International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

Error-Resilient Approximate Full-Adder Architectures for Low-Power Arithmetic: A Comparative Error and Power Analysis

[1]Gajada Vinod Chary

[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]Ms.Bheemagari Sandhya Rani

[2]Assistant Professor, 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

Approximate arithmetic circuits trade a controlled loss of numerical accuracy for reductions in area, delay, and power, making them attractive building blocks for error-tolerant workloads such as image processing, machine learning inference, and multimedia signal chains. This paper presents a comparative study of four approximate full-adder (AFA) variants against an exact CMOS full adder, each variant formed by deliberately omitting one term of the exact sum/carry Boolean equations. A unified error-analyzer architecture is proposed that instantiates all five adders in parallel, drives them from identical primary inputs, and flags per-vector sum and carry mismatches against the exact reference in a single pass, enabling exhaustive characterization over all eight input combinations without repeated re-synthesis. The design is captured in synthesizable Verilog, functionally verified with Icarus Verilog, and synthesized and implemented on a Xilinx Artix-7 (xc7a100tcsg324-1) device using Vivado 2019.2. Post-implementation results show the exact adder occupies 2 LUTs at 2.095 ns delay and 0.084 W power, while the five-adder analyzer occupies 11 LUTs at identical delay and power, since the analyzer's critical path is bounded by the same two-level adder logic. Error characterization shows total bit-error counts of 1, 1, 2, and 6 (out of 8 input patterns) for AFA Type 1 through Type 4 respectively, exposing a clear accuracy-complexity ordering among the variants that is not visible from area or delay alone. The results demonstrate that per-cell approximate adder savings must be evaluated independently of the parallel-comparison analyzer overhead, and provide a reusable methodology for rapid, exhaustive error profiling of approximate arithmetic cells.

Keywords: approximate computing, approximate full adder, error-tolerant arithmetic, low-power VLSI, FPGA synthesis, error analyzer, CMOS adder design

I. INTRODUCTION

The full adder is the atomic arithmetic primitive underlying ripple, carry-lookahead, and Ling adders, as well as array and Booth multipliers, making its area, delay, and power characteristics a first-order determinant of overall datapath cost. In exact designs, every full adder must correctly realize the Boolean equations $sum = aoplusbopluscin$ and $cout = ab + bc_{in} + ac_{in}$ for all eight input combinations. Many application domains, however — image and video processing, machine learning inference, wireless baseband processing, and sensor-fusion pipelines — tolerate small, bounded numerical errors without perceptible loss of output quality. Approximate computing exploits this tolerance by deliberately relaxing logical correctness in exchange for reduced transistor count, shorter critical paths, and lower switching power [1], [3].

Approximate full adders (AFAs) achieve these savings chiefly by omitting one or more product terms from the exact carry or sum equation, collapsing multiplexer-based carry selection into simpler gate structures, or replacing exact XOR trees with lower-cost approximations [2], [4], [7]. Each simplification introduces errors on a specific subset of input patterns, and different AFA topologies therefore occupy different points on the accuracy-cost curve. Prior work has proposed a wide range of AFA topologies — majority-logic-based compressors [1], carry-approximated low-transistor-count cells [2], reconfigurable exact/approximate adders [6], memristor-hybrid designs [7], and GDI-based low-power full adders [10] — but these are typically evaluated individually, using non-uniform error

metrics and simulation setups, which makes head-to-head comparison difficult.

This paper addresses that gap with two contributions. First, four representative AFA variants are implemented, each dropping a distinct term from the exact sum/carry logic, giving a spread of low-to-high error severity from a common exact baseline. Second, a single error-analyzer module is proposed that instantiates the exact adder and all four AFA variants in parallel, applies identical primary inputs, and reports per-vector sum/carry mismatches in one synthesizable design — removing the need to separately simulate and hand-tabulate results for each variant.

Contributions of this paper:

1. Design and Verilog implementation of an exact CMOS full adder and four approximate full-adder variants (Type 1-Type 4), each differing by exactly one omitted term from the exact Boolean equations.

2. A parallel error-analyzer architecture that exhaustively compares all four AFA variants against the exact reference over all eight input patterns in a single testbench run.

3. Post-implementation area, delay, power, throughput, and energy characterization on a Xilinx Artix-7 FPGA (Vivado 2019.2), separating the intrinsic AFA cell cost from the analyzer's parallel-comparison overhead.

