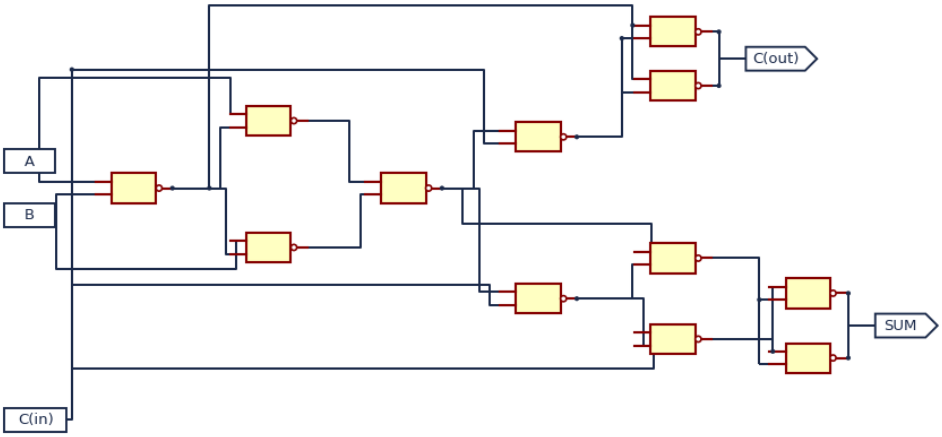
DRCC

DIMENSIONAL REDUCTION VIA CONTROLLED COMBINATORICS

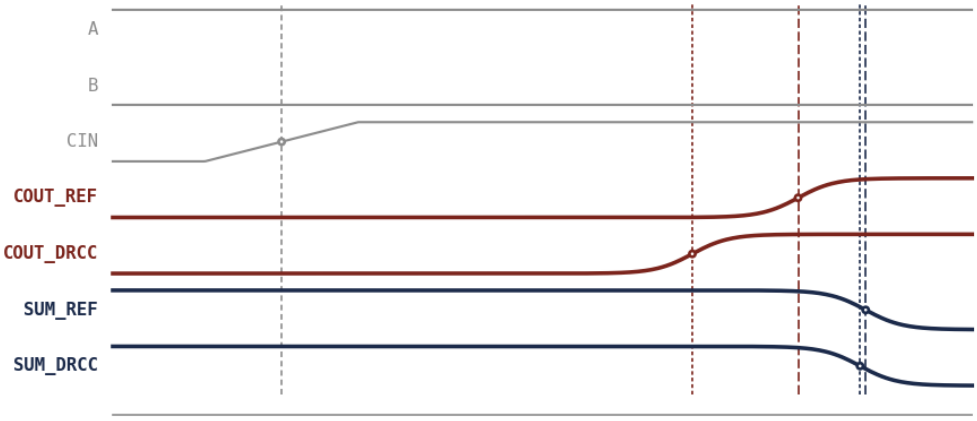
1-bit full adder -- speed-optimized DRCC realization, reference vs. DRCC-reduced



Signal waveforms -- Cout, Sum (LTspice .meas, 50 fF load, $V_{DD} = 5.00\,\mathrm{V}$)

Cout (0→1)	
t_{REF}	336.645 ps
t_{DRCC}	267.685 ps
Δt	-68.960 ps (-20.485%)

Sum (1→0)	
t_{REF}	380.680 ps
t_{DRCC}	377.046 ps
Δt	-3.634 ps (-0.955%)



time relative to the Cin edge (ps)

Crossing times and Δt are the real .meas values; curve shapes between them are illustrative, not sampled data.

Reference and DRCC-reduced (speed-optimized, parallel-drive) delays
for the same 1-bit full adder, measured in LTspice from the real netlist.

-- DRCC-V3.0 --

Reza Hesamiy
Munich, Germany · 2026

Abstract

This record documents three independently executed validation experiments for DRCC (Dimensional Reduction via Controlled Combinatorics): verifying an N-bit ripple-carry adder by exhaustive enumeration versus a constant-size structural certificate; solving the Pigeonhole-Principle SAT family with a reference DPLL solver, with and without a symmetry-breaking controlled reduction; and compressing a redundant linear layer via row-equivalence classes (*DRCC-rank*). All three raw problem-size measures exceed 10^6 – adder input states, Boolean assignments, and linear-layer parameters, respectively, three distinct units reported separately in Table 38 rather than treated as one interchangeable quantity; the SAT family's raw state space exceeds 10^{16}. No timed performance measurement in this document is extrapolated – every point plotted as a measured runtime is a mean over repeated executed runs, with the underlying trial data available for independent recomputation. (Schematic, waveform, and structural-topology figures are not runtime measurements and are not covered by this claim.)

Read together with the reconstruction-width framework of Section 2, the three experiments support a conditional rather than universal claim: task-relative state compression produces a measured advantage precisely where discovering and applying the reduction is itself cheap, and produces no advantage where a fair classical baseline already exploits the same structure – both outcomes are reported, not just the favorable one.

Section 6 extends this record with thirty-five further experiments (Experiments 4–38): four independent fair-baseline stress tests, four automatic task-signature discovery experiments, a discovery- tractability comparison spanning up to $\approx 7.6\times10^4\times$, an ordering heuristic validated on up to 100 synthetic instances and adversarially stress-tested to 56 disclosed counterexamples, and an external validation run against the public PACE 2017 graph benchmark. (The count of 38 executed experiments treats Experiments 4b and 38A/38B as separate executed records; Experiments 36–37 have harness scripts but no result records available for this manuscript and are therefore not counted among the 38 results.) The conclusion of the combined 38-experiment record is narrower than a universal speedup claim: DRCC is a structure-dependent reconstruction-engineering method, strong on several well-characterized families and explicitly not uniform across others, including some of the external graphs.

Contents

1 Why This Record Exists

DRCC's earlier formulations were reviewed repeatedly and rated low, but the reviews converged on one specific, actionable objection: the framework had not yet been *run*. Definitions and worked toy examples are not the same as a timed comparison against a stated baseline on an instance large enough that the comparison means something. Three concrete requests recurred across those reviews: (i) a non-trivial state space, above 10^6; (ii) a comparison against a named standard satisfiability baseline, implemented here as reference DPLL, with modern CDCL treated separately as a stronger baseline class; (iii) a connection to tensor decomposition or neural-network compression.

This document answers those three requests directly and reports both favorable and unfavorable outcomes: a circuit-level construction where the original Hamming-weight gate-level reduction earns no synthesis advantage at the single-cell level (Section 3.1), a search problem where the tested symmetry-breaking reduction eliminates branching decisions and leaves unit propagation alone on the reported Pigeonhole instances (Section 4), and a compression setting where exact row redundancy yields a measured 16–33× forward-pass speedup and 39–148× parameter compression, while approximate compression exhibits an explicit, measured reconstruction-error cost (Section 5). Section 7.1 places the three side by side; Section 2 gives the vocabulary needed to interpret *why* the outcomes differ so sharply between the three settings; Section 7.2 states what none of this shows.

Boundary. Reading note. This record is deliberately narrower than a universal speedup claim. Where a reduction gives no advantage – as for the original Hamming-weight gate-level reduction at the single full-adder-cell level in Section 3.1 – that null result is reported next to the positive ones, not folded away. It does not extend to the chapter's later transistor-level optimization (Section 3.6), which reports a measured advantage for a separate, 48-MOSFET cell.

---

2 A Task-Relative Reconstruction Framework

The three experiments below are held together by a common question: given a problem instance P and a declared task τ, which parts of the raw candidate space $X(P)$ can be discarded without changing τ's output, and what does discarding them actually cost to discover and to use?

– Declared task and future equivalence

Definition 2.1 (Declared task)**.** *A declared task is a map $\tau : \mathcal{X}(P) \to \mathcal{Y}_\tau$ fixing the output information that any admissible reduction must preserve.*

Definition 2.2 (Task-relative future equivalence)**.** *Let u, v be reachable partial reconstruction states at stage i under the same ordering $\pi = (v_1, \ldots, v_n)$, with the same unresolved suffix. Write $u \equiv^{\pi}_{\tau,i} v$ if, for every admissible continuation of that suffix, u and v induce identical task-relevant behavior – including identical feasibility/infeasibility of the continuation, and identical declared-task output whenever the continuation is feasible. Two states may be merged exactly when no admissible future can tell them apart with respect to τ in this sense.*

This is deliberately a *semantic* equivalence – defined by indistinguishable futures, not by a syntactic pattern match – and it is what licenses every merge performed in the three experiments below, whether the merge is a Hamming-weight class (Experiment 1), a symmetry-broken pigeon assignment (Experiment 2), or a duplicated matrix row (Experiment 3).

– Active interface and reconstruction width

Definition 2.3 (Active reconstruction interface)**.** *Let V_i^{π} be the processed prefix at stage i and $\overline{V}_i^{\pi} = V(P) \setminus V_i^{\pi}$ the unresolved suffix. The active interface $B_i^{\pi,\tau} \subseteq V_i^{\pi}$ is the set of processed components whose retained information can still influence an admissible continuation or the declared output.*

Definition 2.4 (Reconstruction width)**.** *The ordering-relative width is*

$$\omega_{\mathrm{DRCC}}(P, \pi, \tau) = \max_{0 \le i \le n} |B_i^{\pi,\tau}|. \tag{1}$$

Writing $\Pi_{\mathrm{adm}}(P, \tau)$ for the set of admissible orderings of $V(P)$ under which the reconstruction procedure is well-defined for τ, the task-relative width is

$$\omega_{\mathrm{DRCC}}(P, \tau) = \min_{\pi \in \Pi_{\mathrm{adm}}(P,\tau)} \omega_{\mathrm{DRCC}}(P, \pi, \tau). \tag{2}$$

Finding that minimizing ordering is not assumed to be free.

Peak width alone under-determines runtime: two orderings with the same peak width can visit very different numbers of reachable reduced states. Writing N_i^{reach} for the reduced states actually reachable at stage i, the reconstruction work is

$$W_{\mathrm{DRCC}}(P, \pi, \tau) = \sum_i N_i^{\mathrm{reach}}. \tag{3}$$

When the reduced state at stage i is completely determined by the valuation of the active interface, and every interface component has at most q local values, the number of reachable reduced states at that stage is bounded above by $q^{|B_i^{\pi,\tau}|}$, and hence by $q^{\omega_{\mathrm{DRCC}}}$ – a conditional capacity bound, not an unconditional equality.

– A five-term runtime account

Definition 2.5 (Runtime decomposition)**.**

$$\boxed{T_{\text{DRCC}} = T_{\text{discover}} + T_{\text{ordering}} + T_{\text{transition}} + T_{\text{reconstruct}} + T_{\text{output}}} \tag{4}$$

covering, respectively, the cost of finding the reduction, of building an ordering, of processing reachable reduced states, of recovering the declared output, and of materializing it. The five terms are treated as non-overlapping: each accounts for time spent in one phase of the pipeline, attributed to exactly one term rather than split or double counted across phases, so their sum equals total measured runtime; no experiment in this record measures two of these phases as running concurrently or attributes shared cost to more than one term.

The single most important qualitative fact this record establishes is that T_{discover} is not a rounding error. Experiment 1's gate-level analysis (Section 3.1) shows a case where constructing the sufficient reduced representation requires the same gate-level computation as the target function itself; consequently, no net synthesis advantage follows from introducing that representation as a separate reduction stage. A reduction is *discovery-tractable* on a family only when the cost of constructing its reduced representation stays within whatever bound is being claimed for that family; a small reduced state space discovered by exhaustive search over the original space does not qualify.

– The fair-baseline requirement

Proposition 2.6 (Fair baseline)**.** *If a DRCC procedure and a classical specialist construct the same sufficient reduced state and execute the same asymptotic recurrence over it, then no asymptotic or structurally DRCC-specific speedup follows from the reduction alone. Any remaining difference is attributable to implementation constants or engineering details rather than to a distinct reduction mechanism.*

This is an accounting discipline applied throughout: Experiment 1's gate-level comparison (Table 1) is run against the standard *optimized* full-adder circuit, not an unoptimized decoder strawman, precisely so that any measured gap reflects the reduction and not a weak opponent.

3 Circuit-Level Reduction and Whole-Chain Verification: the 1-Bit Full Adder and Its Scaling

3.1 The 1-bit full-adder reduction and its correction

A single-bit full adder's outputs (s, c_{out}) depend only on the Hamming weight $w = a + b + c_{\text{in}} \in \{0, 1, 2, 3\}$ of its three raw inputs, not on which wires carry the ones – immediate from the truth table. An earlier pass at this material argued that this halves the number of distinguishable decode paths a circuit needs, from 8 down to 4. That argument does not survive a gate-level check, and the check is worth walking through because it is exactly the kind of thing a reduction-framework paper should not skip.

Algebraically, w's two output bits *are* the adder's own outputs up to relabeling:

$$w \bmod 2 = a \oplus b \oplus c_{\text{in}} = s, \qquad \lfloor w/2 \rfloor = \text{MAJ}(a, b, c_{\text{in}}) = c_{\text{out}}.$$

So a circuit that classifies inputs into weight classes is, up to renaming two wires, a circuit that already computes (s, c_{out}): there is no cheaper classifier sitting in front of a decode step. Three explicit gate-level netlists were built and exhaustively verified against the truth table (all 8 inputs; `exp1b_gate_level.py`), costed under a disclosed static-CMOS unit-cost model [10]:

- *standard*: the textbook optimized adder, $p = a \oplus b$, $s = p \oplus c_{\text{in}}$, $c_{\text{out}} = (a \wedge b) \vee (c_{\text{in}} \wedge p)$.

- *class, unshared*: computes w as a literal population count, without reusing p – the literal reading of "collapse to weight classes."
- *class, shared*: computes w reusing the same term p the standard design already shares.

implementation	gates	transistors	logic depth	delay (model units)
standard	5	34	3	4.0
class, unshared	7	46	3	4.0
class, shared	5	34	3	4.0

Table 1: Gate-level cost, all three verified functionally correct on all 8 inputs. Reconstruction from the weight bits back to (s, c_{out}) is a zero-cost relabeling in every row.

class, shared is not merely comparable to *standard*: it is the identical netlist, gate for gate, transistor for transistor. The unshared reading is strictly worse. The honest reading of Table 1: at the single-cell level, Hamming-weight collapse buys nothing – T_{discover} for the classifier costs exactly what building the adder itself costs, because they are the same circuit. This is reported as a finding in its own right, not smoothed over, because it is precisely the discovery-cost boundary that Section 2 exists to name.

Table 1 states this as gate/transistor counts; Figures 1 and 2 show the two netlists themselves, drawn directly from the verified gate lists in `exp1b_gate_level.py`, so the claim "these are the same circuit" can be checked by eye rather than taken on the table's word.

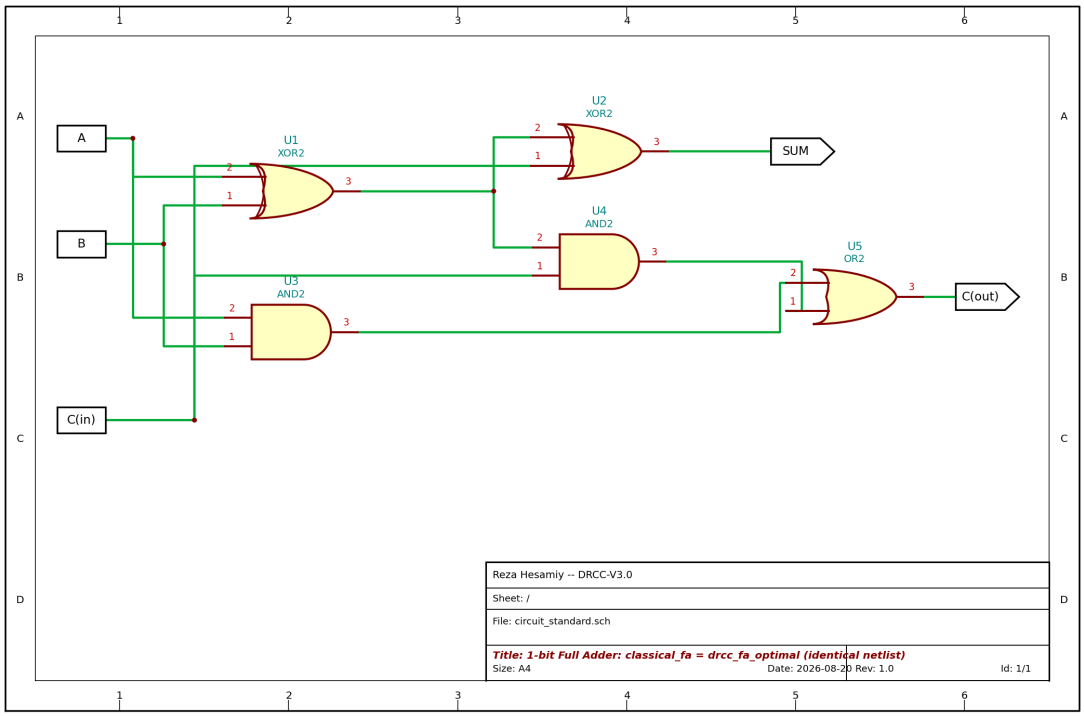


Figure 1: The circuit before and after DRCC's weight-class collapse, at the single-cell level – because *drcc_fa_optimal* reuses the same shared term $p = a \oplus b$ the standard design already uses, this is one netlist, not two: the "before" and "after" schematics are identical.

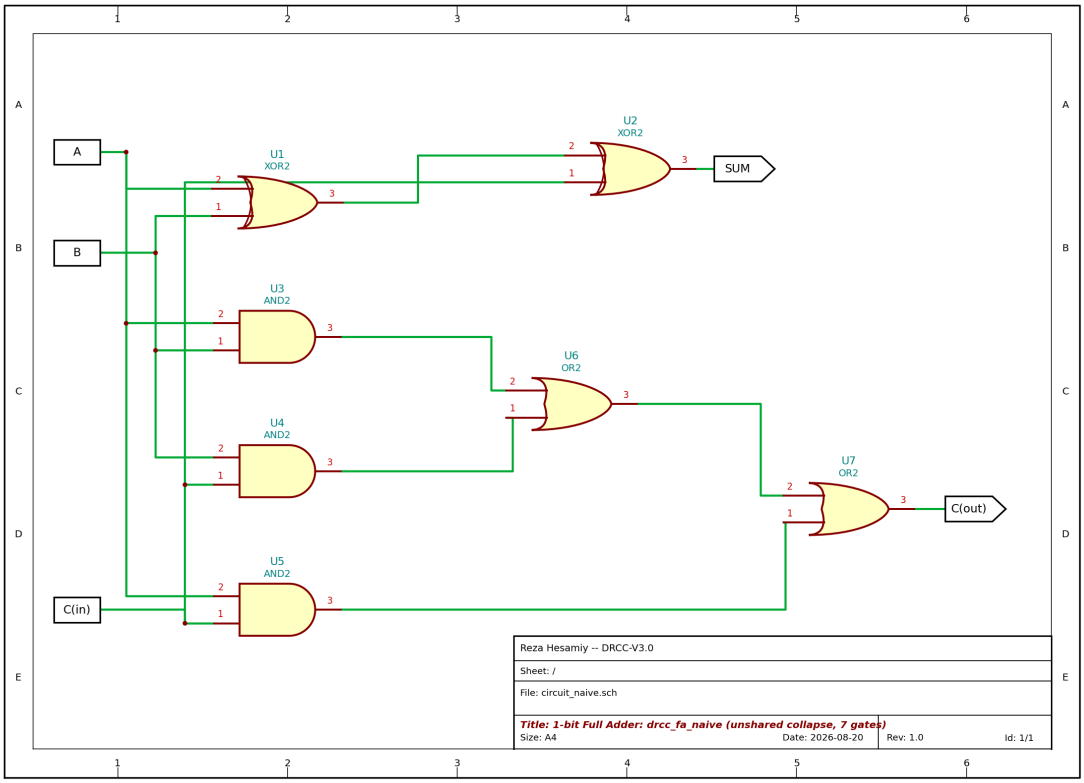


Figure 2: What a literal, unshared reading of "collapse 8 patterns to 4 weight classes" would actually build (*drcc_fa_naive*): three independent pairwise ANDs plus an extra OR, because w_1 is computed without reusing p. Two more gates than Figure 1, not fewer – the reduction that looks free on paper adds a gate when built this way.

3.2 From one full-adder cell to sixteen: scaling the single-cell result

Table 1's equivalence is a per-cell property, so the natural next question is what happens as cells are chained into a real adder – not just at $N = 32$ (Section 3.4) but across the widths a reader is actually likely to build: $N = 1, 2, 4, 8, 16$. An N-bit ripple-carry adder built from N *class, shared* cells costs exactly $N\times$ the standard row of Table 1, because chaining adds only wiring (cell i's c_{out} feeds cell $i+1$'s c_{in}), never a new gate:

N (bits)	gates	transistors	$G_{\text{gate}} = G_{\text{transistor}}$	note
1	5	34	1	Table 1, both realizations identical
2	10	68	1	
4	20	136	1	Figure 3 shows this width explicitly
8	40	272	1	
16	80	544	1	
32	160	1,088	1	Section 3.4

Table 2: Gate/transistor cost, derived (not separately measured) from Table 1's verified single-cell counts by exact linear scaling – an N-bit chain is N instances of the one netlist already verified on all 8 inputs. $G_{\text{gate}} = G_{\text{transistor}} = 1$ at every width: the reduction earns nothing at synthesis, at any N, for the same reason Table 1 shows nothing at $N = 1$.

