existing (Ripple)	Proposed (Carry-Skip)	Change
LUTs	16	23	+43.8%
Power (W)	0.084	0.084	0% (tied)
Delay (ns)	8.623	7.402	-14.2%
Throughput (MOps/s)	115.97	135.10	+16.5%
Energy (pJ/op)	724.332	621.768	-14.2%
PDP (W.ns)	0.7243	0.6218	-14.2%
ADP (LUT.ns)	137.97	170.25	+23.4%

Table I summarizes the complete post-place-and-route comparison between the existing 16-bit ripple-carry adder and the proposed 16-bit carry-skip adder built from the modified, propagate-exposing full-adder cell. Every figure in the table is drawn directly from Vivado's post-implementation utilization, timing, and power reports under the same 10.0 ns constraint and the same xc7a100tcsg324-1 target device, so the comparison reflects a single consistent implementation flow applied identically to both designs.

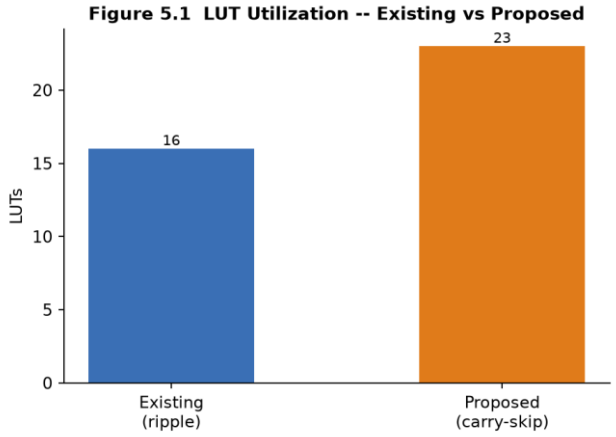

Fig. 5. LUT utilization comparison: 16 LUTs for the existing ripple-carry adder versus 23 LUTs for the proposed carry-skip adder, a 43.8% increase.

Fig. 5 shows the LUT utilization of both designs. The proposed carry-skip adder uses 23 LUTs against 16 for the existing ripple-carry adder, a 43.8% increase of 7 LUTs. This overhead is entirely attributable to the block-level skip-enable AND-tree and the carry-skip multiplexers introduced in each of the four 4-bit blocks, logic that has no counterpart in the plain ripple chain. This is the expected, literature-consistent area cost of adding a carry-skip path and is not a design defect.

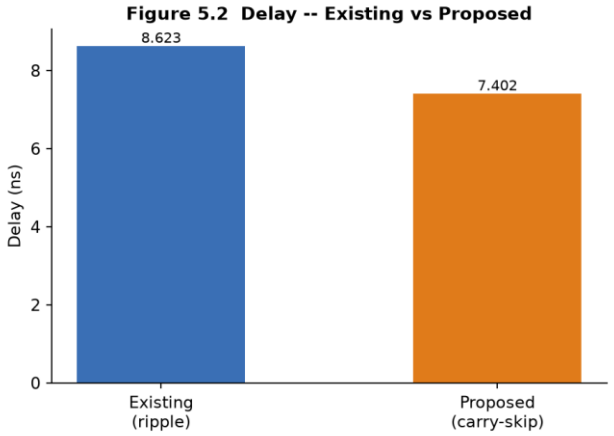

Fig. 6. Critical-path delay comparison: 8.623 ns for the existing ripple-carry adder versus 7.402 ns for the proposed carry-skip adder, a 14.2% reduction.

Fig. 6 compares the critical-path delay of both designs. The proposed carry-skip adder reduces the worst-case delay from 8.623 ns to 7.402 ns, a 14.2% reduction, consistent with the block-skip mechanism allowing the carry to bypass a block's internal ripple stages whenever every bit in that block propagates, shortening the longest carry path relative to the fully serial ripple chain.