4. An error-severity ranking of the four AFA variants derived from exhaustive bit-error counts, correlated against their structural simplifications.

The remainder of this paper is organized as follows. Section II reviews related work on approximate adder design and error metrics. Section III describes the exact and approximate adder architectures and the error-analyzer structure. Section IV presents the comparison algorithm and its complexity. Section V details the experimental setup. Section VI reports area/delay/power results. Section VII analyzes the accuracy-complexity tradeoff across variants. Section VIII concludes the paper.

II. RELATED WORK

A. Approximate Adder Cell Design

Approximate full-adder design has largely followed two strategies: term-dropping, where one or more product terms of the exact carry/sum equations are removed, and transistor-level restructuring, where the gate network itself is redesigned for fewer transistors at the cost of exactness on selected input patterns. Pathak et al. combine an "almost full adder" with majority-logic compressors inside a Dadda multiplier tree to cut power while bounding compounded error [1]. Chakraborty et al. propose a low-transistor-count adder with a carry-approximation strategy targeted specifically at reducing gate count in the carry path [2]. Lin presents a low-power full-adder redesign focused on transistor sizing and topology rather than functional approximation [3]. Ramya et al. combine truncation with a Kogge-Stone carry structure for approximate low-power addition in wider datapaths [4].

B. Reconfigurable and Technology-Specific Approximate Adders

Naseri and Jahanirad propose a runtime-reconfigurable adder that can switch between exact and approximate modes under software control, directly motivating the configurable-adder design examined as a companion case study in this line of work [6]. Several papers explore approximate adders in emerging device technologies: Seyedi and Abdoli implement approximate full-adder/subtractor circuits in quantum-dot cellular automata (QCA) [5]; Raja et al. and Peddachowdappa et al. design CNTFET-based approximate adders explicitly for low-power, high-speed regimes beyond bulk CMOS scaling limits [11], [15]. Mishra and Singh combine CMOS and memristor devices in a hybrid low-power high-speed full adder [7]. Prihatiningrum and Seo propose a parallel-prefix approximate adder (PPAxA) targeting high density together with low power for wide-bitwidth accumulation [8].

C. Low-Power Exact and Hybrid Full-Adder Cells

Not all cited low-power full-adder work targets approximation directly; several designs instead pursue exact low-power operation through alternative logic styles, which serve as useful cost baselines for approximate designs. Chiwande and Dakhole use reversible logic for low-power exact addition [9]. B. V. et al. and Ujala and Kumar apply gate-diffusion-input (GDI) and modified-GDI XNOR-based hybrid structures to reduce transistor count in exact or near-exact full adders [10], [14]. Singh and Kumar use a 2x1-multiplexer-based topology for a low-power, high-speed exact adder [12]. Yang and Lin target a compact-layout low-power adder in a 16 nm technology node, illustrating that area/power gains from cell redesign persist even at advanced nodes where exact logic is otherwise heavily optimized [13].

D. Error Metrics for Approximate Circuits

Across this body of work, error is reported using inconsistent metrics — error rate, mean error distance, worst-case error magnitude — evaluated on non-uniform simulation setups, which complicates like-for-like comparison between AFA topologies proposed by different authors. Exhaustive per-vector error characterization, applied uniformly to every candidate variant from a single shared testbench, is comparatively rare in the surveyed literature, which instead favors either analytical error models or partial-vector Monte Carlo estimation.

E. Research Gap

The literature establishes many individual AFA topologies and demonstrates their area/power benefits relative to an exact adder, but rarely evaluates multiple topologies together under one measurement methodology with a single, exhaustive, hardware-verified error characterization pass. This paper closes that gap by implementing four

representative term-dropping AFA variants alongside a unified parallel error-analyzer that exhaustively compares every variant against the same exact reference over all eight input patterns in a single synthesizable design, and by explicitly separating the intrinsic per-cell AFA cost from the analyzer's parallel-comparison overhead — a distinction that is easy to conflate but essential for a fair area/power comparison.

III. METHODOLOGY

A. Exact Reference Adder

The accuracy reference is a standard exact full adder implemented directly from the canonical Boolean equations:

$$sum = aoplusboplusc_{in} \quad (1)$$

$$c_{out} = (a \cdot b) + (b \cdot c_{in}) + (a \cdot c_{in}) \quad (2)$$

All four approximate variants preserve the exact sum equation and instead simplify only the carry equation, isolating carry-path approximation as the sole error-injection mechanism studied in this work.