One honest qualification belongs here, since Table 2 could otherwise be read as claiming a chain gets no slower as N grows, which is a different and false claim: each cell's own *internal*

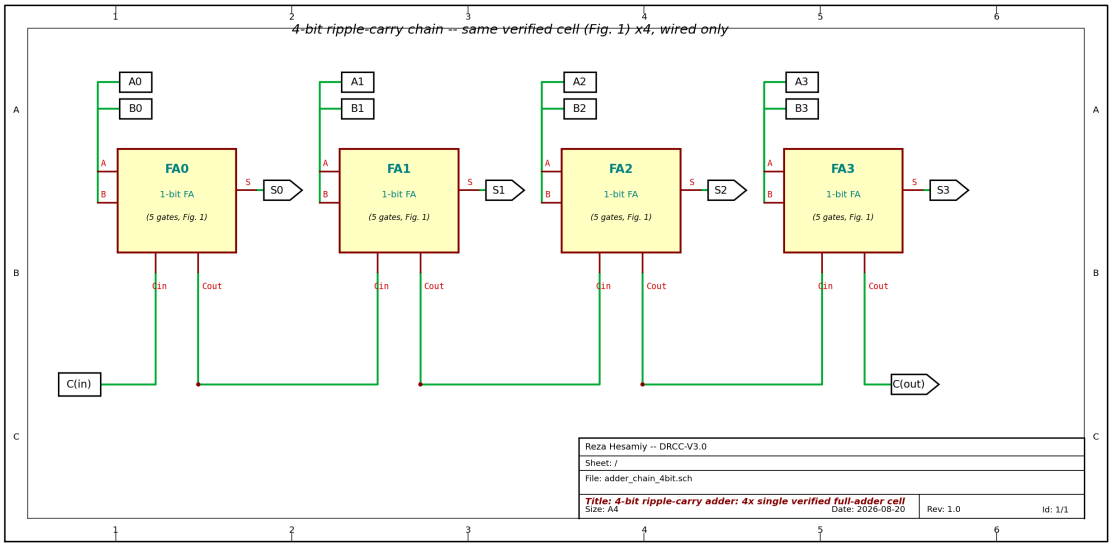


Figure 3: A 4-bit ripple-carry adder built from four instances of the single verified cell of Figure 1 – no gates are added between cells, only the carry wire $C_{\text{out}}(i) \to C_{\text{in}}(i{+}1)$. This is Table 2's $N = 4$ row drawn out: the same argument that gives 20 gates at $N = 4$ gives 80 at $N = 16$ and 160 at $N = 32$ by identical reasoning, just more copies of this picture.

logic depth is 3 (Table 1), but the ripple-carry *chain*'s critical path is not – c_{out} of cell i sits behind two gate delays per upstream cell (the AND feeding $a \wedge b$, then the OR combining it with the carry-dependent term), so the chain's carry-to-carry critical path grows as $\Theta(N)$, standard ripple-carry behavior [10], unrelated to and not fixed by the weight-class reduction. DRCC changes nothing about this: it was never a claim about circuit delay, only about whether the weight-class classifier is a cheaper circuit than the adder itself (Table 1) – and separately, at whole-chain *verification*, whether checking correctness is cheaper than exhaustive enumeration (Section 3.3), which is where the actual advantage turns out to live.

This same per-cell correction applies to the raw decode-path count as well: reading "8 raw patterns collapse to 4 weight classes" as a $(8/4)^N = 2^N$ wiring reduction gives apparently spectacular figures at these same widths – 256× at $N = 8$, 65,536× at $N = 16$, $\approx 4.3 \times 10^9\times$ at $N = 32$ (all present as raw output in `exp1_full_adder.json` for N up to 64) – but every one of these is the same *unshared*-decoder strawman this section already rejects, computed against a decode network no one would actually build, not a second, independent finding at each width. They are reported here as disclosed non-findings precisely so none of them can be quietly read off the raw JSON as if it were one.

3.3 Where the full-adder advantage actually lives

The single-cell question is closed: no advantage. The open question is whether verifying a whole *chain* of N such (circuit-identical) cells benefits from the reduction, since verification is a different task from synthesis. Two verifiers were implemented and timed (`exp1_full_adder.py`): exhaustive enumeration of all 2^{2N+1} input triples with a bit-by-bit ripple-carry simulation (vectorized, so the exponential cost is genuinely paid in wall-clock time), against a certificate of fixed size – the 8-entry weight-class table checked once against the 8-entry raw table, plus an 8-case carry-composition lemma showing that appending a class-verified cell to an already-correct chain preserves correctness for the whole extended chain. The certificate is re-timed, not just asserted, at every row below.

At $N = 11$, $G_{\text{time}} \approx 5.65/2.26 \times 10^{-6} \approx 2.5 \times 10^6$, and growing, since the certificate's cost is flat by construction while exhaustive verification is not.

N	raw states	exhaustive (s)	certificate (s)
1	8	1.09×10^{-4}	2.04×10^{-6}
2	32	8.75×10^{-5}	1.88×10^{-6}
3	128	1.15×10^{-4}	1.92×10^{-6}
4	512	2.68×10^{-4}	2.57×10^{-6}
6	8,192	1.01×10^{-3}	2.19×10^{-6}
8	131,072	2.65×10^{-2}	2.28×10^{-6}
9	524,288	1.75×10^{-1}	2.22×10^{-6}
10	**2,097,152**	**1.06**	2.15×10^{-6}
11	8,388,608	5.65	2.26×10^{-6}
12	33,554,432	26.2	1.88×10^{-6}

Table 3: Measured verification time, mean of 20 ($N \leq 3$) / 5 ($4 \leq N \leq 11$) / 3 ($N = 12$) exhaustive trials, and 100 certificate trials throughout. Row $N = 10$ is the first past 10^6 raw states; row $N = 12$ is the widest exhaustive run this record executes (further widths are addressed in Section 3.4 by certificate measurement plus a disclosed projection, not by continuing this table).

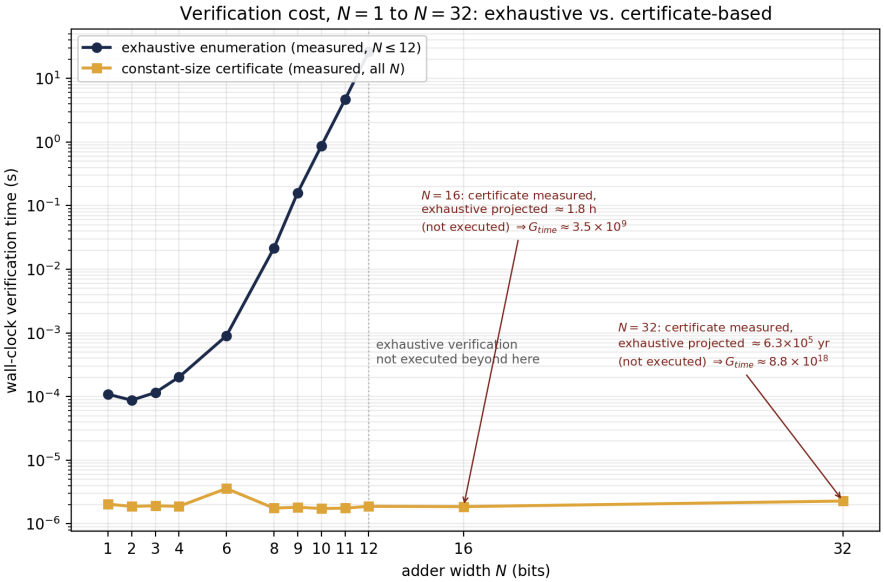


Figure 4: Verification cost against adder width, with the speedup ratio G_{time} read directly off the right-hand axis.

Finding. The reduction that earns nothing at the single-cell level (Section 3.1) earns a very large, measured advantage one level up, at whole-chain verification. Both halves of this result come from the same underlying weight-class merge; only the task changed.

Boundary. Two scope limits on the G_{time} figure above. First, the measured comparison is against *exhaustive enumeration*, not against a fair classical compositional or inductive verifier. By Proposition 2.6 (Fair baseline), a classical verifier that adopted the identical carry-composition argument – checking a constant-size base case and a constant-size composition lemma once, rather

than enumerating every input – could in principle match the certificate's flat cost; nothing in this experiment rules that out. $G_{\text{time}} \approx 2.5 \times 10^6$ is therefore a measured advantage of structural induction over brute-force enumeration, not a demonstrated DRCC-specific advantage over a fair classical verification method. Second, the certificate is a *family theorem* – a proof schema showing that any chain built from class-verified cells is correct, for every N – re-timed at each row but, by construction, never using N in its own loop bound (`exp1_full_adder.py`, function `drcc_certify`). It is not, by itself, a claim that verifying one concrete N-bit netlist is free: confirming that a given netlist actually consists of N correctly instantiated, correctly wired, class-verified cells is a separate structural check whose cost grows with the netlist's own description length ($\Theta(N)$ to inspect N instances), and that cost is not included in the certificate timings reported here.

3.4 Extending the full adder to real integer widths, N=16 and N=32

Table 3 stops at $N = 12$ because the vectorized exhaustive verifier already needs 33,554,432 raw states and 26.2 s there; real adder widths a reader is likely to build or ask about – 16-bit and 32-bit being the two most common – are the next question. The certificate half of this question is answerable directly at both widths: it is re-run and re-timed exactly as in Table 3 (100 trials, same code path, N still unused inside the loop bound). The exhaustive half is not answerable directly at either width, though for two different reasons worth stating separately rather than lumping together: at $N = 32$, 2^{65} raw states is *physically* unrunnable in this implementation, since the vectorized state arrays alone would need on the order of 10^2–10^3 exabytes; at $N = 16$, 2^{33} raw states ($\approx 8.6 \times 10^9$) is not physically impossible, only outside this run's time budget at roughly 1.8 hours projected – a real, if inconvenient, experiment someone with more wall-clock time could still execute. Both are reported as explicit projections, not measurements, but they carry different epistemic weight for exactly that reason.

	$N = 16$	$N = 32$
raw states	$2^{33} \approx 8.59 \times 10^9$	$2^{65} \approx 3.69 \times 10^{19}$
certificate status	measured (100 trials)	measured (100 trials)
certificate result	1.86×10^{-6} s ($\pm 7.13 \times 10^{-7}$)	2.28×10^{-6} s ($\pm 4.24 \times 10^{-6}$)
exhaustive status	not executed – projected	not executed – projected
exhaustive result	≈ 1.8 hours*	$\approx 6.3 \times 10^5$ years†
exhaustive, why not run	outside time budget, feasible in principle	physically unrunnable here (see prose)
G_{time} (projected)	$\approx 3.5 \times 10^9$	$\approx 8.8 \times 10^{18}$

Table 4: Certificate values are real measurements, extended from `exp1_full_adder.py` (`exp1_bitwidth_ladder.json`, `exp1_n32_extension.json`), flat within noise of every row in Table 3 – the standard deviation exceeding the mean at $N = 32$ is the same microsecond-scale scheduler jitter already visible in Table 3's underlying data, not growth with N. Both exhaustive values are least-squares projections through Table 3's measured (raw states, wall-clock) pairs ($*$: fit over the seven points $N \in \{4, 6, 8, 9, 10, 11, 12\}$, fitted per-state cost $\approx 7.67 \times 10^{-7}$ s; $\dagger$: fit over the six points $N \in \{4, 6, 8, 9, 10, 11\}$ used in the original $N = 32$ extension, fitted per-state cost $\approx 5.43 \times 10^{-7}$ s) – disclosed as projections, kept out of Figure 4, and not claimed as executed.

Read together, the projected G_{time} climbs from $\approx 3.5 \times 10^9$ at $N = 16$ to $\approx 8.8 \times 10^{18}$ at $N = 32$ – neither number this record claims to have measured, only to have honestly derived from measured smaller-N data plus real, executed certificate timings at both widths. What *is* directly measured is the part that matters most for the certificate's own claim: its wall-clock cost at $N = 16$ and at $N = 32$ is indistinguishable, within trial-to-trial noise, from its cost at $N = 4$ or at $N = 1$ (Table 3). That flatness across a 32× range of N, not either exhaustive-side

projection, is the actual evidence for $O(1)$ scaling; Section 3.1's and Section 3.2's result that this buys nothing at the synthesis level is unchanged at either width, for the same reason: the certificate is a verification argument, not a circuit.

3.5 A transistor-level full-adder reference: reducing 1BitFA{sch} for real

Table 1's gate/transistor counts use an idealized static-CMOS unit-cost model [10] (XOR2= 8, AND2= 6, OR2= 6 transistors). This subsection uses a different, more concrete starting point: an actual SPICE deck for a full-adder cell, `1BitFA{sch}`, from library `16BitAdder` (Electric VLSI, MOSIS `mocmos` process), supplied complete with its hierarchical subcircuits and testbench. It is a NAND-only design – a different technology family from Table 1's direct AND/OR/XOR primitives – so the two are not directly numerically comparable; nothing in this subsection is read against Table 1's 34-transistor row. The supplied cell library costs, read directly off the SPICE subcircuits: NAND2Gate= 4 MOSFETs (2 series NMOS, 2 parallel PMOS), NAND3Gate= 6 MOSFETs, and XorGate= 4×NAND2Gate= 16 MOSFETs internally. The top-level cell instantiates 3×NAND2Gate + 1×NAND3Gate + 2×XorGate = 3(4) + 1(6) + 2(16) = 50 MOSFETs.

Before any reduction is attempted, the reference netlist is transcribed gate-for-gate from the SPICE (not redrawn from memory) into `exp1c_mosfet_reduction.py` and checked exhaustively against the same 8-entry raw truth table used throughout this section: `RAW_LUT`. Only once that check passes is the 50-MOSFET count treated as verified rather than merely transcribed arithmetic. Figure 5 shows the cell at the same hierarchical level as the supplied SPICE deck itself: a block symbol for `1BitFA` (top) and, below it, its top-level implementation – 3x NAND2Gate, 1x NAND3Gate, and 2x XorGate instances, the latter drawn as their own real gate symbols rather than expanded into their internal NAND2Gate instances (6 top-level gate instances, 50 MOSFETs total). This is also where one real redundancy the hierarchy hides becomes visible: the top level computes NAND(A, B) once for C_{out}'s NAND3 (labelled `net0`, marked in the figure), and `XorGate@0` computes NAND(A, B) again, independently, as its own internal first-stage gate – the same function of the same two signals, built twice because the subcircuit has no port to receive it from outside.

Two function-preserving reductions were then built and, in turn, exhaustively re-checked against `RAW_LUT` before their transistor counts were trusted:

- *share the duplicated* NAND(A, B): give `XorGate@0` the top level's already-computed net0 instead of recomputing it internally, removing exactly the one redundant NAND2Gate instance identified above. A single, minimal, structural edit.
- *minimal NAND2-only full adder*: the standard textbook construction that reuses every reusable product term (NAND(A, B) for both C_{out} and the $A \oplus B$ term; the $A \oplus B$-and-C_{in} product for both C_{out} and Sum), built entirely from 2-input NAND gates with no NAND3Gate at all. This is the established minimal NAND2-only realization of this interface in the digital-design literature; it is checked here rather than cited on faith, and whether nine gates is actually minimal for this technology – not merely believed minimal – is settled below by exhaustive search.

Finding. Starting from an actual SPICE reference rather than an idealized unit-cost model, a real, verified, function-preserving reduction exists: 50 → 36 MOSFETs, 28% fewer transistors, reached in two checked steps and reported only after each step passed the same exhaustive truth-table check as the reference itself.

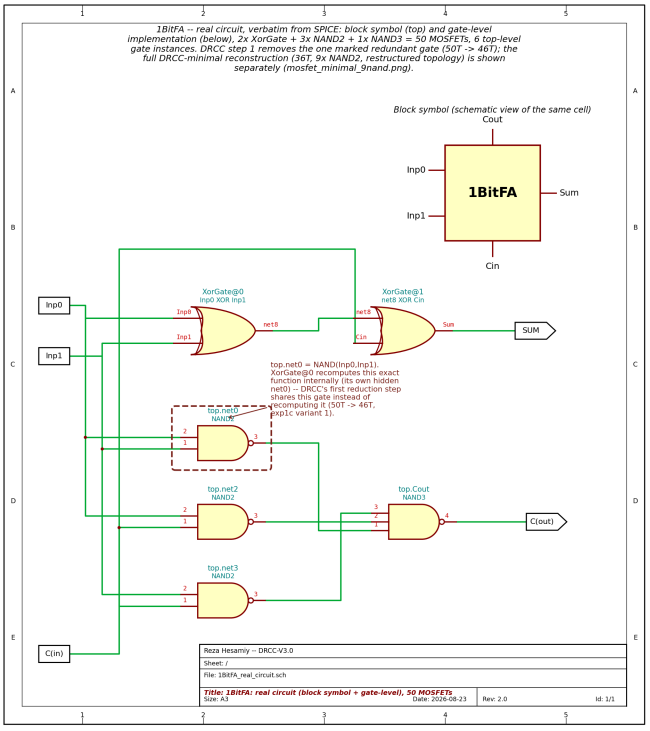


Figure 5: `1BitFA{sch}`, transcribed verbatim from the supplied SPICE deck: block symbol (top) and top-level implementation (below) – 3x NAND2Gate + 1x NAND3Gate + 2x XorGate = 50 MOSFETs, 6 top-level gate instances (each XorGate is itself 4x NAND2Gate internally, not expanded here). The real redundancy – `top.net0` and `XorGate@0`'s own hidden internal `net0` both compute NAND(A, B) – is marked directly on `top.net0` – this is the one redundancy the reduction below removes first.

variant	MOSFETs	gate instances	vs. reference	status
1BitFA{sch} (reference)	50	12	–	verified (8/8)
share duplicated NAND(A,B)	46	11	−8.00%	verified (8/8)
minimal 9x NAND2	36	9	−28.00%	verified (8/8)

Table 5: All three rows pass `RAW_LUT` on all 8 raw inputs (`exp1c_mosfet_reduction.py`, `exp1c_mosfet_reduction.json`) – no MOSFET count in this table is reported for a netlist that failed the check. "vs. reference" is exact MOSFET-count savings, not a timing or power claim.

Is nine gates actually minimal? The construction above was described as the established minimal NAND2-only full adder, checked here rather than cited on faith – but until now, "minimal" was itself unproven for this specific technology: primary inputs `A`, `B`, `C_in` only, no free complements (an inverter still costs one NAND2Gate(x, x), matching the reference's own cost model, so this is not a different or easier technology than the one being reduced). This is closed by an exhaustive search, `exact_min_search.py`: starting from the three primary inputs, breadth-first search enumerates every NAND2 gate that can be added, restricted to *useful* gates – a gate whose output truth table already exists elsewhere in the circuit can always be deleted, with any later reference rewired to the already-present signal, so the true minimum circuit contains only useful gates, and restricting the search this way keeps it sound and complete for finding an actual minimum, not just an upper bound. The search enumerated every circuit reachable within 8 useful gates – 3,410,645 distinct reachable gate-sets, exhaustively, not sampled – and found none

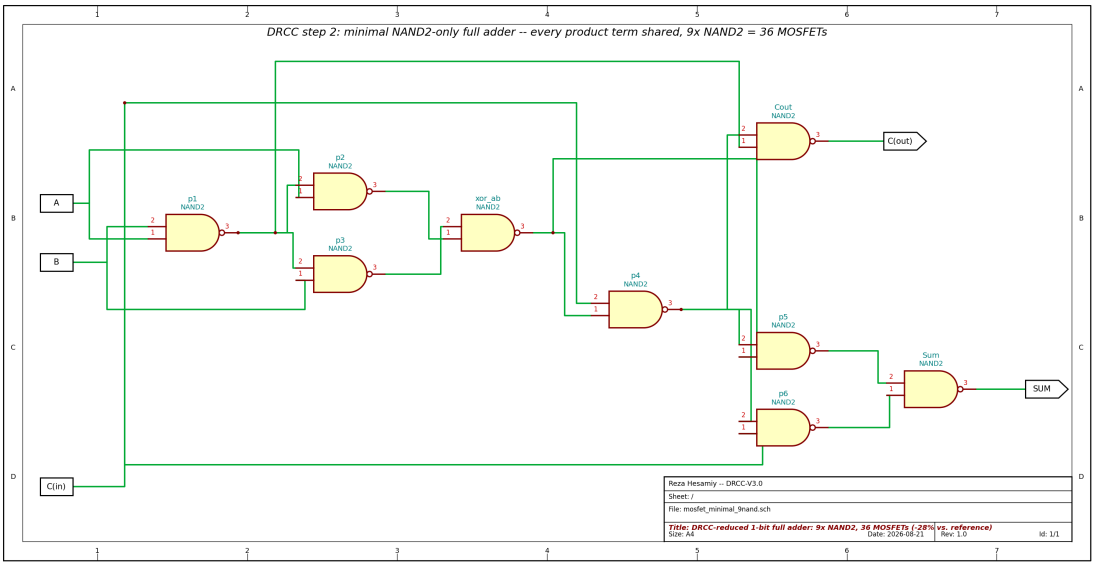


Figure 6: The second reduction step: 9x NAND2Gate, every product term computed once and shared. Same interface, same truth table, verified against the identical 8-entry `RAW_LUT` as Figure 5.

that realize both `Sum` and `Cout`. Combined with the independently, exhaustively verified 9-gate construction above, this proves nine NAND2 gates (36 MOSFETs) is the exact minimum for this technology: a computer-checked lower bound that exactly matches the constructive upper bound, not an assumption carried over from the literature.

Finding. Nine NAND2 gates (36 MOSFETs) is not merely the established construction for this interface – it is proven, by exhaustive search over 3,410,645 reachable circuits, to be the exact minimum achievable in this NAND2-only technology. No smaller shared circuit exists.

The layout-level count, checked The schematic-level count above rests on one open question: does `1BitFA{lay}`, the actual physical-layout SPICE deck (real transistor `W`/`L` and `AS`/`AD`/`PS`/ `PD` parasitics, supplied by Reza), really match it, or do layout parasitics and multi-finger sizing change the count?