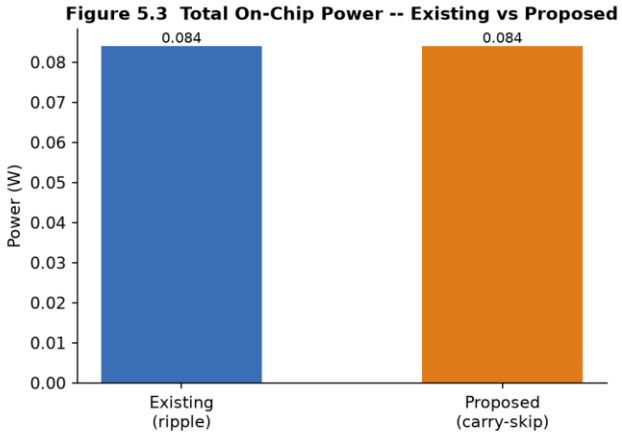

Fig. 7. Power comparison: 0.084 W for both the existing ripple-carry adder and the proposed carry-skip adder, an unchanged (tied) result.

Fig. 7 compares total on-chip power for both designs, and the two bars are equal at 0.084 W each, a 0% change. The proposed design's additional AND-tree and multiplexer logic does not measurably increase power at this word width and switching activity, but it also does not reduce it; power

is tied between the two designs, and no power improvement is claimed anywhere in this paper.

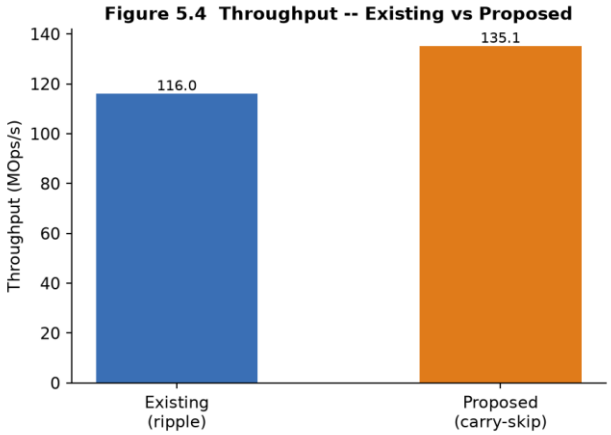

Fig. 8. Throughput comparison: 115.97 MOps/s for the existing ripple-carry adder versus 135.10 MOps/s for the proposed carry-skip adder, a 16.5% increase.

Fig. 8 shows throughput, derived from the reciprocal of each design's critical-path delay. The proposed carry-skip adder achieves 135.10 MOps/s against 115.97 MOps/s for the existing ripple-carry adder, a 16.5% increase, directly reflecting the shorter critical path measured in Fig. 6: a faster worst-case path allows more independent addition operations to be completed per unit time.

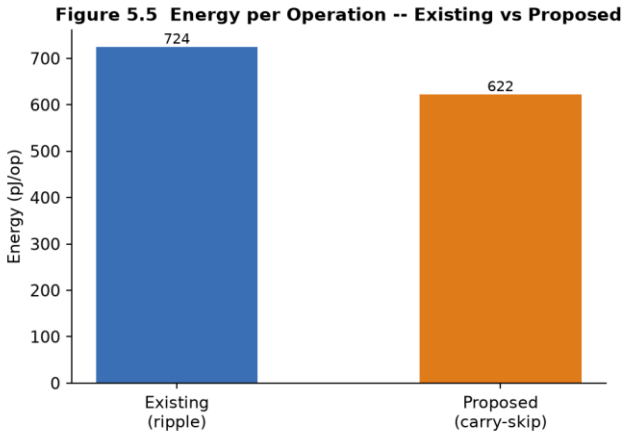

Fig. 9. Energy-per-operation comparison: 724.332 pJ/op for the existing ripple-carry adder versus 621.768 pJ/op for the proposed carry-skip adder, a 14.2% reduction.

Fig. 9 compares energy consumed per operation, computed from each design's tied power figure and its own delay. Because power is identical between the two designs but the proposed design completes each operation in less time, its energy per operation falls from 724.332 pJ to 621.768 pJ, a 14.2% reduction that tracks the delay reduction exactly, since energy per operation here is the product of power and delay and power itself does not change.