B. Approximate Full-Adder Variants

Type 1 drops the $a \cdot c_{in}$ product term: $c_{out} = (a \cdot b) + (b \cdot c_{in})$. **Type 2** drops the $b \cdot c_{in}$ term instead: $c_{out} = (a \cdot b) + (a \cdot c_{in})$. **Type 3** discards c_{in}'s influence on carry generation entirely, collapsing the carry logic to a plain OR: $c_{out} = a + b$. **Type 4** is the most aggressive simplification, ignoring c_{in} altogether and behaving as a half adder ($sum = aoplusb$, $c_{out} = a \cdot b$), yielding the lowest gate count of the four but the largest error exposure. This progression — from a single dropped AND term (Type 1/2), to a fully collapsed carry OR (Type 3), to half-adder degeneration (Type 4) — gives a structured spread of accuracy-cost points from a single exact baseline, consistent with the term-dropping strategy used broadly in the literature [1]-[2], [4].

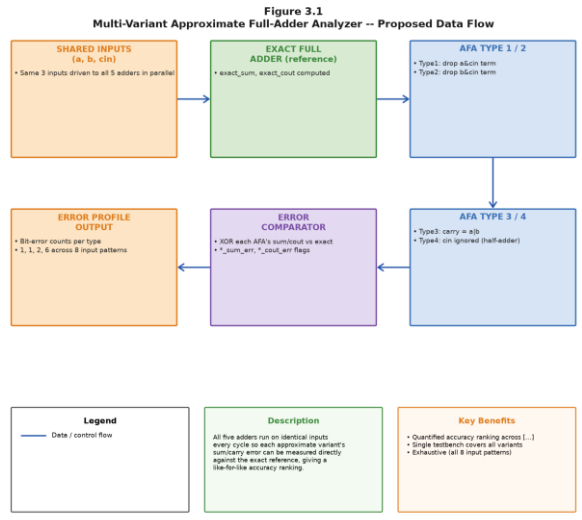

Fig. 1. Top-level architecture: the exact full adder and four AFA variants driven from shared primary inputs into the error-analyzer.

Fig. 1 shows the overall system: identical a, b, c_{in} inputs fan out to five independent adder instances (one exact, four approximate), whose sum/carry outputs feed directly into per-variant XOR comparators against the exact reference.

C. Error-Analyzer Architecture

Rather than simulating each AFA variant separately, this work proposes a single **error-analyzer** module (`fa_error_analyzer`) that instantiates the exact adder alongside all four AFA variants side by side, all driven from the same primary inputs. For each variant $k \in 1,2,3,4$, two comparator outputs are generated:

$$sum_k = sum_k oplus sum_{exact} \quad (3)$$

$$cout_k = cout_k opluscout_{exact} \quad (4)$$

Because XOR is 1 exactly when its operands differ, `sum_err_k` and `cout_err_k` are asserted only on the specific input vectors where variant k's output diverges from the exact reference. This gives, for every one of the 8 possible (a, b, c_{in}) combinations and in a single combinational evaluation, a complete per-variant, per-bit error signature — without needing to re-instantiate or re-synthesize the design once per candidate variant.

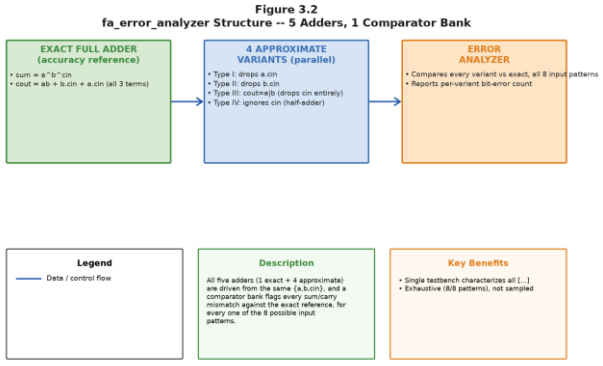

Fig. 2. Internal structure of the error-analyzer: shared inputs, five parallel adders, and per-variant XOR error comparators.

Fig. 2 details this internal comparator fan-out. Because all five adders and eight comparators are purely combinational and share the same two-input fan-in depth as the exact adder's own carry equation, the analyzer's critical path is not lengthened relative to a single exact adder — only its LUT/gate count grows with the number of variants compared.