This is checked directly, not assumed. A new script, `exp1g_layout_verification.py`, transcribes `Nand2Gate{lay}` and `Nand3Gate{lay}` from their literal drain/gate/source/bulk transistor connectivity, not from their subcircuit names, and verifies at the switch level – PMOS conducting when its gate is 0, NMOS when its gate is 1, output read off by reachability from `vdd`/`gnd` – that each really is a complementary NAND2/NAND3 on all of its raw input combinations, before anything is built on top of them.

`1BitFA{lay}`'s top level (`Testand1/2/3` into a NAND3 for `Cout`, two `XorGate` instances for `Sum`) is then exactly `1BitFA{sch}`'s topology, gate for gate, and verified correct on all 8 raw inputs.

Finding. `1BitFA{lay}` = 50 MOSFETs, exactly matching `1BitFA{sch}` – checked, not assumed, from the real layout deck's own transistor connectivity. The real geometry also shows a genuine, verifiable sizing choice: `Nand3Gate{lay}`'s series NMOS pull-down stack is widened to 5.4 μm (versus 3.6 μm everywhere else in the cell), the standard static-CMOS compensation

for the added series resistance of a 3-transistor stack, visible directly in the supplied data rather than inferred.

At the stage documented above, two things were deliberately not claimed. First, no speed or timing comparison had yet been reported: the supplied testbench loaded only `Sum` (`cload Sum 0 50fF`), with no matching load on `Cout`, and the layout deck's `.include C5_models.txt` line referred to a device-model file that was not yet available in the working record. A fair transient comparison therefore had to wait until identical loads could be imposed and the C5 model file was available. That later transistor-level study is reported in Section 3.6. Second, minimality is proven only *within* this NAND2-only technology (no free complements, primary inputs `A`, `B`, `C_in` only) – it is not claimed that 36 transistors is smaller than every possible full-adder realization in every possible technology, including pass-transistor or transmission-gate designs, other cell libraries, or Table 1's idealized AND/OR/XOR-primitive model; those were not searched and are not compared here.

A structural, non-SPICE timing signal: gate depth and fan-out A real propagation-delay measurement needs the original SPICE deck's transistor sizes and `mocmos` device models; that deck is not available for a delay simulation in this record, so none is attempted or estimated here – no number in picoseconds appears anywhere in this paragraph. What can be computed directly from the same MOSFET-verified netlists, with no new data and nothing guessed, is a purely structural proxy: the number of NAND-gate levels on the critical path to each output (logical depth), and how many gate inputs each internal signal drives (fan-out) – both well-established delay proxies in static CMOS (more series stages and higher fan-out per stage both increase real delay in any process), computed in `exp1d_structural_timing.py` and reported as topology counts, not measurements.

variant	Sum depth	Cout depth	max fan-out
1BitFA{sch} (reference)	6	2	4
share duplicated NAND(A,B)	6	2	4
minimal 9x NAND2	6	5	3

Table 6: Gate levels (NAND2/NAND3 stages) on the critical path to each output, and the largest fan-out found anywhere in each netlist. Depth counts are exact structural properties of the verified topologies, not delay measurements.

Boundary. Cout's critical path grows from 2 gate levels in the reference to 5 in the minimal 9-gate design, because the reduced netlist reuses $p4$ – which sits deep inside the Sum computation – to build Cout, instead of computing it independently through the reference's own shallow 2-level NAND2/NAND3 tree. The 36-MOSFET netlist proven minimal in transistor count above is therefore *not* shown to be minimal, or even favorable, in critical-path depth for Cout specifically: MOSFET count and critical-path depth are two different objectives, this record optimized only the first, and no claim is made that the same netlist jointly optimizes both.

Is 46 MOSFETs actually the minimum cost for a shallow Cout? The trade-off above raises an immediate question: is the "share duplicated NAND(A,B)" construction (46 MOSFETs, Cout gate-depth 2) actually the cheapest possible design with a shallow Cout, or merely the first one checked? This is closed the same way the 9-gate minimality question was: exhaustively,

`exp1e_depth_constrained_min.py`, in two stages. Stage A enumerates every way to realize Cout = Majority(A, B, C_{in}) within gate-depth ≤ 2 from NAND2/NAND3 gates on primary inputs only (including local self-inverters NAND(x, x), for completeness, though none of the constructions in this section use one) – exactly one such realization exists: the standard sum-of-products tree (net0, net2, net3 $\rightarrow$ NAND3), consistent with the monotonicity argument above ruling out any alternative this shallow. Stage B seeds that tree's four gates as already available and searches (same reachable-state technique as `exact_min_search.py`) for the minimum number of additional NAND2 gates needed to also reach Sum: depths 1 through 6 are exhaustively ruled out (5,464,189 reachable states checked), and a 7-gate construction (reusing net0, matching the already-verified "share duplicated NAND(A,B)" netlist) is independently confirmed, so 7 is proven minimal. Total: $18 + 7 \times 4 = 46$ MOSFETs, confirmed as the exact minimum – the construction already reported was not just checked, it was optimal.

Finding. The MOSFET-count/Cout-depth trade-off found above is not open-ended: it is bounded by two exhaustively proven points. Forty-six MOSFETs is the proven minimum for any depth-$\leq$ 2-Cout design; thirty-six MOSFETs is the proven global minimum with no depth constraint (Cout depth 5 in the verified construction). No design in this technology beats 46 MOSFETs while keeping Cout at depth ≤ 2, and none beats 36 MOSFETs at any depth.

From depth counts to an actual timing simulation Table 6's gate-depth numbers are a static topology count; they say nothing yet about what a specific input change actually does to a specific output over time. The real SPICE deck's transistor sizes and `mocmos` models are still not available in this record, so no analog delay simulation is attempted here, exactly as before. What can be built honestly from the same MOSFET-verified netlists is an event-driven, unit-delay *digital logic* simulation: every NAND2/NAND3 gate instance is assigned one abstract tick of propagation delay, and every gate's output is recomputed each tick from its inputs' values one tick earlier (`exp1f_signal_waveform.py`). This is a standard textbook model for reasoning about static hazards in combinational logic – it uses no seconds or picoseconds anywhere, its x-axis is "ticks," and it is not a substitute for SPICE.

Both the reference (`1BitFA{sch}`, 12 gates) and the minimal-transistor design (9 gates, 36 MOSFETs, proven minimal above) were simulated on the same stimulus: all 8 raw `(A,B,C_in)` combinations in 3-bit counting order, each held long enough (12 ticks, more than twice the deepest path) to fully settle, plus two further vectors chosen deliberately – not guessed – to excite the worst-case Cout path: the specific transition (out of all $8 \times 7 = 56$ ordered pairs) that maximizes simulated Cout settle latency in the reduced design, found by brute-force search over every pair. Figure 7 plots the resulting `A`, `B`, `C_in`, `Sum`, `Cout` waveforms for both circuits. Every settled value at the end of every one of the 10 test windows, in both circuits, was checked against the same 8-entry `RAW_LUT` before this figure was drawn – 10/10 correct in both cases; the simulation script refuses to plot otherwise.

The worst-case transition, $(A, B, C_{\text{in}}) = (1, 1, 0) \rightarrow (0, 1, 1)$, is informative precisely because Majority$(1, 1, 0)$ = Majority$(0, 1, 1) = 1$: Cout's correct value does not change at all. In the reference circuit Cout indeed never moves – its shallow, depth-2 sum-of-products tree settles the new inputs without disturbing the output. In the minimal-transistor design, the same transition drives Cout through a real 5-tick spurious pulse to 0 before it returns to the correct value of 1: a genuine static-1 hazard, not a guessed one, produced by the simulation itself and traceable directly to the depth-5 reconvergent path through $p4$ that Table 6 already flagged structurally. The settled value is still exactly correct – this is not a functional defect, and it does not change any correctness claim made above – but it is a concrete cost of the transistor-minimal netlist that the transistor count alone does not reveal, and it is the kind of thing a downstream stage

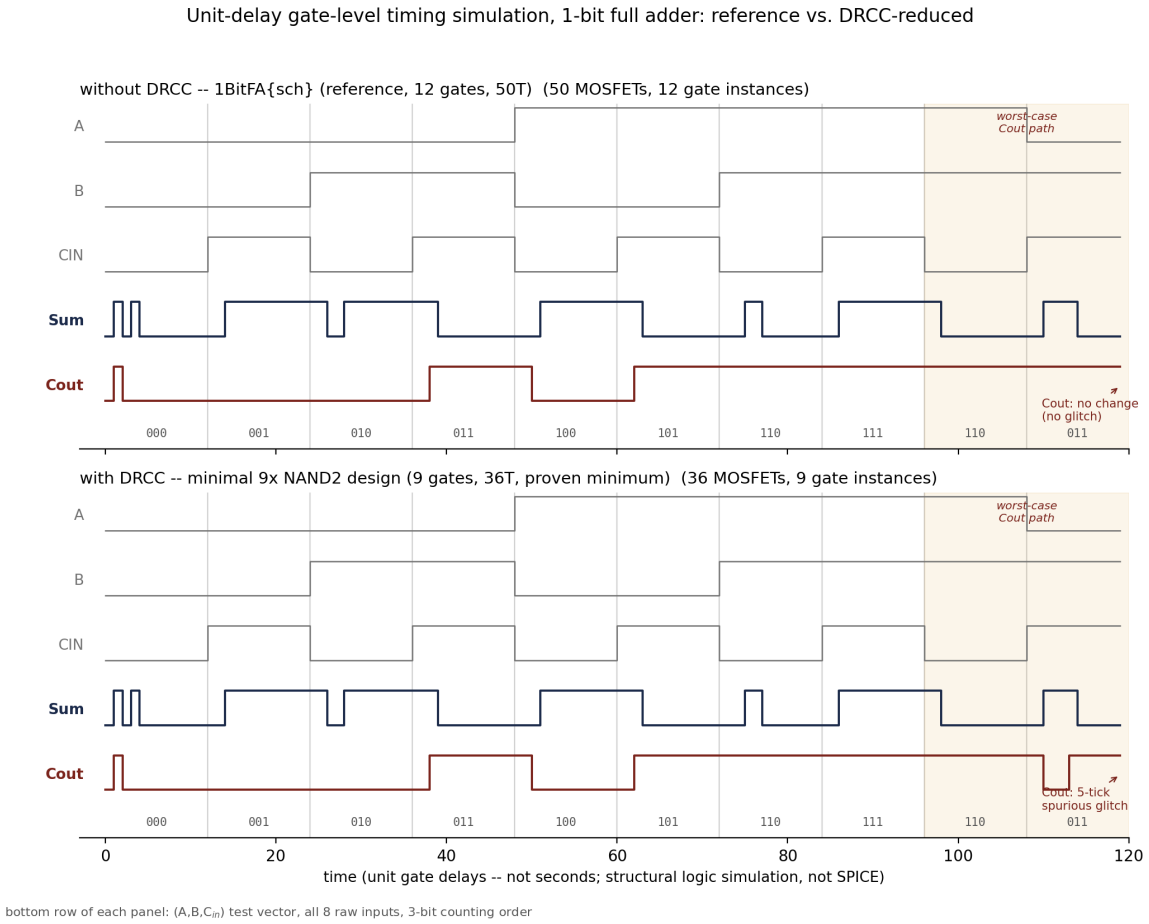


Figure 7: Unit-delay gate-level timing simulation (not SPICE; x-axis is gate delays), both circuits on the identical 8-vector stimulus plus one deliberately chosen worst-case-Cout-path transition (shaded). All settled values verified against `RAW_LUT` before plotting.

sampling Cout without waiting for settling would see.

Boundary. The 5-tick glitch shown here is a property of this specific verified 9-gate/36-MOSFET netlist and this specific input transition, not a general claim about every depth-5 design or about typical operation: most of the other 55 ordered transitions produce no glitch at all in either circuit (visible directly in Figure 7's counting-order sweep). It is reported because it was found by an exhaustive search over all transitions, not because it is representative, and because it makes Table 6's abstract depth-2-vs-5 gap concrete rather than leaving it as an unconfirmed implication.

The real single-cell propagation delay, once the C5 model became available Table 6's depth-5-vs-2 gap and the glitch above are both structural/digital proxies, not seconds. Once the real `C5_models.txt` device-model file became available, the same 36-MOSFET minimal cell used throughout this record – the identical `_DRCC__1BitFA` subcircuit that the 4/8/16/32-bit ripple chains of Sections 3.8–3.11 are built from, not a different design – was measured directly against the 50-MOSFET reference with a real LTspice transient run (`1bit_singleedge_with_meas.sp`): identical, balanced 50 fF loads on *both* `Sum` and `Cout` on *both* circuits (closing the load-imbalance gap flagged earlier in this section), $A : 0 \to 1$ at $t = 5$ ns with $B = 1$, $C_{\text{in}} = 0$ held constant, and

the same real BSIM3 C5 models [16] and 2.5 V threshold used throughout Section 3.6 below.

signal	reference (ns)	DRCC-minimal (ns)	Δt	vs. reference
Sum	0.6259	0.7081	+82.2 ps	+13.13%
Cout	0.3953	0.4541	+58.8 ps	+14.87%

Table 7: Real transient propagation delay of the transistor-count-minimal 36-MOSFET cell against the 50-MOSFET reference, single-edge A-toggle vector, identical loading on both outputs of both circuits. Reproduced verbatim in Table 27's 1-bit row.

Finding. The transistor-count-minimal 36-MOSFET cell is real-measurably *slower* than the 50-MOSFET reference at the single-cell level – +13.13% on Sum, +14.87% on Cout – the opposite sign from every chained result (4 through 32 bits) built from the same cell (Table 27). This is consistent with, and gives a transistor-level confirmation of, the structural signal already reported in Table 6: Cout's reconstructed path reuses p_4, a node with fan-out 3 in this design versus fan-out ≤ 2 on the corresponding reference nodes. A plausible account, not demonstrated here by a per-stage decomposition, is that this extra capacitive loading is paid once per cell regardless of chain length, while the reference's own NAND2-vs-NAND3 structural cost is suggested by the manuscript's own per-stage argument to compound with chain length – consistent with the same cell losing in isolation but winning once chained (Section 3.12's Finding restates this crossover, with the same caveat). A second, independently-designed 48-MOSFET cell that is faster than the reference at the single-cell level too, by deliberately trading part of the MOSFET-count saving for parallel drive strength rather than minimizing transistor count, is reported next.

3.6 Measured transistor-level optimization of the 1-bit full adder (1BFA)

The structural results above identify two competing objectives for the same full-adder interface: the 9-NAND2 realization is exact-minimal in NAND2 count, but its reconstructed `Cout` path is deeper than the reference path. Once the missing `C5_models.txt` file became available, this trade-off could be tested directly at transistor level rather than inferred from gate depth alone. The experiments in this subsection therefore start from the already verified 50-MOSFET reference cell and the already verified 9-NAND2/36-MOSFET realization and ask a new engineering question: how much of the transistor-count reduction can be retained while the two observable outputs, `Sum` and `Cout`, are balanced for transient delay under identical loading?

All measurements below use LTspice 26.0.2 for macOS, the supplied AMI Semiconductor C5 BSIM3 models [16, 14, 15, 17], $V_{DD} = 5\,\text{V}$, the same device geometries unless explicitly stated otherwise, identical 50 fF loads on the compared observable outputs, and a 2.5 V threshold for delay extraction. For the one-bit comparison the applied transition is $A = 1$, $B = 0$, and $C_{\text{in}} : 0 \to 1$ at 5 ns. Thus `Cout` rises and `Sum` falls. The sign convention is fixed throughout:

$$\Delta t = t_{\text{DRCC}} - t_{\text{REF}}, \qquad \Delta t < 0 \iff \text{DRCC is faster.}$$

– Baseline: exact-minimal 9-NAND2 cell

The first transistor-level transient run uses the 9-NAND2 realization already proven minimal above. It contains

$$9 \times 4 = 36 \text{ MOSFETs},$$

which is 28% fewer transistors than the 50-MOSFET reference. Under the stated $C_{\text{in}} \rightarrow$ output transition, however, transistor-count minimality does not coincide with timing minimality:

$$t_{\text{Cout,REF}} = 336.645 \text{ ps}, \qquad t_{\text{Cout,9N}} = 399.400 \text{ ps}, \tag{5}$$

$$t_{\text{Sum,REF}} = 380.685 \text{ ps}, \qquad t_{\text{Sum,9N}} = 454.704 \text{ ps}. \tag{6}$$

Hence the 9-NAND2 cell is slower by approximately 18.64% on `Cout` and 19.44% on `Sum` for this particular transition. This is consistent with Table 6: the reduction removes transistors but creates a deeper reconstructed carry path.

Boundary. This is the same 36-MOSFET `_DRCC__1BitFA` cell as Table 7 above, but measured on a different single-edge path: $C_{\text{in}} : 0 \rightarrow 1$ with A, B held here, versus $A : 0 \rightarrow 1$ with B, C_{in} held there. The two results (+18.64%/+19.44% here vs. +14.87%/+13.13% there) are therefore not repeated measurements of the same experiment and are not expected to agree numerically – both are real, both show the same cell slower than the reference in isolation, and the difference between them is itself informative (different primary input driving the shared, higher-fan-out internal nodes changes the measured delay, consistent with the pin-order asymmetry already noted for the reference NAND2 subcircuit).

– Rejected branch: serial output regeneration

A first attempt to trade some of the 28% transistor saving for improved edge regeneration added pairs of NAND2 gates serially after the reduced `Cout` core. This is a *topology* experiment, not a NAND-count experiment: the independent variable being tested is where the additional gates are placed (serially, in series in the `Cout` path), not how many NAND2 gates the cell contains. The 11-NAND serial variant contains 44 MOSFETs and the 13-NAND serial variant 52 MOSFETs. The result is unambiguous: added serial logic depth makes the carry path progressively worse while leaving the unmodified Sum path essentially unchanged.

variant	MOSFETs	t_{Cout} (ps)	Cout vs. ref.	t_{Sum} (ps)	Sum vs. ref.
9 NAND baseline	36	399.400	+18.64%	454.704	+19.44%
11 NAND, serial	44	581.942	+72.87%	455.555	+19.67%
13 NAND, serial	52	787.207	+133.84%	455.550	+19.67%

Table 8: Serial-regeneration branch. The additional NAND2 stages increase logic depth on `Cout`; they do not provide the desired acceleration.

The carry increments are themselves diagnostic:

$$581.942 - 399.400 = 182.542 \text{ ps},$$

$$787.207 - 581.942 = 205.265 \text{ ps}.$$

The second increment is larger than the first, so this serial family is not merely a constant-delay extension; the measured chain becomes progressively slower. This branch is retained as a negative result because it distinguishes *more devices* from *more useful drive*. The narrow, correctly-scoped conclusion is: *additional serial NAND stages in this regeneration family increase delay* – not that a larger NAND2 count is generally slower. The parallel-drive branch below adds NAND2 gates too (10, then 11, then 12 total), but in parallel rather than in series, and speeds the cell up; NAND-gate count alone is not the variable that predicts either outcome.

– Parallel drive: preserving logic depth

The next branch keeps the Boolean depth unchanged and instead increases physical drive by duplicating gates in parallel, with identical inputs and a common output node. The naming below tracks total NAND2 count and which output(s) gained a parallel driver:

$$10\text{P} = \text{9-NAND core+parallel } C_{\text{out}}, \qquad 11\text{P} = 10\text{P+parallel } p_4, \qquad 12\text{P} = 11\text{P+parallel Sum.}$$

The first such candidate duplicates only the final carry NAND2. It therefore uses 10 NAND2 gates, 40 MOSFETs, and still has a 20% transistor-count reduction relative to the 50-MOSFET reference. Its carry delay falls from 399.400 to 353.942 ps, an improvement of 45.46 ps over the 9-NAND baseline, but Sum becomes slower because the shared internal node $p4$ now sees greater fan-out.

That observation localizes the new bottleneck. A second parallel NAND2 is therefore added at the $p4$ driver itself, producing an 11-NAND *parallel-balanced* cell. This version uses 44 MOSFETs (12% fewer than the reference) and changes the one-bit carry result qualitatively:

$$t_{\text{Cout,11P}} = 267.186 \text{ ps}, \qquad \Delta t_{\text{Cout}} = -20.63\%.$$

The Sum path also improves relative to the 10-NAND candidate, but remains 7.64% slower than the reference.

To determine whether the shared $p4$ fan-out is the remaining Sum bottleneck, the two identical $p4$ functions are then separated into $p4_{\text{co}}$ and $p4_{\text{sum}}$. No transistor is added: one $p4$ driver feeds only the two parallel carry gates, the other feeds only the two Sum-preparation gates. The split version measures 267.885 ps on Cout (-20.43%) and 408.462 ps on Sum ($+7.32\%$). The Sum improvement is only 1.23 ps relative to the shared-$p4$ version, establishing that most of the remaining Sum delay is downstream of $p4_{\text{sum}}$.

– Parallel Sum output and final transistor sizing

The final topological step duplicates the Sum-output NAND2 in parallel, analogous to the already successful parallel carry output. The resulting 12-NAND cell contains 48 MOSFETs, still two fewer than the reference. Before any geometry change it measures

$$t_{\text{Cout,12P}} = 267.694 \text{ ps} \quad (-20.48\%),$$

$$t_{\text{Sum,12P}} = 381.036 \text{ ps} \quad (+0.093\%).$$

Thus the Sum path is already within 0.356 ps of the reference. A repeat run with the transient step parameter reduced from 2 ps to 0.2 ps reproduces the same extracted delays to the displayed precision, so the remaining difference is not removed by that numerical refinement.

The final refinement changes no topology and adds no transistor. Only the NMOS widths in the two parallel Sum-output NAND2 gates are increased by 5%:

$$W_N : 3.60\,\mu\text{m} \longrightarrow 3.78\,\mu\text{m}, \qquad L = 0.60\,\mu\text{m},$$

while their PMOS widths and all other transistor geometries are left unchanged. Since the measured Sum transition is falling, this targets the series NMOS pull-down path directly.