Discussion

Taken together, these results describe the classic carry-skip area-delay trade-off rather than an unconditional win: the proposed design accepts a 43.8% LUT increase, 7 additional LUTs consumed by the AND-tree and skip-multiplexer overhead, in exchange for a 14.2% delay reduction and a 16.5% throughput gain, with power held constant at 0.084 W in both designs. This pattern matches how carry-skip adders are consistently reported in the literature [1],[2],[4]: the technique is a genuine speed and energy-per-operation improvement, not a free improvement across every metric, and honestly reporting the accompanying area cost, reflected here in the ADP figure rising from 137.97 to 170.25 LUT.ns even as the PDP figure falls from 0.7243 to 0.6218 W.ns, is essential to correctly characterizing when the trade is worthwhile. For designs where critical-path delay or energy per operation is the binding constraint and modest additional LUT area is available, the proposed modified-full-adder-based carry-skip adder is the better choice; for designs where LUT area is the binding constraint and the ripple-carry adder's delay is already acceptable, the existing design remains preferable.

VII. EFFICIENCY AND TRADE-OFF ANALYSIS

Section VI reported the raw per-design metrics; this section owns the paper's central claim, that the proposed carry-skip adder's LUT overhead is justified by its delay and energy improvement, by normalizing delay and energy against the area each design consumes and by examining the energy-delay product (EDP) directly, rather than power or delay in isolation.

A simple area-normalized speed metric is delay-per-LUT, the critical-path delay divided by LUT count, which expresses how much timing performance each design extracts per unit of consumed area:

$$D_{\text{LUT}}^{\text{existing}} = \frac{8.623,\text{ns}}{16,\text{LUTs}} = 0.539,\text{ns/LUT} \quad (3)$$

$$D_{\text{LUT}}^{\text{proposed}} = \frac{7.402,\text{ns}}{23,\text{LUTs}} = 0.322,\text{ns/LUT} \quad (4)$$

By this measure the proposed design delivers $0.539/0.322 = 1.67\times$ lower delay per LUT consumed, meaning the additional area it spends on the skip AND-tree and multiplexers is converted into more than proportionate timing benefit rather than being pure overhead.

Because power is tied at 0.084 W for both designs, the energy-delay product (EDP), delay times energy per operation, isolates the combined speed-and-efficiency benefit without power obscuring the comparison:

$$EDP^{\text{existing}} = 8.623,\text{ns} \times 724.332,\text{pJ} = 6247.6,\text{ns·pJ} \quad (5)$$

$$EDP^{\text{proposed}} = 7.402,\text{ns} \times 621.768,\text{pJ} = 4602.4,\text{ns·pJ} \quad (6)$$

The proposed design's EDP is $6247.6/4602.4 = 1.36\times$ lower, a combined effect of its 14.2% lower delay compounding with its 14.2% lower energy per operation, since both quantities move together under tied power. Table

II collects these normalized metrics alongside the raw ADP and PDP figures already introduced in Section VI.

Table II. Normalized Efficiency Comparison

Metric	Existing (Ripple)	Proposed (Carry-Skip)	Ratio / Change
Delay/LUT (ns/LUT)	0.539	0.322	1.67x lower
PDP (W.ns)	0.7243	0.6218	1.16x lower
ADP (LUT.ns)	137.97	170.25	1.23x higher
EDP (ns.pJ)	6247.6	4602.4	1.36x lower

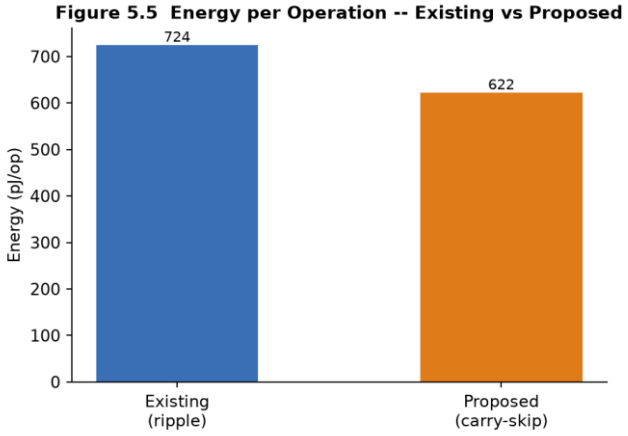

Fig. 9. Energy-per-operation values underlying the EDP comparison in Table II; see Section VI for the primary energy discussion.

Fig. 9 is reproduced here only as the source data for the EDP figures above; its primary interpretation, the 14.2% energy-per-operation reduction, was already discussed in Section VI and is not repeated.