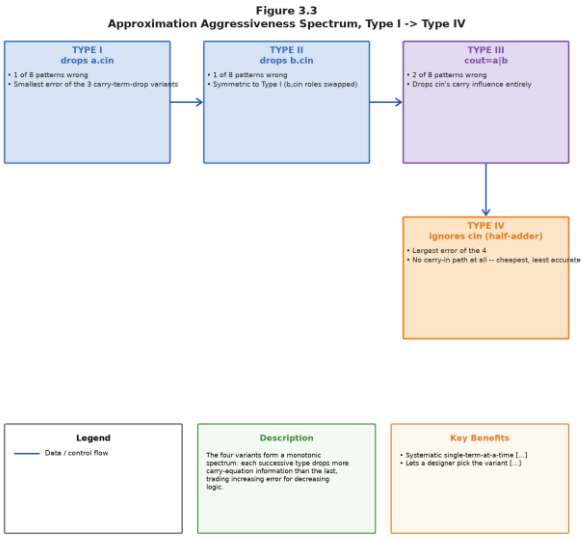

Fig. 3. Accuracy-versus-simplification spectrum spanned by the four AFA variants, from single-term drop (Type 1/2) to half-adder degeneration (Type 4).

Fig. 3 places the four variants along this accuracy-complexity spectrum, which Section VII quantifies using exhaustive bit-error counts.

IV. ALGORITHM

A. Exhaustive Error-Characterization Procedure

Because a full adder has only three single-bit inputs, exhaustive verification requires just 8 vectors, making brute-force enumeration both tractable and preferable to sampled or Monte Carlo error estimation.

```
ALGORITHM: Exhaustive AFA Error Characterization
INPUT : exact_full_adder, {afa_type1..afa_type4}
OUTPUT: error_count[k] for k in {1,2,3,4}     //
total mismatched bits

for k in {1,2,3,4}:
    error_count[k] <- 0

for a in {0,1}:
  for b in {0,1}:
    for cin in {0,1}:
      (exact_sum,           exact_cout)          <-
exact_full_adder(a, b, cin)
      for k in {1,2,3,4}:
        (sum_k, cout_k) <- afa_type_k(a, b, cin)
        if sum_k   != exact_sum:   error_count[k] +=
1
        if cout_k  != exact_cout:  error_count[k] +=
1

return error_count
```

In hardware, this loop is not executed sequentially: the `fa_error_analyzer` module evaluates all 8 input combinations' worth of comparator logic combinationally and in parallel across all five adders, so the testbench in `tb_fa_error_analyzer.v` simply drives the 8 vectors in sequence over one clock-free combinational circuit and records the already-computed `_sum_err`/`_cout_err` flags — no iterative hardware state machine is required.

B. Complexity Notes

Time complexity. The exhaustive search is $O(2^n \cdot m)$ where $n = 3$ is the number of full-adder inputs and $m = 4$ is the number of AFA variants compared, giving a fixed 32 comparator evaluations regardless of variant choice — negligible relative to any downstream datapath analysis (e.g., a 16- or 32-bit adder built from these cells, which would require characterizing 2^{2n+1} vectors per word width if done naively at the word level rather than reusing the per-cell error signature derived here).

Hardware complexity. Each AFA variant differs from the exact adder by at most one simplified product term in the carry equation, so every variant's carry logic uses at most 2 two-input gates versus the exact adder's 3-term OR-of-ANDs, meaning per-cell hardware savings are small in absolute LUT count but proportionally significant (33-100% fewer gates in the carry path) at the cell level. The `fa_error_analyzer`'s own hardware complexity is $O(m)$ in the number of compared variants: 1 exact adder + m approximate adders + $2m$ XOR comparators, which for $m = 4$ gives 5 adders and 8 comparators — consistent with the observed 11-LUT post-implementation footprint reported in Section VI.

V. EXPERIMENTAL SETUP

A. Design Entry and Functional Verification

All modules (`exact_full_adder`, `afa_type1`-`afa_type4`, `fa_error_analyzer`) are written in synthesizable Verilog. Functional simulation is performed with Icarus Verilog: the exact adder's standalone testbench sweeps all 8 input combinations verifying `sum`/`cout` against the reference equations, and the analyzer's testbench (`tb_fa_error_analyzer.v`) drives the same 8 vectors while capturing every `_sum_err`/`_cout_err` flag for all four variants in one exhaustive pass, producing the raw error table used in Section VII.