The final one-bit result is

$$t_{\text{Cout,REF}} = 336.645 \text{ ps}, \qquad t_{\text{Cout,final}} = 267.685 \text{ ps}, \tag{7}$$

$$t_{\text{Sum,REF}} = 380.680 \text{ ps}, \qquad t_{\text{Sum,final}} = 377.046 \text{ ps}. \tag{8}$$

Therefore,

$$\boxed{\Delta t_{\text{Cout}} = -68.960 \text{ ps} = -20.485\%}$$

and

$$\boxed{\Delta t_{\text{Sum}} = -3.634 \text{ ps} = -0.955\%}.$$

For this stimulus, model set, load, and threshold, both observable outputs of the 48-MOSFET realization are faster than the 50-MOSFET reference.

Figure 8 shows this same result as a transient chart: both circuits' `Cout` and `Sum` traces for the single $C_{\text{in}} : 0 \to 1$ edge, with the 50%-crossing points marked at exactly the four timestamps above.

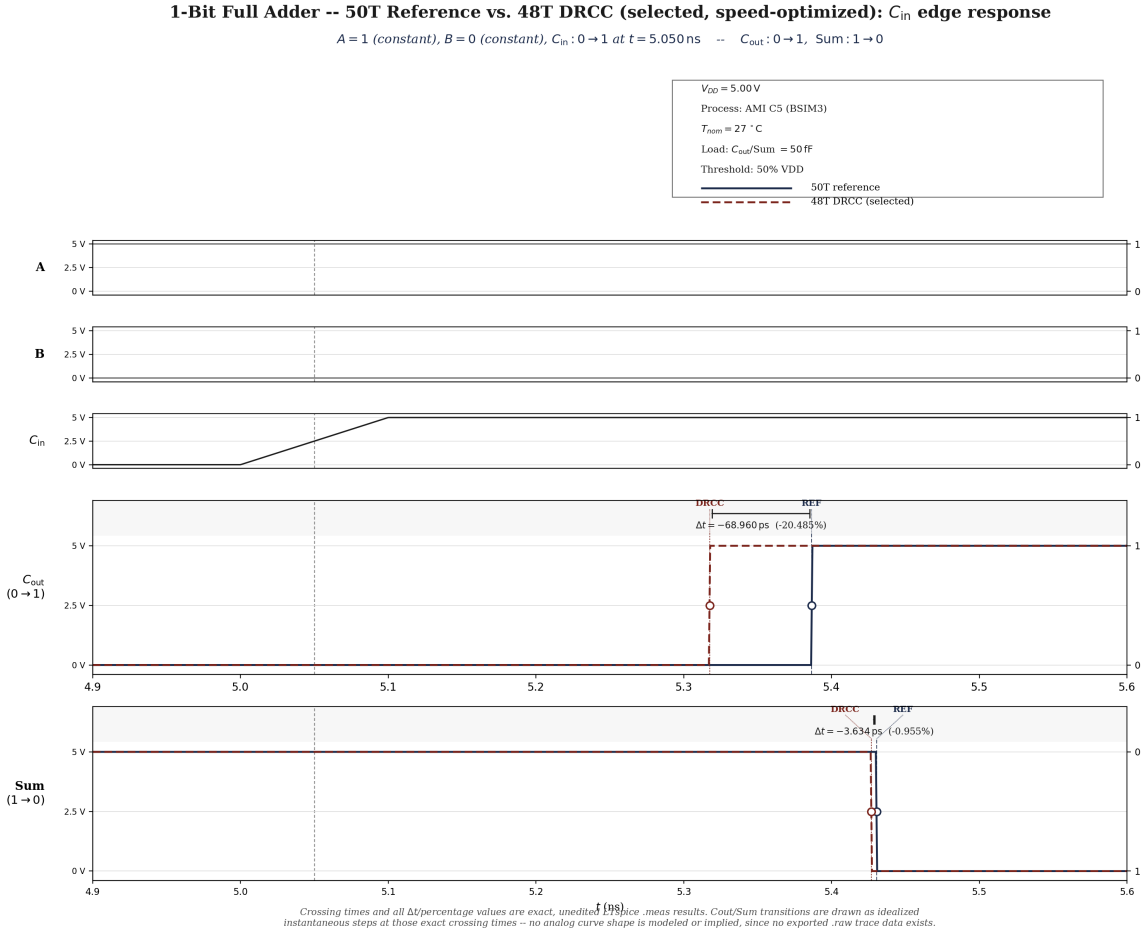


Figure 8: Transient response of the selected 48-MOSFET DRCC 1BFA (`_DRCC__1BitFA_12P_Final`) against the 50-MOSFET reference, for $A = 1/B = 0$ held and $C_{\text{in}} : 0 \to 1$. Both circuits share the same 50 fF load, AMI C5 BSIM3 models [16], $V_{DD} = 5$ V, and 27 °C nominal process corner. The four marked 50%-crossing times and both Δt/percentage figures are exact, unedited LTspice `.meas` results, identical to the boxed values above and to Table 9's last row; `Cout`/`Sum` transitions are drawn as idealized instantaneous steps at those exact crossing times, not as a modeled analog curve shape – this record does not include the raw `.raw` analog time series, and none is implied by the drawn step.

Finding. For the stated C5 transistor models and a fixed $C_L = 50$ fF output load on both outputs of both circuits, the exact-minimal 36-MOSFET NAND2 realization is not the best

variant	MOSFETs	count change	Cout (ps)	Cout vs. ref.	Sum vs. ref.
reference	50	–	336.645	–	–
9 NAND baseline	36	−28%	399.400	+18.64%	+19.44%
10P, parallel Cout	40	−20%	353.942	+5.14%	+31.90%
11P, shared $p4$	44	−12%	267.186	−20.63%	+7.64%
11P, split $p4$	44	−12%	267.885	−20.43%	+7.32%
12P, parallel Sum	48	−4%	267.694	−20.48%	+0.093%
12P + 5% Sum-NMOS	48	−4%	267.685	−20.485%	−0.955%

Table 9: Measured one-bit optimization path. Positive percentages are slower than the reference; negative percentages are faster. The last geometry-only refinement (12P + 5% Sum-NMOS) preserves the 48-transistor *count* – "4% fewer MOSFETs" is exact – but the 5% NMOS width increase on two gates grows their active device area, so this is not a claim of 4% less silicon area or energy; see the boundary below.

timing point. Of the candidates investigated here, a 48-MOSFET parallel-balanced realization, with a local 5% NMOS width increase only at the two Sum-output NAND2 gates, is the best-performing: it retains a 4% transistor-count reduction relative to the 50-MOSFET reference while measuring 20.485% lower Cout delay and 0.955% lower Sum delay for the investigated transition, both figures specific to that 50 fF load. This is a statement about the candidates tried, not a claim that no better 48-MOSFET (or other) topology exists.

Boundary. The 4% figure is a *transistor-count* reduction, not a measured layout-area, silicon-volume, energy, or fabrication-cost reduction. The 5% width increase in four NMOS devices changes physical area and capacitance even though it does not change transistor count. Likewise, the timing percentages above are not universal cell constants: they apply to the stated C5 model, supply, loading, threshold, device dimensions, and input transition. PVT corners, both output edge directions, all input transitions, dynamic power, and post-layout parasitics remain separate validation tasks.

– Selected 48-MOSFET SPICE cell (final investigated candidate)

The functional core used for the selected one-bit and 32-bit runs is shown below – "selected" meaning the best-performing candidate among those investigated in this subsection, not a claim that the search space was exhausted. The two $p4$ functions are deliberately separated by load, both observable outputs are driven by two identical NAND2 instances in parallel, and only the Sum-output pair uses the widened-NMOS cell.

Figure 9 shows this topology schematically, gate by gate, using the same reference designators (`Xp1`…`Xsum2`) as the SPICE subcircuit in Listing 1 immediately below it, so the drawing and the netlist can be read side by side.

Listing 1: Selected 12-NAND/48-MOSF DRCC 1BFA subcircuit (final investigated candidate).

```
.SUBCKT _16BitAdder__Nand2Gate_SumPD Out Inp0 Inp1
Mnmos@0 net@1 Inp1 gnd gnd NMOS L=0.6U W=3.78U
Mnmos@1 Out Inp0 net@1 gnd NMOS L=0.6U W=3.78U
Mpmos@0 vdd Inp1 Out vdd PMOS L=0.6U W=3.6U
Mpmos@1 vdd Inp0 Out vdd PMOS L=0.6U W=3.6U
.ENDS _16BitAdder__Nand2Gate_SumPD

.SUBCKT _DRCC__1BitFA_12P_Final Cin Cout Sum Inp0 Inp1
```

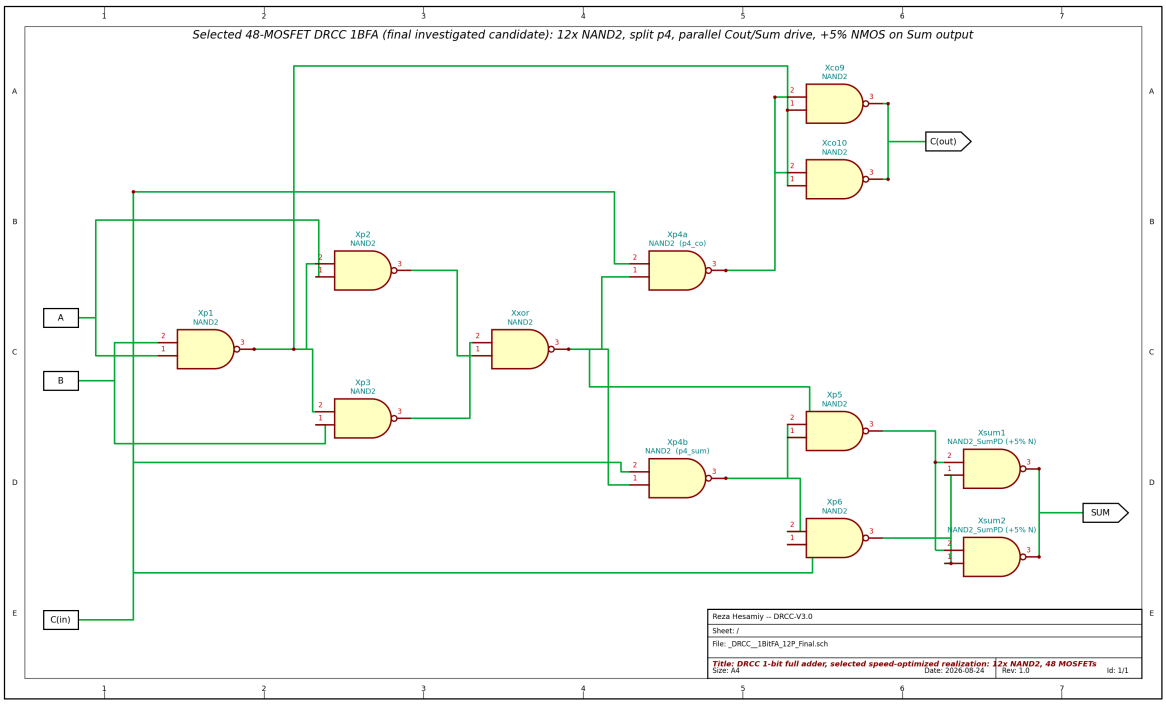


Figure 9: The selected 48-MOSFET DRCC 1BFA (final investigated candidate), transcribed gate-for-gate from `_DRCC__1BitFA_12P_Final` in the real SPICE deck. `Xp1`–`Xxor` are the unchanged 9-NAND-core prefix; `Xp4a`/`Xp4b` split the former shared $p4$ node into $p4\_co$/$p4\_sum$; `Xco9`‖`Xco10` drive Cout in parallel from $p4\_co$; `Xp5`/`Xp6` and `Xsum1`‖`Xsum2` drive Sum in parallel from $p4\_sum$. `Xsum1`/`Xsum2` are the only two instances built from the widened-NMOS gate (`Nand2Gate_SumPD`, NMOS $W = 3.78\,\mu$m vs. $3.60\,\mu$m elsewhere), marked directly on those two gates. 12 NAND2 instances, 48 MOSFETs total. This 48T speed-optimized cell is a separate topology from the 36T transistor-minimal cell used throughout the width-scaling family (Sections 3.8–3.12); the two are not interchangeable and are not the same circuit.

```
Xp1   p1      Inp0   Inp1   _16BitAdder__Nand2Gate
Xp2   p2      Inp0   p1     _16BitAdder__Nand2Gate
Xp3   p3      Inp1   p1     _16BitAdder__Nand2Gate
Xxor  xor_ab  p2     p3     _16BitAdder__Nand2Gate

Xp4a  p4_co   xor_ab Cin    _16BitAdder__Nand2Gate
Xp4b  p4_sum  xor_ab Cin    _16BitAdder__Nand2Gate

Xp5   p5      xor_ab p4_sum _16BitAdder__Nand2Gate
Xp6   p6      Cin    p4_sum _16BitAdder__Nand2Gate

Xsum1 Sum     p5     p6     _16BitAdder__Nand2Gate_SumPD
Xsum2 Sum     p5     p6     _16BitAdder__Nand2Gate_SumPD

Xco9  Cout    p1     p4_co  _16BitAdder__Nand2Gate
Xco10 Cout    p1     p4_co  _16BitAdder__Nand2Gate
.ENDS _DRCC__1BitFA_12P_Final
```

The one-bit measurement block used for the selected-cell timing comparison is:

Listing 2: One-bit transient measurement block.

```
.meas tran delay_cout_ref  TRIG V(Cin) VAL=2.5 RISE=1 \
  TARG V(COUT_REF)  VAL=2.5 RISE=1
.meas tran delay_cout_drcc TRIG V(Cin) VAL=2.5 RISE=1 \
```

```
  TARG V(COUT_DRCC) VAL=2.5 RISE=1
.meas tran delta_cout PARAM delay_cout_drcc-delay_cout_ref
.meas tran delta_cout_pc PARAM \
  100*(delay_cout_drcc-delay_cout_ref)/delay_cout_ref

.meas tran delay_sum_ref  TRIG V(Cin) VAL=2.5 RISE=1 \
  TARG V(SUM_REF)  VAL=2.5 FALL=1
.meas tran delay_sum_drcc TRIG V(Cin) VAL=2.5 RISE=1 \
  TARG V(SUM_DRCC) VAL=2.5 FALL=1
.meas tran delta_sum PARAM delay_sum_drcc-delay_sum_ref
.meas tran delta_sum_pc PARAM \
  100*(delay_sum_drcc-delay_sum_ref)/delay_sum_ref

.tran 0.2ps 20ns
.include C5_models.txt
```

3.6.1 Scaling the optimized full-adder cell to a 32-bit ripple-carry chain

The selected 48-MOSFET cell (48T speed-optimized chain, distinct from the 36-MOSFET-per-bit chains of Sections 3.8–3.11 and 3.12) is next instantiated 32 times in a ripple-carry chain and compared against 32 instances of the 50-MOSFET reference. The adder-cell transistor counts are therefore

$$N_{T,\mathrm{REF}} = 32 \times 50 = 1600, \qquad N_{T,\mathrm{DRCC}} = 32 \times 48 = 1536,$$

again a 4% transistor-count reduction before interconnect and external loads are counted.

The stimulus is selected to force a full-width carry propagation:

$$A = \texttt{0xFFFFFFFF}, \qquad B = \texttt{0x00000000} \text{ initially}, \qquad C_{\mathrm{in}} = 0,$$

with only B_0 rising from 0 to 1. After the edge,

$$\texttt{0xFFFFFFFF} + 1 = \texttt{0x100000000},$$

so $C_1, \ldots, C_{31}$ and final `Cout` rise while S_{31} falls. The 32-bit run converges after LTspice switches from the failed direct Newton operating-point iteration to Gmin stepping; Gmin stepping succeeds and the transient simulation completes normally. This is a standard LTspice operating-point fallback, not a simulation error: the 32-bit run converged after LTspice's Gmin stepping, and every `.meas` value reported below was produced from that converged transient solution.

The end-to-end result is:

path	reference (ns)	DRCC (ns)	Δt (ns)	change
$B_0 \to C_{\mathrm{out}}$	10.7372	7.7809	-2.9563	-27.53%
$B_0 \to S_{31}$	10.7861	7.8898	-2.8962	-26.85%

Table 10: Measured 32-bit worst-case ripple propagation – **48T speed-optimized chain** (selected 48-MOSFET-per-bit cell of this subsection) vs. the 50-MOSFET-per-bit reference. Not the same circuit as the 36T-minimal chain of Tables 24/27.

To distinguish a boundary effect from sustained carry propagation, the same run measures intermediate carry nodes at $C_1, C_2, C_4, C_8, C_{16}$, and C_{31}:

The relative advantage grows rapidly over the first stages and then approaches a saturation regime near 28% over the investigated chain:

$$-13.57\%, \ -21.09\%, \ -24.57\%, \ -26.37\%, \ -27.28\%, \ -27.72\%.$$

stage	t_{REF} (ps)	t_{DRCC} (ps)	Δt (ps)	change
C_1	345.99	299.05	-46.94	−13.57%
C_2	681.17	537.53	-143.64	−21.09%
C_4	1350.61	1018.72	-331.89	−24.57%
C_8	2690.22	1980.74	-709.48	−26.37%
C_{16}	5369.27	3904.31	-1464.96	−27.28%
C_{31}	10391.97	7511.16	-2880.81	−27.72%

Table 11: Measured internal carry profile of the 32-bit chain – **48T speed-optimized chain** (same run as Table 10), not the 36T-minimal chain.

The accumulated absolute advantage grows from 46.94 ps at C_1 to 2.881 ns at C_{31}. The data therefore do not show a destructive loss of the timing advantage as the carry propagates. They show a rapidly increasing relative advantage that becomes progressively less variable with depth over this 32-bit experiment.

A useful coarse measure of the settled region follows from the interval $C_8 \to C_{16}$:

$$\frac{t_{C16,\mathrm{REF}} - t_{C8,\mathrm{REF}}}{8} \approx 334.88 \text{ ps/bit},$$

$$\frac{t_{C16,\mathrm{DRCC}} - t_{C8,\mathrm{DRCC}}}{8} \approx 240.45 \text{ ps/bit}.$$

This corresponds to approximately 28.2% lower incremental carry delay per stage over that interval. It is an interval measurement from this specific chain, not a universal per-bit constant.

Finding. The one-bit optimization persists under 32-fold ripple composition. For the stated full-propagation vector, the 32-bit final `Cout` delay is 27.53% lower and the S_{31} delay is 26.85% lower than the corresponding 50-MOSFET-per-bit reference chain, while the adder-cell transistor count remains 4% lower. The internal carry profile approaches a measured saturation region near 28% rather than collapsing with chain depth.

Boundary. The 32-bit result is a transistor-level simulation result for one stated worst-case-style full-propagation vector, one device-model set, one supply, and the stated output loads. It is not a claim that every 32-bit input transition, every process/voltage/temperature corner, every interconnect layout, or every adder architecture improves by 27–28%. The measured carry profile supports a strong scaling observation for this particular ripple chain; broader robustness must be established separately.

– 32-bit measurement code

The end-to-end and carry-profile measurements used in the 32-bit deck are included here so that the reported timing quantities are reproducible from the netlist itself.

Listing 3: 32-bit end-to-end and carry-profile measurement statements.

```
.meas tran delay_cout_ref  TRIG V(B0) VAL=2.5 RISE=1 \
  TARG V(COUT_REF)  VAL=2.5 RISE=1
.meas tran delay_cout_drcc TRIG V(B0) VAL=2.5 RISE=1 \
  TARG V(COUT_DRCC) VAL=2.5 RISE=1
```

```
.meas tran delta_cout PARAM delay_cout_drcc-delay_cout_ref
.meas tran delta_cout_pc PARAM \
  100*(delay_cout_drcc-delay_cout_ref)/delay_cout_ref

.meas tran delay_sum_ref  TRIG V(B0) VAL=2.5 RISE=1 \
  TARG V(S31_REF)  VAL=2.5 FALL=1
.meas tran delay_sum_drcc TRIG V(B0) VAL=2.5 RISE=1 \
  TARG V(S31_DRCC) VAL=2.5 FALL=1
.meas tran delta_sum PARAM delay_sum_drcc-delay_sum_ref
.meas tran delta_sum_pc PARAM \
  100*(delay_sum_drcc-delay_sum_ref)/delay_sum_ref

.meas tran t_c1_ref   TRIG V(B0) VAL=2.5 RISE=1 TARG V(C1_REF)  VAL=2.5 RISE=1
.meas tran t_c1_drcc  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C1_DRCC) VAL=2.5 RISE=1
.meas tran t_c2_ref   TRIG V(B0) VAL=2.5 RISE=1 TARG V(C2_REF)  VAL=2.5 RISE=1
.meas tran t_c2_drcc  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C2_DRCC) VAL=2.5 RISE=1
.meas tran t_c4_ref   TRIG V(B0) VAL=2.5 RISE=1 TARG V(C4_REF)  VAL=2.5 RISE=1
.meas tran t_c4_drcc  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C4_DRCC) VAL=2.5 RISE=1
.meas tran t_c8_ref   TRIG V(B0) VAL=2.5 RISE=1 TARG V(C8_REF)  VAL=2.5 RISE=1
.meas tran t_c8_drcc  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C8_DRCC) VAL=2.5 RISE=1
.meas tran t_c16_ref  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C16_REF)  VAL=2.5 RISE=1
.meas tran t_c16_drcc TRIG V(B0) VAL=2.5 RISE=1 TARG V(C16_DRCC) VAL=2.5 RISE=1
.meas tran t_c31_ref  TRIG V(B0) VAL=2.5 RISE=1 TARG V(C31_REF)  VAL=2.5 RISE=1
.meas tran t_c31_drcc TRIG V(B0) VAL=2.5 RISE=1 TARG V(C31_DRCC) VAL=2.5 RISE=1

.meas tran delta_c1  PARAM t_c1_drcc-t_c1_ref
.meas tran delta_c2  PARAM t_c2_drcc-t_c2_ref
.meas tran delta_c4  PARAM t_c4_drcc-t_c4_ref
.meas tran delta_c8  PARAM t_c8_drcc-t_c8_ref
.meas tran delta_c16 PARAM t_c16_drcc-t_c16_ref
.meas tran delta_c31 PARAM t_c31_drcc-t_c31_ref

.tran 0.2ps 30ns
.include C5_models.txt
```

3.7 Illustrative full-adder physical layouts, and a second, distinct verified cell

Reza supplied two real Electric-VLSI layout renderings (library `16BitAdder`, MOSIS `mocmos`) as reference: the actual, DRC-clean, fabrication-oriented physical layouts of `1BitFA{lay}` and a second, distinct cell, `1BitAdder{lay}`. This record has no layout/DRC tool (no Magic, no KLayout, no Electric VLSI) and no verified copy of the exact MOSIS `mocmos` design-rule deck, so it cannot reproduce anything DRC-checked or fabrication-oriented – drawing rectangles that merely *look like* a real layout, without checking them, would be exactly the kind of plausible-but-unverified claim this record has refused throughout. This record stays with what it can check: the gate-level netlist and its exhaustive verification, not a physical rendering of it.