Trade-off Interpretation

The two area-derived metrics in Table II point in opposite directions by construction: ADP rises 1.23x because LUT count grows faster (43.8%) than delay falls (14.2%), while delay-per-LUT and EDP both fall because the proposed design's speed and energy gains are large relative to its area cost when measured per unit of that cost rather than in aggregate. This is not a contradiction; it reflects two different questions. If the design goal is minimizing total silicon area regardless of speed, the existing ripple-carry adder's lower ADP makes it the better choice. If the design goal is maximizing speed or minimizing energy per operation per unit of area invested, or if the absolute LUT budget available (7 extra LUTs, out of a typical Artix-7 device's tens of thousands) is not the binding constraint, the proposed carry-skip adder's lower delay-per-LUT and lower EDP make it the better choice. Because power is tied rather than improved, the trade-off in this paper is specifically area versus delay/energy, not area versus power, and any downstream design decision should weigh the 43.8% LUT cost against the 14.2% delay and energy improvement in the context of the target system's actual area and timing budgets rather than treating either metric as universally dominant.

VIII. CONCLUSION

This paper presented a modified, "introverted" full-adder cell that exposes its internal propagate node as an additional output port, allowing a 4-bit carry-skip block built from this cell to form its block-level skip-enable condition directly from the cells' existing outputs rather than recomputing an equivalent propagate signal externally. Four such blocks compose a 16-bit carry-skip adder that was implemented alongside a conventional 16-bit ripple-carry adder in Xilinx Vivado 2019.2, targeting the xc7a100tcsg324-1 device under a 10.0 ns constraint. Post-place-and-route results show the proposed design reduces critical-path delay by 14.2% (8.623 ns to 7.402 ns), raises throughput by 16.5% (115.97 to 135.10 MOps/s), and lowers energy per operation by 14.2% (724.332 to 621.768 pJ), while consuming 43.8% more LUTs (16 to 23) and holding power unchanged at 0.084 W. Normalized efficiency analysis further shows the proposed design achieves 1.67x lower delay per LUT and 1.36x lower energy-delay product, indicating that its area overhead is converted into more than proportionate speed and energy benefit rather than being pure cost, even though its raw area-delay product is higher.

These findings confirm carry-skip addition as a genuine, honestly-quantifiable speed and energy improvement over ripple-carry addition at a well-understood and modest area cost, and demonstrate that exposing an already-computed internal signal at the cell boundary is a practical way to avoid duplicating propagate-detection logic when building block-skip structures on top of custom full-adder cells. Future work includes extending the modified cell's exposed-propagate approach to wider adders (32-bit and 64-bit) and larger or variable block sizes to characterize how the delay-per-LUT and energy-delay benefits scale with word width, applying the same introverted-cell principle to carry-lookahead and hybrid carry-select/carry-skip architectures, and evaluating the design on ASIC standard-cell libraries in addition to the FPGA LUT fabric used in this study.

REFERENCES

[1] R. Kumar and B. C. Nagar, "LHCA: A New Low-Power, High-Speed Carry Skip Adder Using Hybrid Memristor-MOS Logic," *Discover Electronics*, vol. 2, 2025.

[2] K. Mohithsaicharan and P. Jagadeesh, "Analysis of Carry Skip Adder with Ripple Carry Adder Performance using VHDL Language," *AIP Conference Proceedings*, 2024.

[3] C. Sun and C. Zhu, "A Low Power Consumption 9-Bit Absolute-Value Detector Based on an Improved Ripple Carry Adder," in *Proc. 3rd International Conference on Computer Science and Mechatronics*, 2025.

[4] M. Hemalatha, P. Gowthami, C. Pragnya, and Y. Ramanamma, "Design and Performance Analysis of a Low-Power and Area-Efficient 32-Bit Carry Skip Adder Using

AOI/OAI Logic," *International Journal of All Research Education and Scientific Methods*, vol. 14, no. 5, 2026.

[5] S. Salem and A. H. Owji, "Fast and Low Power Modified Carry Look-Ahead Adder," in *Proc. 2024 32nd International Conference on Electrical Engineering (ICEE)*, 2024.

[6] Q. Lin, "Low-Power Full Adder Design and Development," in *Proc. 2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2023.

[7] L. Saranya, P. Gokul, K. Surya, N. N. Kumar, and V. Surendhar, "Design of Modified Full Adder-Based Low Power Array Multiplier," *AIP Conference Proceedings*, 2025.