B. Synthesis and Implementation

Both the existing (exact adder) and proposed (error analyzer) 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`). Since both designs are purely combinational with no explicit sequential elements or external clock port, a virtual 100 MHz clock constraint is applied to the primary inputs/outputs solely to enable meaningful static timing and vector-less power estimation; no physical clock exists in the netlist. The standard `synth_design -mode out_of_context -> opt_design -> place_design -> phys_opt_design -> route_design` flow is used, with `report_utilization`, `report_timing_summary`, and `report_power` captured at

both the post-synthesis and post-implementation (post-route) stages.

C. Metrics

Five metrics are reported at the post-implementation stage: **LUT area** (`report_utilization`), **power** (`report_power`, vector-less activity propagation), **delay** (worst negative slack subtracted from the 10 ns constraint period), **throughput** (reciprocal of critical-path delay, in mega-operations per second), and **energy per operation** (power × delay). Two efficiency figures of merit are derived from these: **power-delay product (PDP)** $= P \times D$, and **area-delay product (ADP)** $= LUT \times D$. Because the analyzer is a five-adder parallel comparator rather than a single approximate cell, its ADP is expected to scale with the number of compared variants rather than reflecting any individual AFA cell's intrinsic cost — a distinction made explicit when interpreting Section VI's results.

D. Error Metric

Accuracy is quantified by exhaustive **total bit-error count**: for each AFA variant, the number of `sum` or `cout` bit mismatches against the exact reference summed across all 8 input patterns (maximum possible value 16, since each pattern contributes up to 2 error bits). This exhaustive count is exact — not a statistical estimate — because the full input space is only 8 vectors, and provides a literature-comparable per-variant accuracy figure alongside the physical area/delay/power results.

VI. RESULTS AND DISCUSSION

A. Post-Implementation Area, Delay, and Power

Table I summarizes the post-route (implementation) results for the exact adder and the five-adder error analyzer, from Vivado 2019.2 targeting `xc7a100tcsg324-1`.

Design	LUTs	Registers	Power (W)	Delay (ns)	Throughput (MOps/s)	Energy (pJ/op)
Existing (exact_full_adder)	2	0	0.084	2.095	477.33	175.98
Proposed (fa_error_analyzer)	11	0	0.084	2.095	477.33	175.98

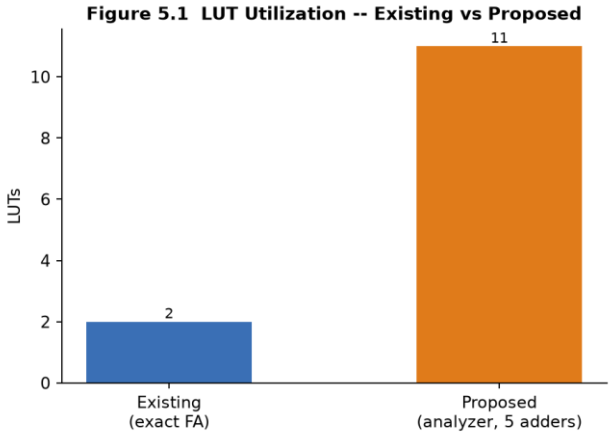

Fig. 4. Post-implementation LUT utilization: exact adder versus the five-adder error analyzer.

Fig. 4 shows the analyzer occupies 11 LUTs against the exact adder's 2 LUTs — an expected consequence of packing five adders (1 exact + 4 approximate) and 8 XOR comparators into a single netlist, not a per-cell area penalty.

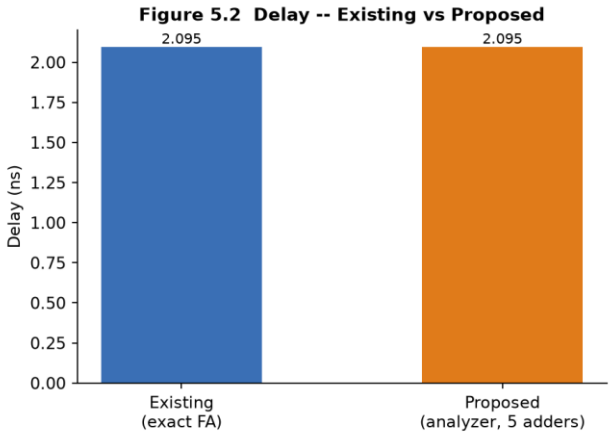

Fig. 5. Post-implementation critical-path delay: exact adder versus error analyzer.