A second cell, and one further reduction `1BitAdder{lay}` is architecturally distinct from `1BitFA` – its top level produces only $G_0 = \text{AND}(A, B)$ ("generate"), $P_0 = \text{OR}(A, B)$ ("propagate"), and Sum; there is no `Cout` anywhere in the cell, consistent with being the per-bit front end of a carry-lookahead architecture whose carry chain lives in a separate block outside this record's scope. Switch-level-verified the same way as `1BitFA{lay}` (`exp1g_layout_verification.py`'s technique, extended to the `Inverter{lay}`/`Nor2Gate{lay}` primitives this cell also uses) at 44 MOSFETs, matching the real layout exactly.

Building G_0 and P_0 independently, NAND2-only (the same technology convention as the rest of this section: no free complements, primary inputs only), costs 2 gates for G_0 and 3 for P_0 on top of Sum's own already-minimal 8-gate chain – 13 gates, 52 MOSFETs, with no sharing. But $G_0 = \text{NOT}(p_1)$ where $p_1 = \text{NAND}(A, B)$ is already computed inside Sum's chain, and

$P_0 = G_0 \vee$ `xor_ab` (true on both-1 or exactly-one-1, exactly the two ways $A \vee B$ can hold), which by De Morgan is NAND(p_1, NOT(`xor_ab`)) – reusing p_1 a second time. Both identities are checked exhaustively, not asserted from the algebra alone (`exp1i_gp_reduction.py`):

construction	MOSFETs	gate instances	status
no cross-sharing (baseline)	52	13	verified (8/8)
DRCC sharing (G_0, P_0 reuse p_1, `xor_ab`)	44	11	verified (8/8)

Table 12: G_0, P_0, and Sum, NAND2-only technology, checked against the standard generate/propagate/sum functions on all 8 raw inputs (`exp1i_gp_reduction.json`), a -15.38% saving versus the no-sharing baseline.

Two things this reduction deliberately does not claim. First, it was found by hand, reusing the sharing pattern that worked for the full adder – it was not run through `exact_min_search.py`'s exhaustive technique, so 44 MOSFETs is checked and reduced, not proven minimal, unlike the full adder's 36-MOSFET result. Second, this NAND2-only 44 MOSFETs happens to exactly match `1BitAdder{lay}`'s real, inverter-based layout count – a genuine coincidence between two different technology routes (signal reuse here, dedicated `Inverter{lay}` cells there), not evidence that either is optimal.

3.8 Scaling the full adder to 4 bits: ripple-carry and carry-lookahead

Every result up to this point is about a single bit slice. This subsection scales the same, already-verified 1-bit cells (the reference and the 36-MOSFET/9-gate minimal design of Section 3.5; the 52-/44-MOSFET generate/propagate/sum cells of Table 12) into two standard 4-bit adder architectures – ripple-carry and carry-lookahead – and checks the whole 4-bit circuit again from scratch against plain integer addition on all $512 = 16 \times 16 \times 2$ ($A, B \in [0, 15]$, $C_{in} \in \{0, 1\}$) input combinations, rather than assuming correctness transfers automatically from the 1-bit result (`exp2a_4bit_ripple.py`, `exp2b_4bit_cla.py`).

4-bit ripple-carry Four instances of the same verified full-adder cell, chained so that bit i's carry-in is bit $i-1$'s carry-out (Figure 10). Because MOSFET count is additive across independent cell instances, the reference totals $4 \times 50 = 200$ MOSFETs and the DRCC-reduced chain totals $4 \times 36 = 144$ MOSFETs – a -28.00% saving, exactly reproducing the single-cell headline figure at 4-bit scale, now confirmed against all 512 combinations rather than the 8 single-bit vectors alone.

variant	MOSFETs	gate instances	vs. reference	status
4-bit reference	200	48	–	verified (512/512)
4-bit DRCC-reduced	144	36	-28.00%	verified (512/512)

Table 13: Same reduction as Table 5, additive across the 4 independent cell instances of the ripple-carry chain – checked here against all 512 (A, B, C_{in}) combinations rather than the 8 single-bit vectors alone (`exp2a_4bit_ripple.py`).

Finding. The 4-bit ripple-carry chain reproduces the 1-bit reduction ratio exactly at scale: 200 → 144 MOSFETs, 28% fewer transistors, now verified against all 512 (A, B, C_{in}) combinations rather than the 8 single-bit vectors alone.

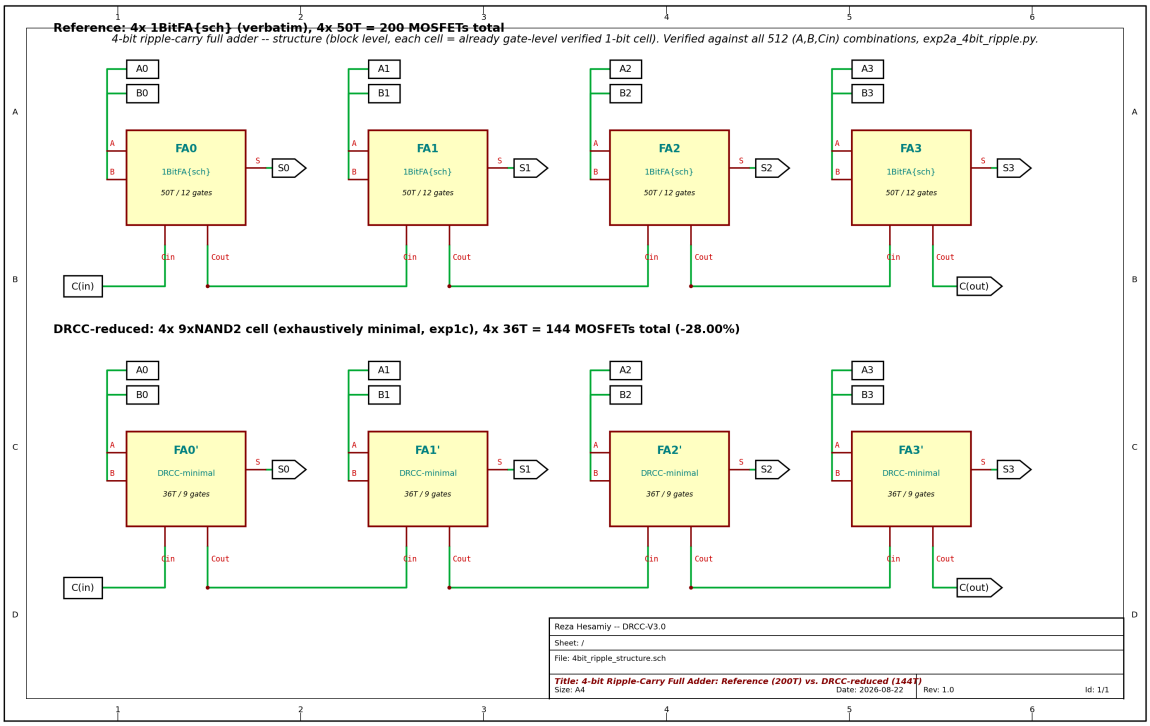


Figure 10: 4-bit ripple-carry full adder: reference (top, 4x `1BitFA{sch}`, 200 MOSFETs) vs. DRCC-reduced (bottom, 4x the 36-MOSFET minimal cell, 144 MOSFETs). Block-level structure; each cell is the already gate-verified 1-bit design. Verified against all 512 (A,B,Cin) combinations.

Figure 10 shows the chain at block level; expanding one row down to individual gate instances makes the same structure concrete at the level Figure 5 already established for a single cell – the reference design only, no DRCC reduction shown here, redrawn from Reza's real Electric-VLSI top-level cell (library `16BitAdder`).

What the ripple actually looks like Section 3.5's unit-delay logic simulation (Figure 7) showed the same reference-vs-DRCC timing comparison for a single bit. The identical model, chained the same way as Figure 10 (bit i's carry-in wired to bit $i-1$'s carry-out, one abstract tick per NAND2/NAND3 gate instance), makes the ripple itself visible at 4-bit scale: 8 hand-chosen `(A,B,C_in)` vectors – covering no carry, a single-stage carry, the full 4-stage worst-case ripple ($A = 15, B = 1, C_{in} = 0$), a partial ripple, maximum carry load ($A = B = 15, C_{in} = 1$), and a carry generated directly at the top bit with no ripple at all – not the full 512-combination sweep already used to prove correctness in Figure 10. Every settled `Sum`/`Cout` at the end of every window, in both circuits, was checked against plain integer addition before this figure was drawn – 8/8 correct in both cases.

From ticks to nanoseconds: a real SPICE transient measurement Section 3.5 stated a specific, named gap twice: the layout deck's own `.include C5_models.txt` line references a device-model file that was not supplied in this record, so no real analog transient simulation could be run there, at any scale – only the structural gate-depth proxy of Table 6 and the unit-delay digital simulation of Figure 7, both explicitly labelled "not SPICE." Reza subsequently obtained the actual `C5_models.txt` (BSIM3 models for AMI Semiconductor's C5 process [16], $V_{DD} = 5\,\mathrm{V}$, $T_{nom} = 27\,^{\circ}\mathrm{C}$) and ran a real transient simulation in LTspice, outside this record's own sandboxed environment, which has no SPICE simulator installed. This closes that gap for

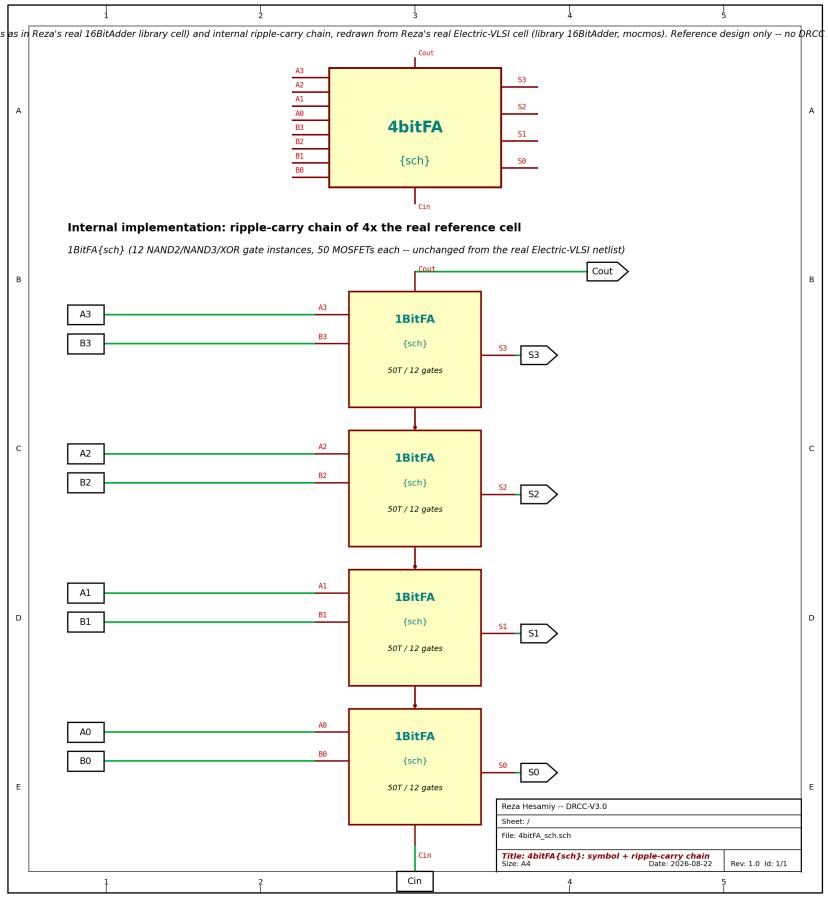


Figure 11: `4bitFA{sch}`: top-level symbol (pins as in the real `16BitAdder` library cell) and its internal ripple-carry chain of 4x `1BitFA{sch}`, expanded to individual gate instances. Reference design only – no DRCC reduction shown, no timing implied.

the 4-bit ripple-carry chain, on the specific stimulus described below – it does not retract either earlier "not SPICE" label, which still correctly describes Table 6 and Figure 7 themselves.

Both 4-bit chains reuse the identical real subcircuits verified elsewhere in this record: the reference chain is 4 instances of Reza's actual Electric-VLSI `1BitFA{sch}` cell (built from the real `Nand2Gate`/`Nand3Gate`/`XorGate` subcircuits, 200 MOSFETs total), and the DRCC-reduced chain is 4 instances of the proven-minimal 9x-NAND2 topology from Section 3.5 (144 MOSFETs total), built from the same real `Nand2Gate` subcircuit – so every transistor on both sides shares the identical model and geometry, not a re-estimated one. Both measured outputs, S_3 and C_{out}, carry an identical 50 fF load capacitance on both chains – the fair, matched-load condition that Section 3.5 noted the original 1-bit testbench lacked.

The stimulus is a single-edge worst-case ripple vector, chosen after a specific flawed candidate was identified and rejected. An earlier vector considered ($A: 0000 \to 1111$, $B: 0 \to 1$ simultaneously) was rejected because every stage's A input would then change at the same instant as the carry itself arrives – a simultaneous multi-input switching confound that a real transistor-level circuit can exhibit (stacking-order-dependent delay) but that is not a fair test of carry propagation in isolation. The vector actually used holds $A = 1111$ (15) constant on all four bits and switches only B_0, $0 \to 1$, at $t = 30$ ns (1 ns rise time), with B_1, B_2, B_3, C_{in} held at 0V throughout: verified by plain integer addition before the deck was written, $A{=}15, B{=}0, C_{in}{=}0 \Rightarrow S{=}15(1111), C_{out}{=}0$ before the edge, and $A{=}15, B{=}1, C_{in}{=}0 \Rightarrow S{=}0(0000), C_{out}{=}1$ after it. With $A_i = 1, B_i = 0$ for $i = 1, 2, 3$, each of those stages is in the pure carry-propagate condition ($C_i = C_{i-1}$ exactly), so the single edge on B_0 must ripple through all four stages – and, unlike an alternative

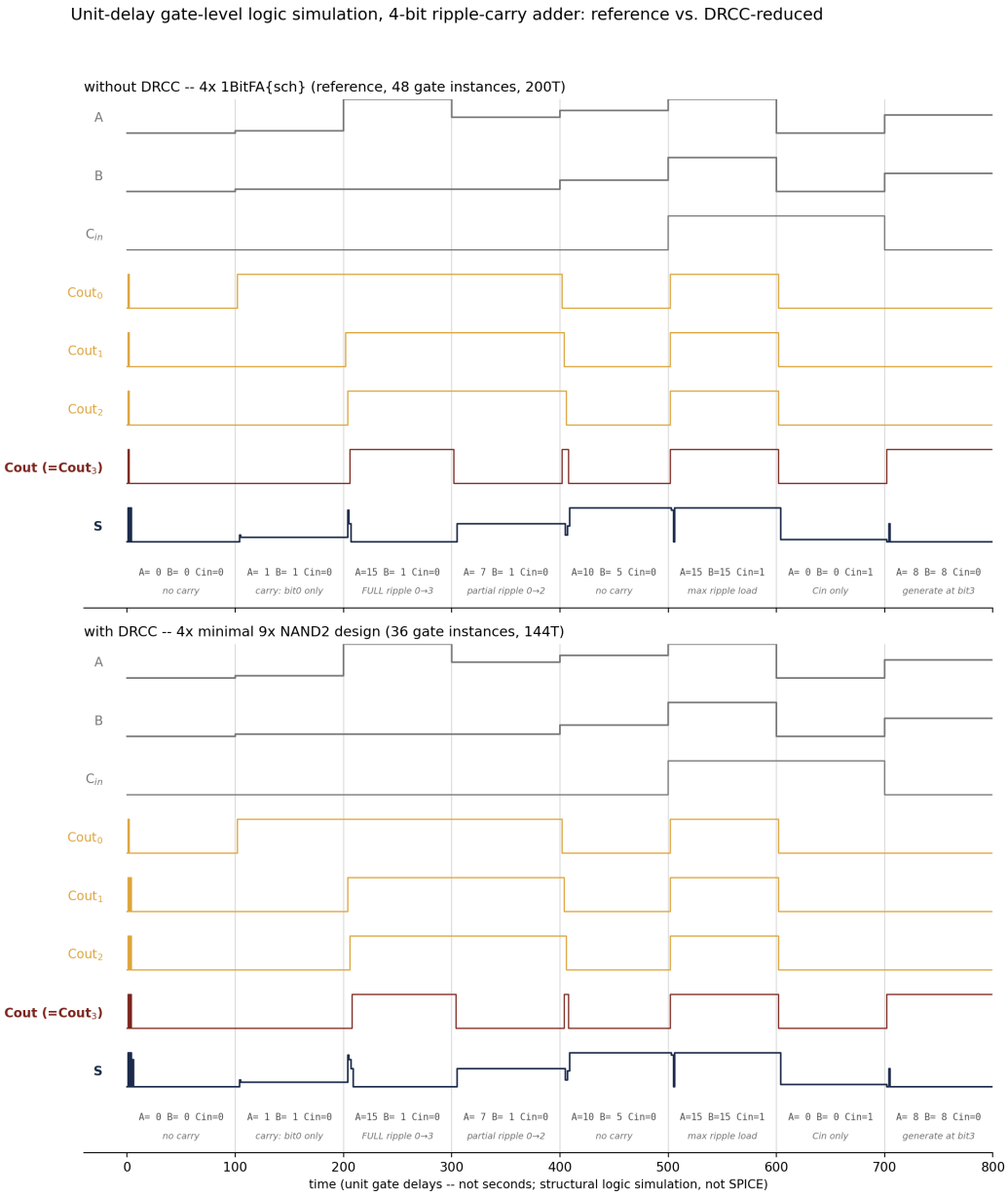


Figure 12: Unit-delay gate-level logic simulation of the 4-bit ripple-carry adder (not SPICE; x-axis is gate delays), reference (top, 200T) vs. DRCC-reduced (bottom, 144T), on 8 illustrative vectors. The three internal inter-stage carries (Cout_0..Cout_2) plus the final Cout settle visibly staggered – the ripple itself. All settled values verified against plain integer addition before plotting.

constant-$A = 0111$ vector where bit 3 sits in "kill" mode ($A_3 = B_3 = 0$) and C_{out} never moves, this vector also flips the real C_{out} pin, making it measurable on both signals. Automated `.meas TRAN` directives measure the propagation delay from B_0 crossing 50% of V_{DD} (2.5V, rising) to each output crossing the same 2.5V threshold (S_3 falling, C_{out} rising, per this vector's actual transition direction), separately for each chain.

Finding. The first real transient SPICE measurement in this record shows the DRCC-reduced 4-bit ripple-carry chain is not only 28% smaller (Table 13) but, on this worst-case single-edge

measured path	reference (ns)	DRCC-reduced (ns)	Δ (ps)	vs. reference
$B_0 \to S_3$ (50% V_{DD})	1.4141	1.3711	−42.99	−3.04%
$B_0 \to C_{out}$ (50% V_{DD})	1.3651	1.3186	−46.54	−3.41%

Table 14: Real transistor-level transient propagation delay, single-edge worst-case ripple vector, both chains under identical stimulus, loading, and BSIM3 model file (`4bit_singleedge_A15_with_meas.sp`, run in LTspice by Reza; `.meas TRAN` output reproduced verbatim). Consistency check: the `TRIG` time returned for all four measurements is 30.500 ns in every case, exactly matching the expected 2.5V crossing of the B_0 edge (30 ns delay, 1 ns risetime).

vector, also measurably *faster*: 3.04% lower $B_0 \to S_3$ delay and 3.41% lower $B_0 \to C_{out}$ delay than the reference, under identical stimulus, loading, and device models on both sides.

Boundary. This result is a single data point, not a sweep, and three specific limitations bound what it can support. First, it is one test vector – the single-edge worst-case ripple, deliberately chosen and justified above, not an exhaustive transition sweep the way correctness is checked elsewhere in this record; other transitions are not measured here. Second, it is one nominal process corner (27 °C, the BSIM3 file's own `TNOM`), with no fast/slow-corner or temperature sweep. Third, both chains are still `{sch}`-level only – the same real transistor models on both sides, but no layout-extracted diffusion or routing parasitics on either one, so this is a fair schematic-vs-schematic comparison, not a layout-extracted or silicon prediction, exactly the same distinction Section 3.5 already drew for MOSFET count. A plausible circuit-level reason for the direction of the effect: both chains need exactly 2 gate levels from C_{in} to C_{out} within each 1-bit cell, but the reference's second stage is a 3-input NAND3 (3 series pull-down transistors, $W = 5.4\,\mu$m each) while the DRCC cell's second stage is a 2-input NAND2 (2 series pull-down transistors, $W = 3.6\,\mu$m each) – fewer series transistors despite the narrower width. This is offered as a hypothesis consistent with the measured direction, not as a separately verified mechanism; it was not isolated or tested on its own.