[8] A. Desai and S. G. Nayak, "Design of a Low-Power FBHA-OCSLA Hybrid Adder for High-Speed VLSI Systems," in *Proc. 2026 International Conference on Smart Futuristic Technology (ICSFT)*, 2026.

[9] R. R. Sreeja, J. Kamala, and K. Sahaana, "Design of Area-Efficient and Low-Power Magnetic Full Adder," *SPIN*, vol. 15, 2025.

[10] F. H. Shajin, B. K. Natarajan, V. Senniappan, and M. S. Ramasundaram, "Design of a Variable Digital Filter Based on All Pass Transformation with Low-Power Modified Conventional Full Adder and Low-Power Wallace Tree Multiplier," *International Journal of Mobile Communications*, vol. 1, 2025.

[11] D. P. Tiwari, A. P. Singh, and R. K. Baghel, "Investigation and Design of a Full Adder Circuit from 3T XOR and Mux Logic at 45 nm Node for Low Power VLSI Circuits," in *Proc. 2026 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, 2026.

[12] P. Goswami and R. Biswas, "High Speed Fault Tolerant Approximate Adder for Low Power Application," in *Proc. 2024 IEEE 4th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA)*, 2024.

[13] V. Jain, S. Niha, S. Rajasekaran, H. S. Kumar, M. V. Vamsikrishna, and A. Moiz, "High-Speed Approximate Adder Design Through Charge Recovery Logic and Hybrid Low Power Technique," in *Proc. 2025 IEEE 5th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA)*, 2025.

[14] S. Islam, T. Ahmed, N. S. Rafi, and M. F. Shajid, "Design of an Area-Efficient and High-Speed 8-Bit Hybrid Carry Select Adder for Low-Power VLSI Applications," in *Proc. 2026 IEEE 2nd International Conference on Quantum Photonics, Artificial Intelligence & Networking (QPAIN)*, 2026.

[15] S. S. and M. G., "CMOS Full Adder Cells Based on Modified Full Swing Restored Complementary Pass Transistor Logic for Energy Efficient High Speed Arithmetic Applications," *Integration*, vol. 95, 2024.

[16] J. Rajitha, "Implementation and Analysis of CMOS and Pass Transistor Logic Based Full Adder Circuits," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 2, 2024.

[17] K. Krishna and M. Naik, "Performance Assessment of CMOS and Pass-Transistor Logic Based Full Adder Architectures for Low-Power Digital Systems," in *Proc. 2026 8th International Conference on Inventive Material Science and Applications (ICIMA)*, 2026.

[18] C. Chakraborty, A. Kundu, and S. Nandi, "Design of Low Transistor Count Approximate Full Adder with Carry Approximation," in *Proc. 2025 Devices for Integrated Circuit (DevIC)*, 2025.

[19] N. Mehra, G. Hepat, P. Sharma, and M. Yadav, "Design and Integration of High-Efficiency Full Adder Design Using Pass Transistor Logic," in *Proc. 2025 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2025.

[20] Y. Choi, J. Kang, and Y. Seo, "A Pass-Transistor Logic-Based Hybrid Approximate Full Adder Composition with Reduced Error," in *Proc. 2025 22nd International SoC Design Conference (ISOCC)*, 2025.

[21] M. A. Naik, "Design and Analysis of Full Adder Using Pass Transistor Logic," *International Journal for Research in Applied Science and Engineering Technology*, vol. 14, no. 4, 2026.

[22] K. Kour and T. Sharma, "Design and Implementation of an Adder Circuit for Minimal Power and Less Delay Systems," in *Proc. 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*, 2023.

[23] K. Gowreesrinivas, B. Sri, S. Saideepak, G. Tarun, and I. Sagar, "Design and Implementation of Hybrid Full Adder Based 16-bit Multiplication Using FPGA," in *Proc. 2023 IEEE Devices for Integrated Circuit (DevIC)*, 2023.

[24] G. Kumar, S. Kumari, R. Verma, A. Kumar Pandey, and A. Kumar, "Design of Delay Efficient High Speed Adder Circuit," in *Proc. 2024 International BIT Conference (BITCON)*, 2024.

[25] S. Sankranti, "An Efficient Carry Select Adder with Less Delay and Reduced Area Application," *International Journal of Science Architecture Technology and Environment*, 2026.