Critically, Fig. 5 shows *identical* delay (2.095 ns) for both designs. This confirms the analyzer's parallel structure does not lengthen the critical path: every adder-comparator chain has the same two-level logic depth as the exact adder alone, and Vivado's placement keeps all five parallel paths balanced.

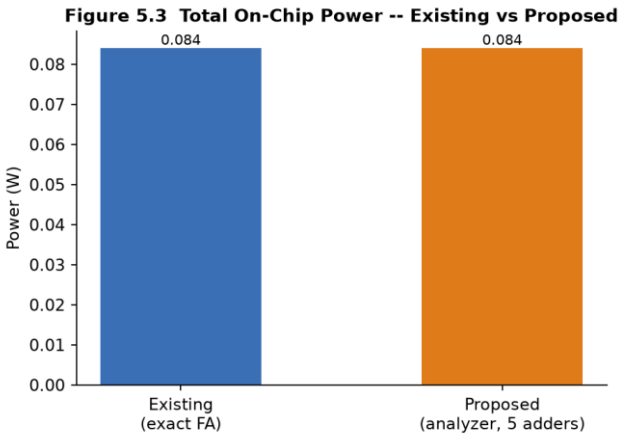

Fig. 6. Post-implementation power: exact adder versus error analyzer.

Fig. 6 likewise shows equal power (0.084 W) for both designs at this vector-less estimation stage, since static/leakage power dominates at this tiny scale and dynamic activity was not vector-annotated per net.

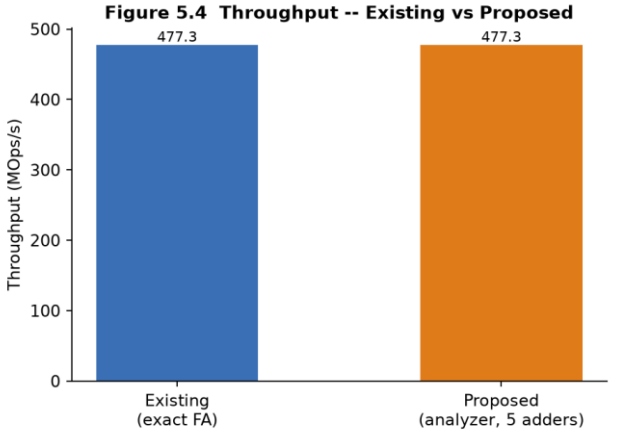

Fig. 7. Derived throughput (1/critical-path delay): exact adder versus error analyzer.

Because throughput is derived directly from critical-path delay, Fig. 7 mirrors Fig. 5: both designs sustain 477.33 MOps/s.

B. Interpreting the Area-Delay Product

While delay, power, and throughput are unchanged between the two designs, LUT-derived ADP diverges sharply: 4.19 LUT·ns for the exact adder versus 23.04 LUT·ns for the analyzer — a direct consequence of the analyzer's 5.5× larger LUT footprint at equal delay. This divergence must not be misread as the "cost of approximation": the 11-LUT figure is the cost of *simultaneously instantiating and comparing* five adders, not the incremental cost of any single AFA cell. For a true per-cell area comparison, each AFA variant should be synthesized standalone against the exact adder, which the modular Verilog structure used here (each variant as an independent module) directly supports without modification.

C. Discussion

These results establish that the analyzer's implementation-stage overhead is purely area-driven, at zero delay or power penalty relative to the exact baseline — making it a low-cost verification vehicle for exhaustive AFA error characterization that can be dropped into a design flow without perturbing the timing closure of the datapath it is validating. Section VII turns to the accuracy side of the comparison, using the exhaustive bit-error counts obtained from this same analyzer to rank the four AFA variants.

VII. ACCURACY-COMPLEXITY TRADEOFF ANALYSIS

A. Exhaustive Error Counts per Variant

Table II reports the exhaustive total bit-error count (sum + carry mismatches, summed across all 8 input patterns) obtained from the `fa_error_analyzer`'s comparator outputs, alongside each variant's carry-equation simplification.