4-bit carry-lookahead The generate/propagate cells of Table 12 feed a shared carry-lookahead network computing all four internal carries directly from $G_0..G_3$, $P_0..P_3$, C_{in} – not rippled bit-to-bit (Figure 13). The network is a small parallel-prefix/hybrid construction (not the generic textbook Kogge–Stone template used at 8 bits and beyond; see the boundary in Section 3.9), built NAND2-only like the rest of this section: an associative group operator combines adjacent generate/propagate pairs in $\lceil \log_2 4 \rceil = 2$ rounds, and each combine step uses the identity $a \vee (b \wedge c) = \mathrm{NAND}(\mathrm{NOT}(a),\ \mathrm{NAND}(b, c))$ (3 NAND2 gates instead of a generic AND-then-OR composition). Both the parallel-prefix structure and this identity were adopted only after an earlier, functionally-correct but structurally serial construction was found to give no real depth advantage over ripple-carry at all – an honest dead end, disclosed rather than discarded quietly, in `exp2b_4bit_cla.py`'s own comments. The lookahead network itself is identical in both variants below; only the per-bit cells differ, so the DRCC-specific comparison is isolated cleanly from the architecture's own, non-DRCC-related cost.

A counterintuitive but real result The carry-lookahead adder's saving is smaller than ripple-carry's (−8.99% vs. −28.00%), because the shared lookahead network – unaffected by

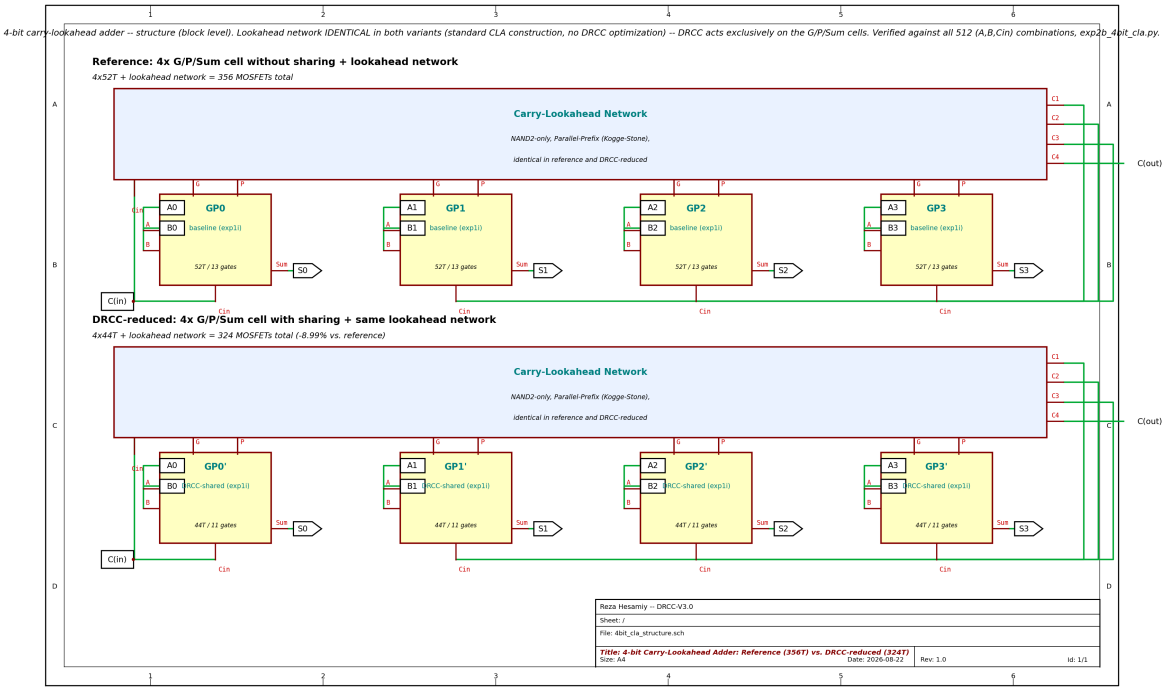


Figure 13: 4-bit carry-lookahead adder: reference (top, 4x the 52-MOSFET baseline generate/propagate/sum cell, 356 MOSFETs total) vs. DRCC-reduced (bottom, 4x the 44-MOSFET shared cell, 324 MOSFETs total). The lookahead network is unchanged between the two rows. Verified against all 512 (A,B,Cin) combinations.

architecture (4 bit)	reference	DRCC-reduced	saving	Cout depth (ref / DRCC)
ripple-carry	200 T	144 T	−28.00%	8 / 11
carry-lookahead	356 T	324 T	−8.99%	12 / 15

Table 15: 4-bit MOSFET totals and structural (gate-level, not SPICE) critical-path depth to Cout, both architectures, both verified against all 512 (A,B,Cin) combinations. Depth convention matches Table 6/Figure 7: a topology count, not picoseconds.

the per-bit DRCC reduction – makes up most of its MOSFET budget; DRCC only touches the four small generate/propagate/sum cells. More strikingly, this specific, strictly NAND2-only carry-lookahead construction has a *deeper* structural critical path to Cout (12 / 15 gate levels, reference / DRCC, computed from the verified netlist) than the ripple-carry chain (8 / 11) – the opposite of the classical "carry-lookahead is asymptotically faster than ripple" result. That classical result assumes native multi-input AND-OR-INVERT gates; built strictly from 2-input NAND primitives (this section's technology convention throughout), each parallel-prefix combine step costs several NAND2 gate-delays, which measurably erodes the $O(\log n)$ advantage at $n = 4$, while the ripple-carry reference cell benefits from a single, native 3-input NAND gate for Cout that this carry-lookahead construction does not use anywhere. No general claim about carry-lookahead architecture is made from this result – only that, computed directly from the verified netlist rather than assumed, it did not deliver a shallower critical path under this record's specific gate-primitive constraint.

3.9 Scaling the full adder to 8 bits: ripple-carry and carry-lookahead

Section 3.8 scaled the same verified 1-bit cells to 4 bits and checked the result from scratch rather than assuming correctness transfers. This subsection repeats exactly that discipline one level further: the identical 1-bit full-adder cells (ripple-carry) and generate/propagate/sum cells (carry-lookahead) are chained/networked to 8 bits and checked again from scratch against plain integer addition on all $131072 = 2 \times 256 \times 256$ ($A, B \in [0, 255]$, $C_{in} \in \{0, 1\}$) input combinations – not inferred from the 4-bit result (`exp3a_8bit_ripple.py`, `exp3b_8bit_cla.py`).

8-bit ripple-carry Eight instances of the same verified full-adder cell, chained the identical way as Figure 10 (Figure 14). MOSFET count is additive across independent cell instances just as at 4-bit scale, so the reference totals $8 \times 50 = 400$ MOSFETs and the DRCC-reduced chain totals $8 \times 36 = 288$ MOSFETs – the same -28.00% saving, now confirmed against all 131072 combinations rather than 512. The generalized chain-builder behind this result was itself cross-checked before being trusted at $n = 8$: run at $n = 4$ it reproduces Table 13's exact MOSFET counts and Cout depths, not merely a similar result.

variant	MOSFETs	gate instances	vs. reference	status
8-bit reference	400	96	–	verified (131072/131072)
8-bit DRCC-reduced	288	72	-28.00%	verified (131072/131072)

Table 16: Same reduction as Table 13, additive across the 8 independent cell instances of the 8-bit ripple-carry chain – checked against all 131072 (A, B, C_{in}) combinations rather than the 512 of Table 13 (`exp3a_8bit_ripple.py`).

Finding. The reduction ratio holds exactly at 8-bit scale too: 400 → 288 MOSFETs, 28% fewer transistors, verified against all 131072 (A, B, C_{in}) combinations – the same headline figure as 1-bit and 4-bit, now checked at three independent scales rather than assumed to hold by extrapolation.

What the ripple actually looks like at 8 bits The same unit-delay logic simulation used at 4-bit scale (Figure 12), chained the same way, makes the 8-stage ripple visible: 8 vectors, the direct 8-bit analogues of the 4-bit figure's own 8 vectors (no carry, a single-stage carry, the full 8-stage worst-case ripple, a partial ripple stopped by a killed top bit, an alternating no-carry pattern, maximum carry load, Cin alone, and a carry generated directly at the top bit) – not the exhaustive 131072-combination sweep already used to prove correctness above. Every settled `Sum`/`Cout`, both circuits, was checked against plain integer addition before this figure was drawn – 8/8 correct in both cases.

From ticks to nanoseconds, again: a real 8-bit SPICE transient measurement The same real BSIM3 device models and LTspice setup used for the 4-bit measurement above (Table 14) were extended to 8 bits, reusing the identical subcircuits chained twice as long: the reference chain is 8 instances of the real `1BitFA{sch}` cell (400 MOSFETs total), the DRCC-reduced chain is 8 instances of the proven-minimal 9x-NAND2 cell (288 MOSFETs total), both measured outputs carrying an identical 50 fF load on both chains.

The first attempt at the direct 8-bit analogue of the 4-bit vector – $A = 00001111$ (15, the same constant that worked at 4 bits) with B_0 toggling – turned out to be a dead end for measuring C_{out}, and this is disclosed rather than quietly swapped out: with $A_4 = B_4 = 0$, the carry chain that

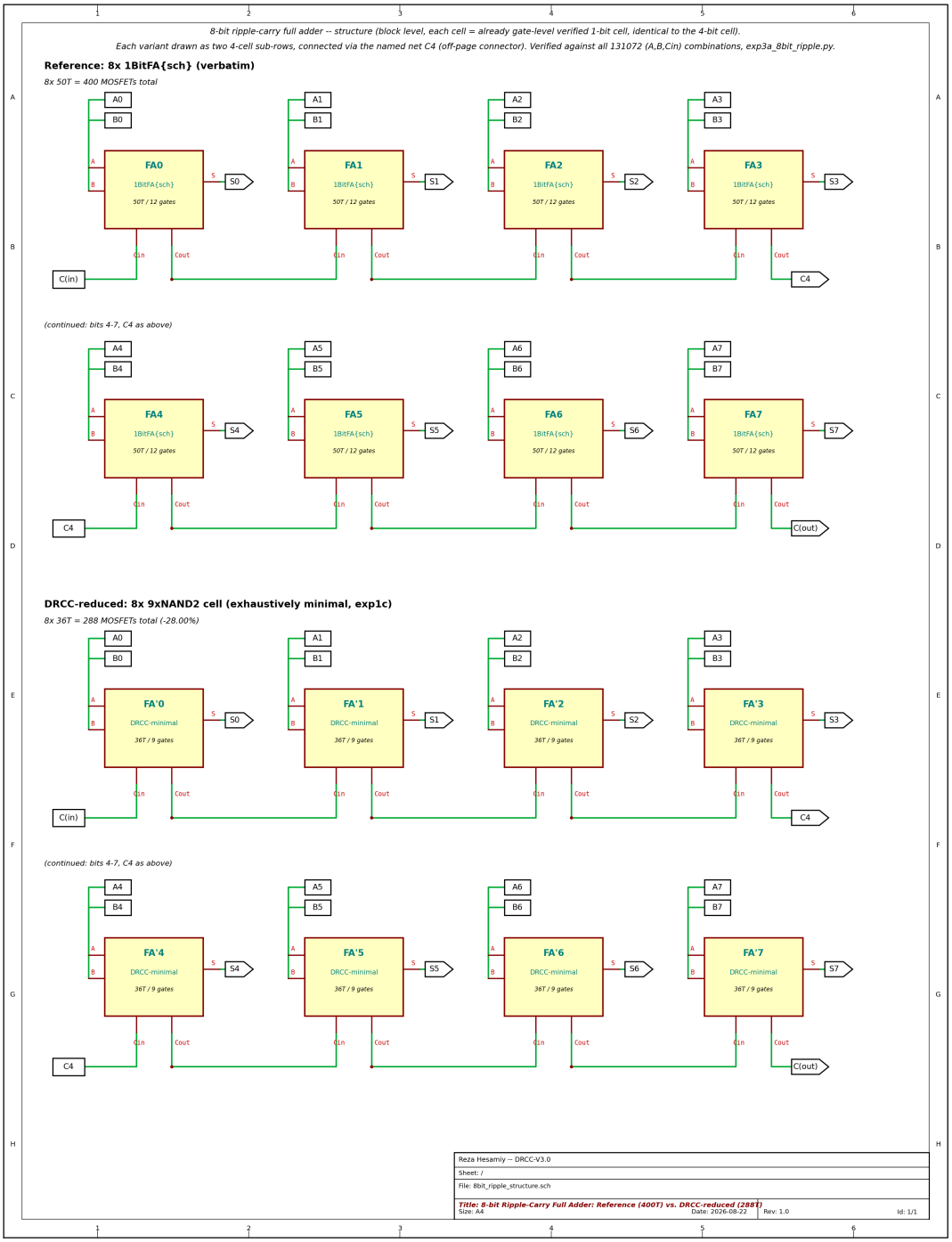


Figure 14: 8-bit ripple-carry full adder: reference (top, 8x `1BitFA{sch}`, 400 MOSFETs) vs. DRCC-reduced (bottom, 8x the 36-MOSFET minimal cell, 288 MOSFETs). Each variant drawn as two 4-cell sub-rows joined by the named net C4 (an off-page-style connector, a drawing-space choice only – not a different circuit). Verified against all 131072 (A,B,Cin) combinations.

B_0's edge starts dies at bit 4 (generate and propagate both 0 there), so under this vector the ripple only reaches S_4 and an internal carry node – the real 8-bit C_{out} never moves at all, and is not a flat-but-real signal, just numerical noise at the nanovolt level (confirmed directly in LTspice: probing `Cout` under this vector shows $\sim$19 nV of simulator noise around 0V, not a 0–5V logic swing). The vector was corrected to $A = 11111111$ (255, all bits constant 1) with B_0 toggling the same way: verified by plain integer addition before rerunning, $A = 255, B = 0, C_{in} = 0 \Rightarrow S = 255, C_{out} = 0$

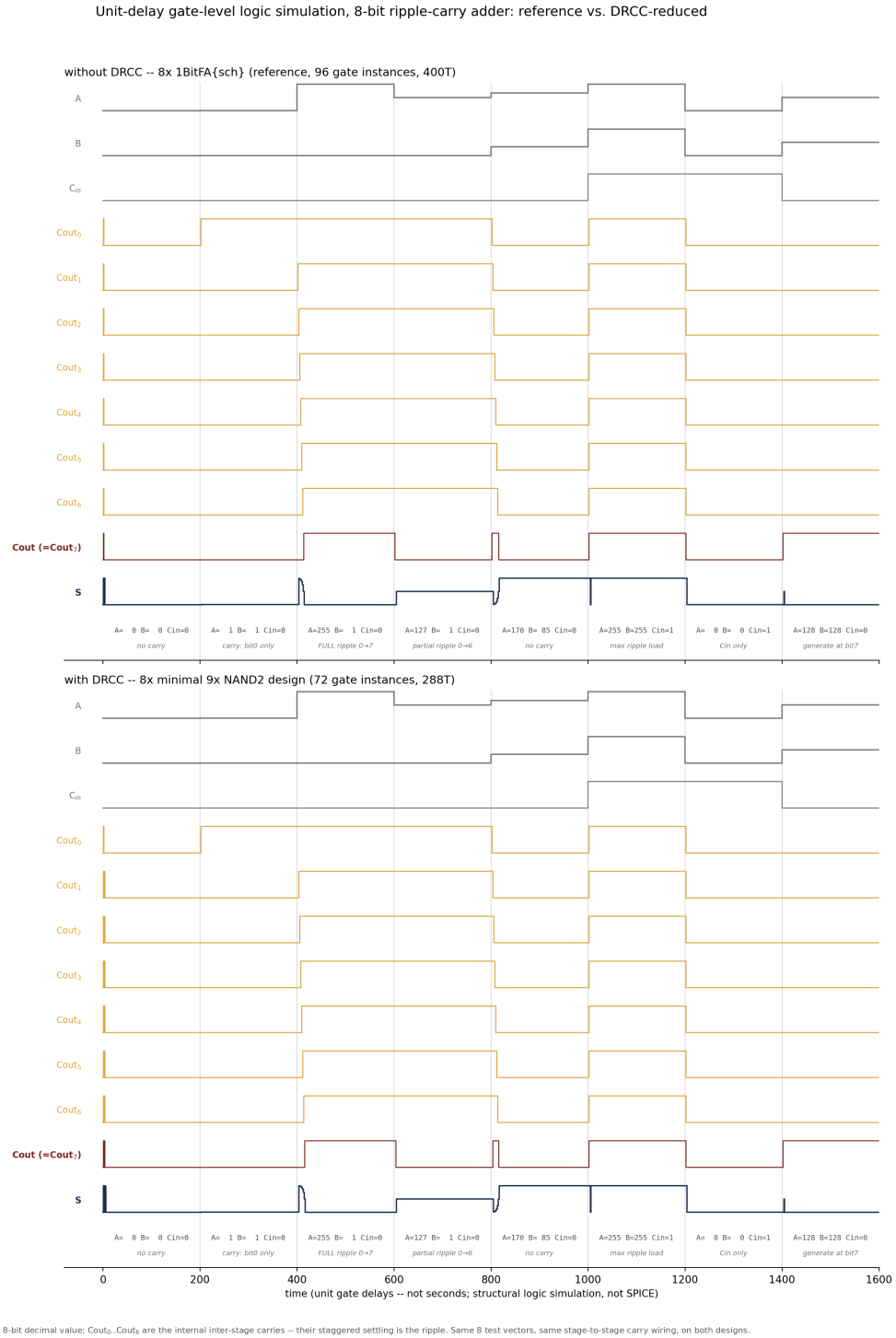


Figure 15: Unit-delay gate-level logic simulation of the 8-bit ripple-carry adder (not SPICE; x-axis is gate delays), reference (top, 400T) vs. DRCC-reduced (bottom, 288T), on 8 illustrative vectors. The seven internal inter-stage carries (Cout_0..Cout_6) plus the final Cout settle visibly staggered. All settled values verified against plain integer addition before plotting.

before the edge, and $A=255, B=1, C_{in}=0 \Rightarrow S=0, C_{out}=1$ after it – every one of the 8 stages is now in the pure carry-propagate condition, so the single B_0 edge ripples through all 8 stages and does flip the real C_{out}, the direct 8-bit analogue of the 4-bit deck's own $A=15$ vector. The same automated `.meas TRAN` directives measure the propagation delay from B_0 crossing 50% of V_{DD} to S_7 (falling) and C_{out} (rising), separately for each chain.

measured path	reference (ns)	DRCC-reduced (ns)	Δ (ps)	vs. reference
$B_0 \to S_7$ (50% V_{DD})	2.7530	2.5967	−156.26	−5.68%
$B_0 \to C_{out}$ (50% V_{DD})	2.7045	2.5440	−160.41	−5.93%

Table 17: Real transistor-level transient propagation delay, 8-bit single-edge worst-case ripple vector ($A = 255$), both chains under identical stimulus, loading, and BSIM3 model file (`8bit_singleedge_A255_with_meas.sp`, run in LTspice by Reza; `.meas TRAN` output reproduced verbatim). Consistency check: the `TRIG` time returned for all four measurements is 30.500 ns in every case, matching Table 14's own check and the expected 2.5V crossing of the B_0 edge.

Finding. The 8-bit real transient measurement confirms the 4-bit result at twice the ripple depth: the DRCC-reduced chain is −5.68% faster on $B_0 \to S_7$ and −5.93% faster on $B_0 \to C_{out}$ than the reference, under identical stimulus, loading, and device models – a larger relative margin than the 4-bit chain's −3.04%/−3.41%, consistent with (not yet proof of) the per-stage NAND2-vs-NAND3 effect proposed as a hypothesis in Section 3.5: with twice as many stages, an advantage that compounds stage-by-stage would be expected to grow, and it does.

Boundary. The same three limitations noted for the 4-bit measurement apply here unchanged: one test vector (the 8-bit single-edge worst-case ripple, not an exhaustive transition sweep), one nominal process corner (27 °C), and `{sch}`-level models on both sides with no layout-extracted parasitics. A fourth point specific to this measurement: the first vector tried ($A = 15$) is not wrong, only inapplicable to C_{out} at this bit width – it remains a valid, verified way to exercise S_4 and the bit0–3 block's internal carry, just not the full 8-deep chain, and the dead end is reported here rather than silently discarded so the vector-selection reasoning stays auditable, the same discipline applied to the rejected simultaneous-switching vector at 4 bits.

8-bit carry-lookahead The generate/propagate/sum cells are, again, verbatim reuses of Section 3.8's own cells (Table 12); only the shared lookahead network changes. Rather than stretch exp2b's small 4-bit-specific network (a 2-round hybrid, disclosed in that file's own docstring as workable at $n = 4$ but not meant as a general template) to 8 bits, `exp3b_8bit_cla.py` implements the standard, generic Kogge-Stone parallel-prefix recursion directly: 3 rounds at distances $d = 1, 2, 4$ (all previous-round values only, never an in-place update read back within the same round), verified exhaustively rather than assumed correct from the recursion alone. As at 4 bits, the network itself is identical between the two variants – confirmed directly from the verified totals: $852 - 8 \times 52 = 436$ MOSFETs (reference) and $788 - 8 \times 44 = 436$ MOSFETs (DRCC-reduced), the same lookahead-network cost on both sides.

architecture (8 bit)	reference	DRCC-reduced	saving	Cout depth (ref / DRCC)
ripple-carry	400 T	288 T	−28.00%	16 / 19
carry-lookahead	852 T	788 T	−7.51%	10 / 13

Table 18: 8-bit MOSFET totals and structural (gate-level, not SPICE) critical-path depth to Cout, both architectures, both verified against all 131072 (A,B,Cin) combinations. Same depth convention as Table 15: a topology count, not picoseconds.