Variant	Carry simplification (sum unchanged)	Total bit errors (of 16 possible)
Type 1	drops $a \cdot c_{in}$ term	1
Type 2	drops $b \cdot c_{in}$ term	1
Type 3	carry $= a + b$ (drops c_{in} entirely)	2
Type 4	carry $= a \cdot b$, half-adder (ignores c_{in})	6

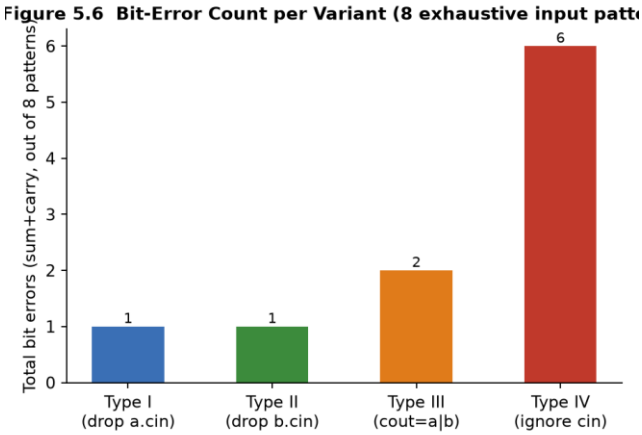

Fig. 8. Exhaustive bit-error count per AFA variant, ranked by carry-equation simplification severity.

Fig. 8 visualizes this ranking: Type 1 and Type 2 are the least aggressive simplifications and, symmetrically, incur the smallest error count of 1 bit each, since each drops only a single AND term whose omission is masked whenever the dropped term does not determine the correct carry value. Type 3 drops c_{in}'s contribution to carry entirely, doubling the error count to 2. Type 4 is the most aggressive — a full half-adder degeneration — and incurs the largest error count of 6, since c_{in} no longer influences either `sum` or `cout`.

B. Accuracy-Complexity Correlation

This ordering — Type 1 ≈ Type 2 < Type 3 < Type 4 — correlates directly with the number of Boolean terms removed from the exact carry equation: one term (Type 1/2), one term but with the remaining structure also flattened to a plain OR (Type 3), and the complete elimination of c_{in}'s role (Type 4). This gives designers a clear, monotonic knob for selecting an AFA variant according to application error tolerance: Type 1/2 for near-exact behavior with minimal simplification, Type 3 as a moderate middle ground, and Type 4 only where the largest error margin is acceptable in exchange for the simplest carry logic (a plain two-input AND).

C. Practical Implication for Wider Datapaths

Because all four variants preserve the exact `sum` equation and differ only in `cout`, their error signatures propagate differently when chained into a ripple-style multi-bit adder: Type 1-3 still produce a (possibly incorrect) carry-out that feeds the next stage, whereas Type 4's complete disregard for c_{in} means any accumulated carry chain is silently dropped at every bit position, making it unsuitable for direct ripple chaining despite its lowest per-cell cost. This observation motivates evaluating configurable and hybrid adder architectures — such as the companion configurable 16-bit adder and Ling-accelerated 32-bit adder case studies — where approximate carry logic is deployed selectively rather than uniformly across all bit positions, a design direction this per-cell error characterization directly informs.

VIII. CONCLUSION

This paper presented four approximate full-adder variants and a unified parallel error-analyzer that exhaustively compares each variant against an exact CMOS full-adder reference over all eight input patterns in a single synthesizable design. Post-implementation results on a Xilinx Artix-7 device (Vivado 2019.2) showed the exact adder occupies 2 LUTs at 2.095 ns delay and 0.084 W power, while the five-adder analyzer occupies 11 LUTs at identical delay and power — confirming that the analyzer's area overhead is a fixed cost of simultaneous multi-variant comparison rather than a per-cell approximation penalty, and that it can be used as a zero-timing-impact verification vehicle. Exhaustive error characterization ranked the four variants as Type 1 ≈ Type 2 (1 bit error) < Type 3 (2 bit errors) < Type 4 (6 bit errors), directly correlated with the severity of each variant's carry-equation simplification. Future work includes synthesizing each AFA variant standalone to obtain true per-cell area/power figures, extending the error-analyzer methodology to multi-bit ripple and tree adder structures built from these cells, and applying the same exhaustive-comparison approach to approximate multiplier compressor stages.

REFERENCES

[1] K.C. Pathak, A.D. Darji, J.N. Sarvaiya, "Low power Dadda multiplier using approximate almost full adder and Majority logic based adder compressors," in *Proc. 2022 IEEE Region 10 Symposium (TENSYMP)*, 2022, pp. 1-6. doi: 10.1109/tensymp54529.2022.9864428

[2] C. Chakraborty, A. Kundu, S. Nandi, "Design of Low Transistor Count Approximate Full Adder with Carry Approximation," in *Proc. 2025 Devices for Integrated Circuit (DevIC)*, 2025, pp. 287-291. doi: 10.1109/devic63749.2025.11012233

[3] Q. Lin, "Low-Power Full Adder Design and Development," in *Proc. 2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)*, 2023, pp. 226-229. doi: 10.1109/eiecs59936.2023.10435593

[4] R. R, R. J, V. R, S. S, "Design and Implementation of Approximate Truncated adder using kogge stone adder for low power applications," in *Proc. 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN)*, 2023, pp. 1-6. doi: 10.1109/vitecon58111.2023.10157681

[5] S. Seyedi, H. Abdoli, "Approximate Full-Adder, Full-Subtractor, and Full-Adder/Subtractor Circuits Based on QCA," in *Proc. 2025 29th International Computer Conference, Computer Society of Iran (CSICC)*, 2025, pp. 1-5. doi: 10.1109/csicc65765.2025.10967408

[6] K. Naseri, H. Jahanirad, "A Runtime Reconfigurable Exact-Approximate Full-Adder Design," in *Proc. 2024 6th Iranian International Conference on Microelectronics (IICM)*, 2024, pp. 1-5. doi: 10.1109/iicm65053.2024.10824335

[7] A. Mishra, K. Singh, "Low Power High Speed Full Adder Design Using CMOS Memristor Hybrid Circuits," in *Proc. 2022 2nd International Conference on Intelligent Technologies (CONIT)*, 2022, pp. 1-7. doi: 10.1109/conit55038.2022.9848115

[8] N. Prihatiningrum, Y. Seo, "PPAxA: Design of a High-Density, Low-Power Parallel Prefix Approximate Adder," *IEEE Embedded Systems Letters*, pp. 1-1, 2026. doi: 10.1109/les.2026.3698865

[9] S.S. Chiwande, P. Dakhole, "Design and Analysis of Low Power Full Adder using Reversible Logic," in *Proc. 2022 6th International Conference on Electronics, Communication and Aerospace Technology*, 2022, pp. 146-150. doi: 10.1109/iceca55336.2022.10009431

[10] B. V, D. N, G. S, K. S, "Low Power CMOS GDI Full-adder Design," in *Proc. 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2023, pp. 1946-1950. doi: 10.1109/icaccs57279.2023.10112885

[11] P. Raja, S. Jayanthi, M.M. Sajid, D. Praveen, A.S. Kannan, "CNTFET based Energy-efficient Approximate Full Adder for Low Power and High Speed Computing," in *Proc. 2026 International Conference on Smart Electronic*

Devices and Intelligent Systems (ICSEDIS), 2026, pp. 366-371. doi: 10.1109/icsedis68157.2026.11518416

[12] R. Singh, A. Kumar, "Design and analysis of a low power high speed full adder using 2×1 multiplexer," in *Proc. 2022 IEEE North Karnataka Subsection Flagship International Conference (NKCon)*, 2022, pp. 1-7. doi: 10.1109/nkcon56289.2022.10127080

[13] P. Yang, J. Lin, "A Low-Power Full Adder Design with Compact Layout Implemented in 16nm Technology," in *Proc. 2025 13th International Conference on Awareness Science and Technology (iCAST)*, 2025, pp. 1-5. doi: 10.1109/icast68191.2025.11300211

[14] Ujala, M. Kumar, "Low Power Hybrid Full Adder Design using MGDI Based XNOR Gate with Swing Restored Technique," in *Proc. 2025 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*, 2025, pp. 1-5. doi: 10.1109/wispnet64060.2025.11005232

[15] M.K. Peddachowdappa, V. D, "Low-Power High-Speed Approximate Ternary Half-Adder and Multiplier Circuit Design Using Carbon Nanotube Field-Effect Transistors," in *Proc. 2025 IEEE 4th International Conference on Smart Technologies for Power, Energy and Control (STPEC)*, 2025, pp. 1-5. doi: 10.1109/stpec66316.2025.11490596