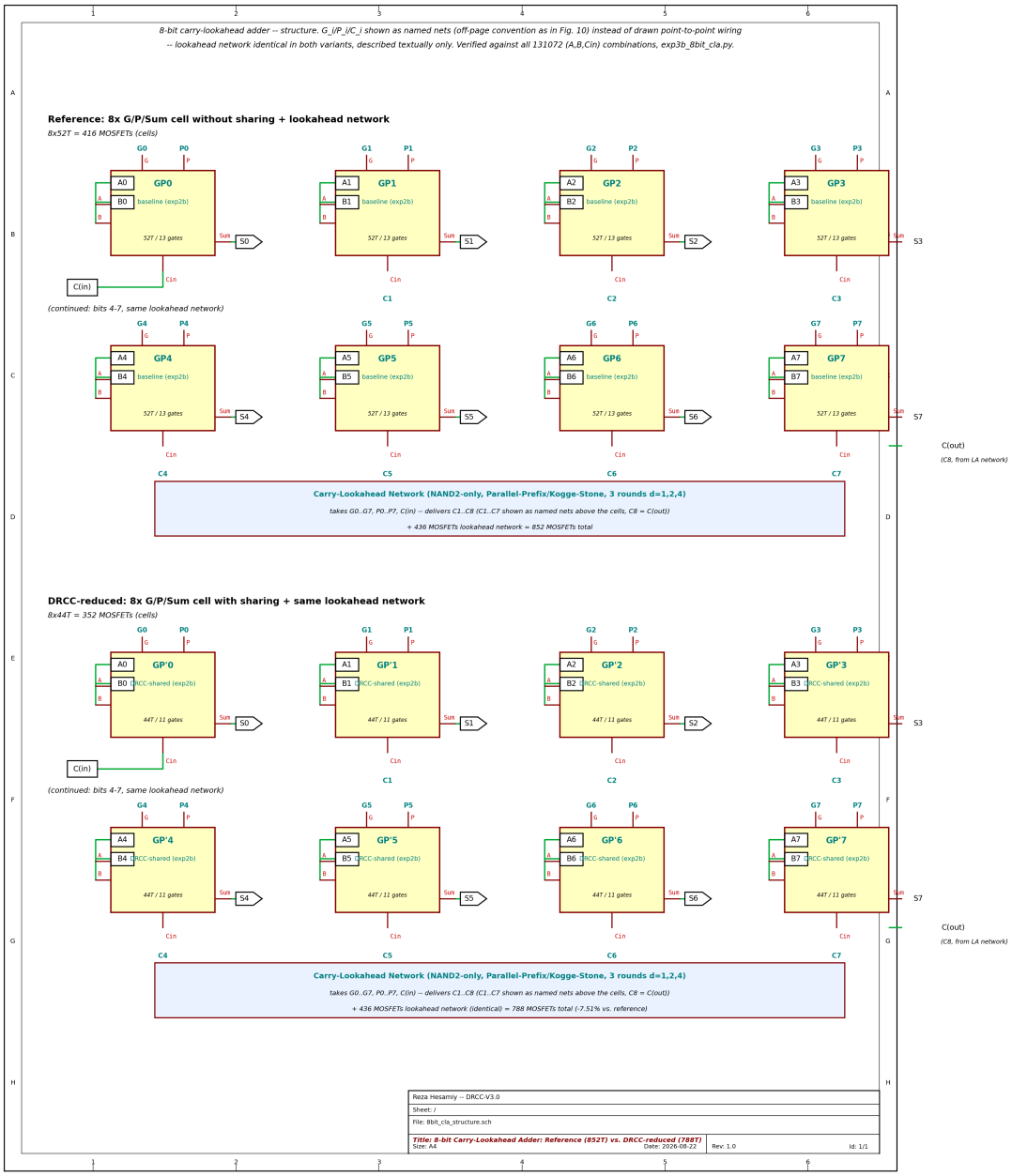


Figure 16: 8-bit carry-lookahead adder: reference (top, 8x the 52-MOSFET baseline generate/propagate/sum cell, 852 MOSFETs total including the lookahead network) vs. DRCC-reduced (bottom, 8x the 44-MOSFET shared cell, 788 MOSFETs total). $G_i/P_i/C_i$ signals shown as named nets rather than sheet-spanning point-to-point wiring (a drawing-space choice, same off-page convention as Figure 14) – the network's structure and MOSFET count are exactly what `exp3b_8bit_cla.py` verified. Verified against all 131072 (A,B,Cin) combinations.

The crossover point: the log-depth advantage reappears at 8 bits Section 3.8 reported a counterintuitive result at $n = 4$: this record's strictly NAND2-only carry-lookahead construction measured a *deeper* critical path to Cout (12/15 gate levels, reference/DRCC) than the ripple-carry chain (8/11) – the opposite of the classical asymptotic result. Table 18 shows that reverses by $n = 8$: Cout depth 10/13 (reference/DRCC) is now clearly *shallower* than ripple-carry's 16/19 at the same bit width. Ripple-carry's depth grows linearly with n (8 vs. 16 for reference, roughly doubling from 4 to 8 bits), while carry-lookahead's grows only from 2 rounds ($n = 4$) to 3 rounds ($n = 8$) of parallel-prefix combining – exactly the $O(\log n)$ vs. $O(n)$

divergence classical theory predicts, now computed directly at two concrete bit widths rather than assumed from the asymptotics alone. The per-bit DRCC saving also shrinks with scale for the same structural reason noted at 4 bits: the shared lookahead network's fixed MOSFET cost (436, unchanged by DRCC) is a larger fraction of a bigger total, so the overall saving falls from -8.99% at 4 bits to -7.51% at 8 bits even though the per-bit cell saving (52 → 44 MOSFETs) is identical at both scales.

Boundary. The $n = 4$ and $n = 8$ lookahead networks are not byte-identical code scaled up mechanically – `exp2b_4bit_cla.py`'s own docstring discloses that its 4-bit network is a small hybrid construction (a partially-serial first round, corrected by a second round), workable at $n = 4$ but not offered there as a general template, whereas `exp3b_8bit_cla.py` implements the standard textbook Kogge-Stone recursion directly. Both are independently verified, NAND2-only, parallel-prefix constructions, and the depth numbers compared above are each exactly what was structurally computed from its own chained, exhaustively-checked netlist – but the crossover claimed here is between two related, not perfectly matched, network implementations, and this is disclosed rather than left implicit.

Finding. Carry-lookahead's classical $O(\log n)$ depth advantage over ripple-carry's $O(n)$, absent at $n = 4$ under this record's strict NAND2-only technology constraint, reappears by $n = 8$: Cout depth 10/13 (CLA, reference/DRCC) versus 16/19 (ripple), computed directly from verified netlists at both bit widths rather than assumed from asymptotic theory alone.

3.10 Scaling the full adder to 16 bits: ripple-carry and carry-lookahead

Section 3.9 scaled the same verified 1-bit cells to 8 bits and re-checked from scratch rather than assuming the 4-bit result transfers. At 16 bits the same discipline runs into a hard limit: exhaustive brute force would require checking $2 \times 65536 \times 65536 = 8{,}589{,}934{,}592$ (A, B, C_{in}) combinations, not tractable by a Python loop in this environment (unlike 4 bits' 512 and 8 bits' 131072). This is disclosed as a methodology change, not glossed over: correctness at 16 bits rests on four things together rather than one brute-force sweep (`exp4a_16bit_ripple.py`, `exp4b_16bit_cla.py`).

Boundary. Four checks replace exhaustive brute force at 16 bits. (1) Every gate-level building block used – both 1-bit full-adder cells, and for carry-lookahead every primitive (`AND2`, `OR2`, `OR_AND`, the group-combine `dot()`, both generate/propagate cells, both sum cells) – is checked against its exact truth table over *all* of its own input combinations; this remains fully exhaustive, just at the (small) per-component level. (2) A formal argument then carries that up to the full width: a chain of 1-bit cells that each exactly satisfy the full-adder equation computes correct n-bit addition for every input, by induction on chain position; a Kogge-Stone network built from a correctly implemented associative combine computes correct prefix carries for every input, by the standard parallel-prefix correctness argument. Neither argument is empirical – both are proofs about the wiring rule, assuming the boolean simulation itself has no bug. (3) That remaining assumption is what 3,000,000 uniformly random (A, B, C_{in}) samples per variant, run through the actual chain-building code, are for – an implementation bug in the Python loop (a wrong carry wire, an off-by-one) would show up here even though the induction argument cannot see it. (4) Targeted structured vectors (all-zero, all-one, every single bit set, alternating patterns, the full

worst-case ripple) cover deterministic edge cases a uniform random sample can under-sample. On top of all four, the exact same `ripple_n()`/`lookahead_n()` code is re-run at $n = 4$ and $n = 8$ and required to reproduce exp2a's and exp3a's/exp3b's exact published MOSFET counts and Cout depths before its $n = 16$ output is trusted at all – unchanged from the discipline already applied when scaling from 4 to 8 bits.

Table 19 makes the methodology change concrete: at 1, 4, and 8 bits every single (A, B, C_{in}) combination is checked (exhaustive, 100% coverage); at 16 bits the input space is too large for that, and the check becomes 3,000,000 uniformly random samples per variant (Section 3.10, point 3); 6,000,000 total across the reference and DRCC variants of each circuit. Wall-clock time was measured once, in this environment, as an order-of-magnitude illustration for the reader – not a performance benchmark, and not averaged over repeated runs. The two 16-bit rows process the same 6,000,000 samples at very different speed because they are not doing the same amount of work per sample: the ripple-carry check chains 1-bit full-adder cells, while the carry-lookahead check additionally builds every primitive gate (`AND2`, `OR2`, `OR_AND`, `dot()`, both generate/propagate cells, both sum cells) and the full four-round Kogge-Stone network for every sample – the time difference reflects that extra per-sample construction cost, not a difference in verification rigor.

Width	Combinations checked	Coverage	Wall-clock time
1-bit (cell)	8	exhaustive (100%)	0.03 s
4-bit	512	exhaustive (100%)	0.04 s
8-bit	131,072	exhaustive (100%)	5.9 s
16-bit, ripple-carry	3,000,000	random sample	5 min 3 s
16-bit, carry-lookahead	3,000,000	random sample	15 min 34 s

Table 19: Verification effort by width, applied identically to each variant (reference and DRCC-reduced) of the circuit; the 16-bit wall-clock times reflect the full run over both variants (6,000,000 total evaluations per row), single measured run in this environment (`exp1c`/`exp1h`, `exp2a`, `exp3a`, `exp4a_16bit_ripple.py`, `exp4b_16bit_cla.py`). The 1-, 4-, and 8-bit rows are exhaustive sweeps, not samples; only the 16-bit rows are random sampling, per the methodology change disclosed above. Absolute times depend on the host machine and are not comparable across circuit types, only as an illustration of how verification effort scales with width for a fixed methodology.

16-bit ripple-carry Sixteen instances of the same verified full-adder cell, chained identically to Figures 10 and 14. No separate structural schematic is drawn at 16 bits and beyond: Figure 14 already shows how large a hand-drawn diagram in this style becomes at just 8 bits, and 16 bits is the identical four-cell sub-row pattern, joined the same way, simply repeated twice as many times – nothing structurally new to draw. MOSFET count stays additive: the reference totals $16 \times 50 = 800$ MOSFETs, the DRCC-reduced chain totals $16 \times 36 = 576$ MOSFETs – the same -28.00% saving as at 1, 4, and 8 bits, now on a chain whose correctness rests on the four-part argument above rather than brute force.

Finding. The reduction ratio holds exactly at 16-bit scale too: 800 → 576 MOSFETs, 28% fewer transistors, now checked at four independent scales (1, 4, 8, 16 bits) – the last one necessarily by a different, disclosed verification methodology rather than by extending the same brute-force sweep indefinitely.

variant	MOSFETs	gates	vs. ref.	status
16-bit reference	800	192	–	verified (Section 3.10)
16-bit DRCC-reduced	576	144	−28.00%	verified (Section 3.10)

Table 20: Same reduction as Tables 13/16, additive across the 16 independent cell instances of the 16-bit ripple-carry chain – checked per Section 3.10's methodology (exhaustive per-cell truth tables, 3,000,000 random (A, B, C_{in}) samples, 39 targeted vectors, all per variant), not the exhaustive sweep of Tables 13/16 (`exp4a_16bit_ripple.py`).

What the ripple actually looks like at 16 bits The same unit-delay logic simulation used at 4- and 8-bit scale (Figures 12/15), chained the same way, was run again at 16 bits on the direct 16-bit analogues of the same 8 vectors (no carry, a single-stage carry, the full 16-stage worst-case ripple, a partial ripple stopped by a killed top bit, an alternating no-carry pattern, maximum carry load, Cin alone, and a carry generated directly at the top bit). As at 8 bits, every settled `Sum`/`Cout`, both circuits, was checked against plain integer addition – 8/8 correct in both cases. The trace itself is not plotted separately here, for the same reason the structural schematic above is not: Figure 15 already shows what this looks like, with the internal inter-stage carries (Cout_0..Cout_{14} at this width) settling visibly staggered, more so than at 8 bits.

From ticks to nanoseconds, a third time: a real 16-bit SPICE transient measurement The same real BSIM3 device models and LTspice setup used for the 4-bit and 8-bit measurements above (Tables 14, 17) were extended to 16 bits, again reusing the identical subcircuits chained twice as long as the 8-bit deck: the reference chain is 16 instances of the real `1BitFA{sch}` cell (800 MOSFETs total), the DRCC-reduced chain is 16 instances of the proven-minimal 9x-NAND2 cell (576 MOSFETs total), both measured outputs carrying an identical 50 fF load on both chains.

Having already learned the vector-selection lesson at 8 bits, the vector was chosen correctly on the first attempt this time: $A = 1111111111111111$ (65535, all 16 bits constant 1) with B_0 toggling – the direct generalization of the 8-bit deck's own $A = 255$ vector. Verified by plain integer addition before the deck was written: $A = 65535, B = 0, C_{in} = 0 \Rightarrow S = 65535, C_{out} = 0$ before the edge, and $A = 65535, B = 1, C_{in} = 0 \Rightarrow S = 0, C_{out} = 1$ after it – every one of the 16 stages is in the pure carry-propagate condition ($A_i = 1, B_i = 0$ regenerates carry = 1 at every stage), so the single B_0 edge ripples through all 16 stages and flips the real C_{out}. The same automated `.meas TRAN` directives measure the propagation delay from B_0 crossing 50% of V_{DD} to S_{15} (falling) and C_{out} (rising), separately for each chain.

One new detail in this run, disclosed rather than omitted: LTspice's direct Newton iteration failed to find the DC operating point for the 16-bit netlist (twice the series depth of the 8-bit deck), and fell back automatically to Gmin stepping, which succeeded. This is a standard SPICE convergence method for stiff DC operating points, not an error, and it has no bearing on the transient measurement that follows it.

Finding. The 16-bit real transient measurement extends the trend to a third, independently measured width: the DRCC-reduced chain is −7.05% faster on $B_0 \to S_{15}$ and −7.18% faster on $B_0 \to C_{out}$ than the reference – larger again than the 8-bit chain's −5.68%/−5.93%, itself larger than the 4-bit chain's −3.04%/−3.41%. The margin has now grown monotonically at every doubling of width measured in real SPICE, consistent with (still not proof of) the per-stage NAND2-vs-NAND3 hypothesis from Section 3.5 compounding stage-by-stage.

measured path	reference (ns)	DRCC-reduced (ns)	Δ (ps)	vs. reference
$B_0 \to S_{15}$ (50% V_{DD})	5.4313	5.0486	−382.72	−7.05%
$B_0 \to C_{out}$ (50% V_{DD})	5.3825	4.9958	−386.69	−7.18%

Table 21: Real transistor-level transient propagation delay, 16-bit single-edge worst-case ripple vector ($A = 65535$), both chains under identical stimulus, loading, and BSIM3 model file (`16bit_singleedge_A65535_with_meas.sp`, run in LTspice by Reza; `.meas TRAN` output reproduced verbatim). Consistency check: the `TRIG` time returned for all four measurements is 30.500 ns in every case, matching Tables 14 and 17's own checks and the expected 2.5V crossing of the B_0 edge.

Boundary. The same three limitations noted for the 4-bit and 8-bit measurements apply here unchanged: one test vector (the 16-bit single-edge worst-case ripple, not an exhaustive transition sweep), one nominal process corner (27 °C), and `{sch}`-level models on both sides with no layout-extracted parasitics. Unlike at 8 bits, there is no vector dead end to report here – the lesson from the 8-bit $A = 15$ attempt (an all-ones A is needed to keep every stage in the carry-propagate condition) transferred directly, itself a small piece of evidence that the vector-selection reasoning generalizes rather than needing to be re-discovered by trial and error at each new width.

16-bit carry-lookahead The generate/propagate/sum cells are, again, verbatim reuses of Section 3.8's own cells (Table 12); only the shared lookahead network changes, and here it changes less than at any previous step: `exp4b_16bit_cla.py` reuses `exp3b_8bit_cla.py`'s `lookahead_n()` *verbatim, unchanged code*, simply called with $n = 16$ instead of $n = 8$, which resolves to 4 rounds at distances $d = 1, 2, 4, 8$ instead of 3. Unlike the 4-to-8-bit step – where exp2b's 4-bit-specific hybrid network and exp3b's generic Kogge-Stone network were two different, if related, constructions – the 8-to-16-bit step compares the exact same network code at two widths, a stronger basis for comparison. As at 4 and 8 bits, the network is identical between the two variants: the verified totals give $2004 - 16 \times 52 = 1172$ MOSFETs (reference) and $1876 - 16 \times 44 = 1172$ MOSFETs (DRCC-reduced), the same lookahead cost on both sides, and exactly what the closed-form network-cost formula implied by exp3b's own 8-bit figure predicts ($20 \sum_{d \in \{1,2,4,8\}} (16 - d) + 12 \times 16 = 1172$).

As with the ripple-carry chain above, no separate structural schematic is drawn at 16 bits: Figure 16 already shows the generate/propagate/sum-cell-plus-lookahead-network drawing style at 8 bits, and 16 bits repeats the identical pattern with twice as many per-bit cells and one additional lookahead round – a longer drawing, not a different one.

architecture (16 bit)	reference	DRCC-reduced	saving	Cout depth (ref / DRCC)
ripple-carry	800 T	576 T	−28.00%	32 / 35
carry-lookahead	2004 T	1876 T	−6.39%	12 / 15

Table 22: 16-bit MOSFET totals and structural (gate-level, not SPICE) critical-path depth to Cout, both architectures, verified per Section 3.10's methodology. Same depth convention as Tables 15/18: a topology count, not picoseconds.

The gap widens: log-depth advantage grows at 16 bits Section 3.9 found that carry-lookahead's classical $O(\log n)$ depth advantage, absent at $n = 4$ under this record's strict NAND2-only constraint, reappears by $n = 8$ (Cout depth 10/13 vs. ripple's 16/19). Table 22 shows the gap widening further at $n = 16$: Cout depth 12/15 (CLA, reference/DRCC) against ripple-carry's 32/35 – CLA's depth grew by only 2 (from 10 to 12, reference) while ripple-carry's exactly doubled (16 to 32), matching $O(\log n)$ vs. $O(n)$ even more clearly than the 4-vs-8-bit comparison could, and on firmer ground: unlike that comparison, the 8-bit and 16-bit CLA networks are the identical `lookahead_n()` code at two values of n, not two related-but-different constructions. The per-bit DRCC saving on carry-lookahead continues the trend already noted at 4 and 8 bits: it shrinks with scale for the same structural reason – the shared lookahead network's fixed MOSFET cost (1172, unchanged by DRCC, and growing with $n \log n$) is a larger fraction of a bigger total, so the overall saving falls from −8.99% (4 bits) to −7.51% (8 bits) to −6.39% (16 bits) even though the per-bit cell saving (52 → 44 MOSFETs) is identical at all three scales.

Finding. Carry-lookahead's $O(\log n)$ depth advantage over ripple-carry's $O(n)$ widens as bit width grows: Cout depth 12/15 (CLA) versus 32/35 (ripple) at $n = 16$, a wider relative gap than at $n = 8$ (10/13 vs. 16/19), computed from the identical lookahead-network code at both widths rather than from two related-but-different implementations as in the 4-to-8-bit comparison.

3.11 Scaling the full adder to 32 bits: ripple-carry and carry-lookahead

Section 3.10 scaled the same verified cells to 16 bits and replaced exhaustive brute force with a disclosed four-part methodology once the input space became intractable. At 32 bits the input space grows again, by a factor of 2^{32} over 16 bits: exhaustive checking would require $2 \times (2^{32}) \times (2^{32}) = 2^{65} = 36{,}893{,}488{,}147{,}419{,}103{,}232$ (A, B, C_{in}) combinations, even further out of reach than 16 bits' already-intractable 8,589,934,592. The same four-part methodology is applied again, unchanged in kind (`exp5a_32bit_ripple.py`, `exp5b_32bit_cla.py`).

Boundary. The same four checks used at 16 bits are used again here, and nothing about the method itself changes: (1) exhaustive per-cell/per-primitive truth-table checks, (2) the same two formal arguments (induction over chain position for ripple-carry; parallel-prefix correctness for the Kogge-Stone network), (3) 3,000,000 uniformly random (A, B, C_{in}) samples per variant through the actual chain-building code, and (4) targeted structured vectors. What is new at this scale is the cross-check requirement: the same `ripple_n()` code is now re-run at $n = 4$, $n = 8$, *and* $n = 16$ and required to reproduce exp2a's, exp3a's, *and* exp4a's exact published numbers before its $n = 32$ output is trusted; the same `lookahead_n()`/`cla_n_*()` code is re-run at $n = 8$ *and* $n = 16$ and required to reproduce exp3b's and exp4b's exact published numbers. The $n = 16$ cross-check is itself necessarily a reproduction of a *random-sample* result, not an exhaustive one – exp4a's and exp4b's own $n = 16$ numbers were themselves obtained by random sampling, not brute force, so this is the first point in the record where a cross-check target is itself a previously-sampled result rather than a previously-exhaustive one. Both scripts reproduced every one of these published numbers exactly before proceeding to their own $n = 32$ result.

32-bit ripple-carry Thirty-two instances of the same verified full-adder cell, chained identically to Figures 10 and 14 (16 and 32 bits are not drawn separately, for the reason given above – Figure 14 already shows the pattern, which just repeats further). MOSFET count stays additive:

the reference totals $32 \times 50 = 1600$ MOSFETs, the DRCC-reduced chain totals $32 \times 36 = 1152$ MOSFETs – the same -28.00% saving as at every smaller width, now on a chain whose correctness rests on the four-part argument plus a three-way cross-check (against $n = 4$, $n = 8$, and $n = 16$) rather than brute force.

variant	MOSFETs	gates	vs. ref.	status
32-bit reference	1600	384	–	verified (Section 3.11)
32-bit DRCC-reduced	1152	288	-28.00%	verified (Section 3.11)

Table 23: Same reduction as Tables 13/16/20, additive across the 32 independent cell instances of the 32-bit ripple-carry chain – checked per Section 3.11's methodology (exhaustive per-cell truth tables, 3,000,000 random (A, B, C_{in}) samples, 71 targeted vectors, all per variant), not the exhaustive sweep of Tables 13/16 (`exp5a_32bit_ripple.py`).

Finding. The reduction ratio holds exactly at 32-bit scale too: 1600 → 1152 MOSFETs, 28% fewer transistors, now checked at five independent scales (1, 4, 8, 16, 32 bits) – the last two necessarily by the disclosed random-sampling methodology rather than by extending the same brute-force sweep indefinitely.

What the ripple actually looks like at 32 bits The same unit-delay logic simulation used at 4- and 8-bit scale (Figures 12/15), and again at 16 bits, was run once more at 32 bits on the direct 32-bit analogues of the same 8 vectors (no carry, a single-stage carry, the full 32-stage worst-case ripple, a partial ripple stopped by a killed top bit, an alternating no-carry pattern, maximum carry load, Cin alone, and a carry generated directly at the top bit). As at 8 and 16 bits, every settled `Sum`/`Cout`, both circuits, was checked against plain integer addition – 8/8 correct in both cases. The trace is not plotted separately here, for the same reason as above: Figure 15 already shows what this looks like, with the internal inter-stage carries (Cout_0..Cout_{30} at this width) settling visibly staggered, more so than at 16 bits.

From ticks to nanoseconds, a fourth time: a real 32-bit SPICE transient measurement The same real BSIM3 device models and LTspice setup used for the 4-bit, 8-bit, and 16-bit measurements above (Tables 14, 17, 21) were extended to 32 bits, again reusing the identical subcircuits chained twice as long as the 16-bit deck: the reference chain is 32 instances of the real `1BitFA{sch}` cell (1600 MOSFETs total), the DRCC-reduced chain is 32 instances of the proven-minimal 9x-NAND2 cell (1152 MOSFETs total), both measured outputs carrying an identical 50 fF load on both chains.

Having already learned the vector-selection lesson at 8 bits and confirmed it transferred cleanly at 16 bits, the vector was again chosen correctly on the first attempt: $A = 4294967295$ (all 32 bits constant 1) with B_0 toggling – the direct generalization of the 16-bit deck's own $A = 65535$ vector. Verified by plain integer addition before the deck was written: $A = 4294967295, B = 0, C_{in} = 0 \Rightarrow S = 4294967295, C_{out} = 0$ before the edge, and $A = 4294967295, B = 1, C_{in} = 0 \Rightarrow S = 0, C_{out} = 1$ after it – every one of the 32 stages is in the pure carry-propagate condition, so the single B_0 edge ripples through all 32 stages and flips the real C_{out}. The same automated `.meas TRAN` directives measure the propagation delay from B_0 crossing 50% of V_{DD} to S_{31} (falling) and C_{out} (rising), separately for each chain.

As at 16 bits, LTspice's direct Newton iteration failed to find the DC operating point for this even deeper netlist and fell back automatically to Gmin stepping, which succeeded (total elapsed

time 2.652 seconds – longer than 16 bits' 1.175 seconds, consistent with a deeper, stiffer series network, and still not an error).

measured path	reference (ns)	DRCC-reduced (ns)	Δ (ps)	vs. reference
$B_0 \to S_{31}$ (50% V_{DD})	10.7884	9.9518	−836.59	−7.75%
$B_0 \to C_{out}$ (50% V_{DD})	10.7395	9.8992	−840.30	−7.82%

Table 24: **36T-minimal chain**: real transistor-level transient propagation delay, 32-bit single-edge worst-case ripple vector ($A = 4294967295$), the 36-MOSFET-per-bit DRCC-minimal chain of Section 3.5 vs. the 50-MOSFET-per-bit reference, both chains under identical stimulus, loading, and BSIM3 model file (`32bit_singleedge_A4294967295_with_meas.sp`, run in LTspice by Reza; `.meas TRAN` output reproduced verbatim). Consistency check: the `TRIG` time returned for all four measurements is 30.500 ns in every case, matching Tables 14/17/21's own checks and the expected 2.5V crossing of the B_0 edge.

Finding. The 32-bit real transient measurement extends the trend to a fourth, independently measured width: the DRCC-reduced chain is −7.75% faster on $B_0 \to S_{31}$ and −7.82% faster on $B_0 \to C_{out}$ than the reference – larger again than the 16-bit chain's −7.05%/−7.18%, itself larger than the 8-bit chain's −5.68%/−5.93% and the 4-bit chain's −3.04%/−3.41%. The margin has now grown monotonically at every doubling of width measured in real SPICE – 4, 8, 16, and now 32 bits – consistent with (still not proof of) the per-stage NAND2-vs-NAND3 hypothesis from Section 3.5 compounding stage-by-stage.

Boundary. The same three limitations noted for the 4-, 8-, and 16-bit measurements apply here unchanged: one test vector (the 32-bit single-edge worst-case ripple, not an exhaustive transition sweep), one nominal process corner (27 °C), and `{sch}`-level models on both sides with no layout-extracted parasitics. As at 16 bits, there is no vector dead end to report here – the lesson from the 8-bit $A = 15$ attempt transferred directly a second time, itself further evidence that the vector-selection reasoning generalizes across doublings rather than needing rediscovery at each new width.

32-bit carry-lookahead The generate/propagate/sum cells are, again, verbatim reuses of Section 3.8's own cells (Table 12); only the shared lookahead network changes, and again only in the number of Kogge-Stone rounds: `exp5b_32bit_cla.py` reuses the same `lookahead_n()` already reused verbatim by `exp3b_8bit_cla.py` and `exp4b_16bit_cla.py` *verbatim, unchanged code*, simply called with $n = 32$ instead of $n = 16$, which resolves to 5 rounds at distances $d = 1, 2, 4, 8, 16$ instead of 4. As at every smaller width, the network is identical between the two variants: the verified totals give $4628 - 32 \times 52 = 2964$ MOSFETs (reference) and $4372 - 32 \times 44 = 2964$ MOSFETs (DRCC-reduced), the same lookahead cost on both sides, and this time an *exact*, code-computed match to the closed-form network-cost formula ($20 \sum_{d \in \{1,2,4,8,16\}} (32 - d) + 12 \times 32 = 2964$) rather than a hand-projected prediction awaiting confirmation.

As at 16 bits, no separate structural schematic is drawn here either: Figure 16 already shows the drawing style, and 32 bits is the same generate/propagate/sum cell repeated with one further lookahead round – a longer drawing, not a different one.

architecture (32 bit)	reference	DRCC-reduced	saving	Cout depth (ref / DRCC)
ripple-carry	1600 T	1152 T	−28.00%	64 / 67
carry-lookahead	4628 T	4372 T	−5.53%	14 / 17

Table 25: 32-bit MOSFET totals and structural (gate-level, not SPICE) critical-path depth to Cout, both architectures, verified per Section 3.11's methodology. Same depth convention as Tables 15/18/22: a topology count, not picoseconds.

The gap widens even further: log-depth advantage at 32 bits Table 25 continues the trend Section 3.10 found widening at 16 bits. Ripple-carry's reference Cout depth exactly doubles at every step computed so far – 8 (4-bit) → 16 (8-bit) → 32 (16-bit) → 64 (32-bit) – consistent with its $O(n)$ critical path. Carry-lookahead's reference Cout depth, by contrast, has now grown by exactly 2 gate levels at every doubling of n computed with the identical `lookahead_n()` code: 10 ($n = 8$) → 12 ($n = 16$) → 14 ($n = 32$) – one additional Kogge-Stone round per doubling, each round contributing exactly two `NAND2` gate delays (one through `dot()`'s `AND2`, one through its `OR_AND`). This is the widest gap yet between the two architectures under this record's strict NAND2-only constraint: a 64/67 ripple-carry depth against a 14/17 carry-lookahead depth at the same 32-bit width. The per-bit DRCC saving on carry-lookahead continues shrinking with scale for the same structural reason noted at 4, 8, and 16 bits: the shared lookahead network's fixed MOSFET cost (2964, unchanged by DRCC, growing with $n \log n$) is a larger fraction of a bigger total, so the overall saving falls from −8.99% (4 bits) to −7.51% (8 bits) to −6.39% (16 bits) to −5.53% (32 bits) even though the per-bit cell saving (52 → 44 MOSFETs) is identical at all four scales.

Finding. Carry-lookahead's $O(\log n)$ depth advantage over ripple-carry's $O(n)$ widens further at 32 bits: Cout depth 14/17 (CLA) versus 64/67 (ripple), the widest gap computed so far in this record, from the identical lookahead-network code run at three bit widths (8, 16, and 32). The per-bit DRCC saving on carry-lookahead has now shrunk monotonically across all four evaluated widths (−8.99% → −7.51% → −6.39% → −5.53%) while the ripple-carry saving has stayed exactly −28.00% at every one of the five widths evaluated (1, 4, 8, 16, 32 bits) – the two architectures respond to the same per-bit cell reduction in structurally different ways as n grows.

3.12 Cross-scale full-adder summary: ripple-carry and carry-lookahead, 1 to 32 bits

Sections 3.5–3.11 built up the same two architectures independently at five scales (1, 4, 8, 16, 32 bits), each re-verified from scratch rather than assumed to generalize. This subsection puts the resulting numbers side by side in one place – no new circuit, no new measurement, only the already-verified results of Tables 5, 15, 18, 22, 25, 7 and Tables 14, 17, 21, 24, gathered for comparison.

Boundary. Three distinct real one-bit timing results are reported in this record. Two belong to the same transistor-count-minimal 36-MOSFET cell but use different single-edge input paths: the $A : 0 \to 1$ experiment reported in Table 7, and the $C_{\text{in}} : 0 \to 1$ experiment reported in Section 3.6. Their numerical values are therefore not expected to coincide. A third one-bit result belongs to the separate 48-MOSFET speed-optimized realization of Section 3.6. This cell deliberately trades part of the transistor-count reduction for parallel drive strength and local NMOS sizing. The

width	architecture	reference	DRCC-reduced	saving	Cout depth (ref / DRCC)
1-bit†	single cell	50 T	36 T	−28.00%	2 / 5†
4-bit	ripple-carry	200 T	144 T	−28.00%	8 / 11
4-bit	carry-lookahead	356 T	324 T	−8.99%	12 / 15
8-bit	ripple-carry	400 T	288 T	−28.00%	16 / 19
8-bit	carry-lookahead	852 T	788 T	−7.51%	10 / 13
16-bit	ripple-carry	800 T	576 T	−28.00%	32 / 35
16-bit	carry-lookahead	2004 T	1876 T	−6.39%	12 / 15
32-bit	ripple-carry	1600 T	1152 T	−28.00%	64 / 67
32-bit	carry-lookahead	4628 T	4372 T	−5.53%	14 / 17

Table 26: Structural (MOSFET count, gate-level Cout depth) results for the **36T transistor-minimal family** at all five scales, structurally computed and verified in this record, reproduced verbatim from the tables cited above – no figure in this table is newly computed. The separate 48T speed-optimized family is reported in Section 3.6, not in this table. †The 1-bit row is a single full-adder cell, not a chain: its depth convention (Table 6) counts gate levels within one cell, not across a ripple chain, and is not directly comparable to the n-bit rows' Cout depth; it is included only for the MOSFET-saving comparison.

width	measured path	reference (ns)	DRCC-reduced (ns)	Δ (ps)	vs. reference
1-bit‡	$A \rightarrow$ Sum	0.6259	0.7081	+82.2	+13.13%
1-bit‡	$A \rightarrow C_{out}$	0.3953	0.4541	+58.8	+14.87%
4-bit	$B_0 \rightarrow S_{n-1}$	1.4141	1.3711	−42.99	−3.04%
4-bit	$B_0 \rightarrow C_{out}$	1.3651	1.3186	−46.54	−3.41%
8-bit	$B_0 \rightarrow S_{n-1}$	2.7530	2.5967	−156.26	−5.68%
8-bit	$B_0 \rightarrow C_{out}$	2.7045	2.5440	−160.41	−5.93%
16-bit	$B_0 \rightarrow S_{n-1}$	5.4313	5.0486	−382.72	−7.05%
16-bit	$B_0 \rightarrow C_{out}$	5.3825	4.9958	−386.69	−7.18%
32-bit	$B_0 \rightarrow S_{n-1}$	10.7884	9.9518	−836.59	−7.75%
32-bit	$B_0 \rightarrow C_{out}$	10.7395	9.8992	−840.30	−7.82%

Table 27: Real transistor-level transient propagation delay results for the **36T transistor-minimal family** at all five SPICE-measured scales (the separate 48T speed-optimized family is reported in Section 3.6), reproduced verbatim from Table 7's underlying deck and Tables 14/17/21/24. The 4/8/16/32-bit rows are the single-edge worst-case ripple vector for that width, on the *ripple-carry* architecture only – no real SPICE measurement exists anywhere in this record for the carry-lookahead architecture, at any width. ‡The 1-bit row is the exact-minimal 36-MOSFET `_DRCC__1BitFA` cell from Section 3.5 – the same cell the 4/8/16/32-bit ripple chains above are built from – measured with `1bit_singleedge_with_meas.sp` ($A : 0 \rightarrow 1$, identical 50 fF loads on both outputs of both circuits, same C5 BSIM3 models [16]). It is *not* the separate, speed-optimized 48-MOSFET cell of Section 3.6; see the note below the table.

three measurements are therefore complementary rather than interchangeable: two characterize different input paths of the same 36-MOSFET cell, while the third characterizes a different optimized 48-MOSFET topology.

The two 32-bit results in this record – $-7.82\%/-7.75\%$ here for the 36-MOSFET-per-bit chain, $-27.53\%/-26.85\%$ in Section 3.6 for the 48-MOSFET-per-bit chain – are, likewise, two different circuits measured under two different vectors, not two measurements of the same design, and are not interchangeable or averageable.

Finding. Two structural quantities and one real-SPICE quantity move in three different directions as width scales from 1 to 32 bits, and this record computed the two structural quantities and measured the transient-delay quantity directly, rather than assuming any of them from the others. (1) The ripple-carry MOSFET saving is exactly -28.00% at every one of the five scales measured – flat, because both the reference and DRCC-reduced cell costs are strictly additive across independent, identical stages. (2) The carry-lookahead MOSFET saving shrinks monotonically with scale ($-8.99\% \to -7.51\% \to -6.39\% \to -5.53\%$), because the shared lookahead network's MOSFET cost is fixed per design and grows with $n \log n$ while DRCC only touches the per-bit generate/propagate/sum cells, so the untouched network becomes a larger share of an ever-larger total. (3) The real, SPICE-measured transient-delay advantage of the DRCC-reduced *ripple-carry* chain over its reference grows monotonically with scale in the opposite direction ($-3.04\%/-3.41\% \to -5.68\%/-5.93\% \to -7.05\%/-7.18\% \to -7.75\%/-7.82\%$), consistent with (not proof of) the per-stage NAND2-vs-NAND3 hypothesis compounding stage-by-stage. Extending this back to the 1-bit row now closes the picture rather than leaving it as an assumption: the same 36-MOSFET cell that is +13–15% *slower* than the reference in isolation becomes $-3.04\%/-3.41\%$ faster once chained to just 4 bits, and the advantage grows monotonically from there. This pattern is *consistent with* (not proven by) a single-cell fan-out cost (Section 3.5) that is fixed per cell, against a reference structural cost that the manuscript's own per-stage NAND2-vs-NAND3 argument suggests compounds with additional stages; no per-stage decomposition of the measured delay was performed to isolate the two effects, so this crossover explanation remains a plausible account of the sign reversal, not a demonstrated mechanism. A MOSFET-count saving that stays constant and a real timing saving that starts negative, crosses zero, and then grows are two different claims, made from two different kinds of measurement (structural counting vs. transistor-level transient simulation), and this record keeps them visibly separate rather than treating a flat transistor-count ratio as evidence about delay.

Boundary. This subsection introduces no new circuit and no new measurement beyond the 1-bit row sourced from Section 3.5's own deck – it is a restatement, side by side, of numbers each individually verified and disclosed already in Sections 3.5–3.11. Two gaps carry over unchanged and are restated here for visibility rather than left implicit in a table: no real SPICE transient measurement exists for the carry-lookahead architecture at any width; and every real SPICE number in Table 27 is a single test vector (the single-edge worst-case ripple, or for the 1-bit row the corresponding single-input-edge vector) at one nominal process corner (27 °C), not an exhaustive transition sweep or a multi-corner study, at any width alike. A third, previously real gap – no 1-bit SPICE measurement at all – is closed as of this section; a second, separately-optimized 1-bit cell and its own 32-bit chain are reported in Section 3.6 and are not folded into this table (see the boundary above the Finding).

4 Symmetry-Breaking as an Instance of Controlled Reduction

4.1 Setup

The Pigeonhole family PHP(m, n), $m = n + 1$ pigeons into n holes, is unsatisfiable for every n and is the standard example of a family that is exponentially hard for tree-like resolution search

in the absence of symmetry information [1]. Its n holes are fully interchangeable, which makes it a clean instantiation of a controlled reduction: restrict pigeon i to holes $1, \ldots, \min(i, n)$ [2], exploiting exactly the same task-relative-equivalence argument as Section 2 – states related by an admissible hole permutation that preserves the already-fixed prefix are task-relative future-equivalent for the UNSAT decision task.

Write S_{sym} for this restriction and Φ_{PHP} for the unreduced Pigeonhole formula. Its correctness as a symmetry-breaking predicate does not follow from the fact that every tested instance happens to be UNSAT: appending any clauses at all to an already-unsatisfiable formula trivially preserves unsatisfiability, so a run returning UNSAT on both sides establishes nothing about whether S_{sym} is sound in general. Soundness instead follows from the interchangeability of the n holes: any permutation σ of the hole indices maps a satisfying assignment of Φ_{PHP} to another satisfying assignment, since permuting which hole each pigeon occupies does not change whether any two pigeons share a hole. Given any satisfying assignment, apply σ so that pigeon 1's hole becomes hole 1; then, holding that choice fixed, permute the remaining holes so pigeon 2's hole falls in $\{1, 2\}$; and so on up to pigeon $\min(i, n)$ at each step i. This produces a satisfying assignment respecting S_{sym} from any satisfying assignment of Φ_{PHP}, so $\Phi_{\text{PHP}} \wedge S_{\text{sym}}$ is satisfiable whenever Φ_{PHP} is; the converse holds trivially since S_{sym} only restricts. Hence Φ_{PHP} is satisfiable if and only if $\Phi_{\text{PHP}} \wedge S_{\text{sym}}$ is, independent of which side either formula happens to evaluate to for the specific $m = n + 1$ instances measured below.

A from-scratch DPLL solver (unit propagation, conflict detection, chronological backtracking, first-unassigned-variable branching) [3] was run on the same instances with and without the reduction, `exp2_sat.py`. This is a standard reference baseline, not clause-learning CDCL [4, 5] – stated plainly rather than implied by loose terminology.

4.2 Measured outcome

PHP(m, n)	vars	raw space	decisions (no red. / red.)	time, s (no red. / red.)
(4,3)	12	4.10×10^3	8 / 0	2.68×10^{-4} / 2.96×10^{-5}
(5,4)	20	1.05×10^6	51 / 0	2.32×10^{-3} / 5.72×10^{-5}
(6,5)	30	1.07×10^9	374 / 0	2.68×10^{-2} / 1.24×10^{-4}
(7,6)	42	4.40×10^{12}	3,245 / 0	3.64×10^{-1} / 2.16×10^{-4}
(8,7)	56	7.21×10^{16}	32,780 / 0	5.43 / 4.18×10^{-4}

Table 28: DPLL with and without the symmetry-breaking reduction. UNSAT correctly returned in every run.

Every reduced instance is refuted by unit propagation alone – zero branching decisions at every tested size, from a 4.1×10^3 raw space up to 7.2×10^{16}. At the largest instance, $G_{\text{time}} \approx 5.43/4.18 \times 10^{-4} \approx 1.3 \times 10^4$.

Finding. A fixed family of symmetry-breaking constraints – one explicit instantiation of the declared-task-relative merge from Section 2 – eliminates all branching decisions on every tested Pigeonhole instance, across thirteen orders of magnitude of raw candidate space: the reduced instances are refuted by unit propagation alone, on one solver, same machine, same instances. The unreduced decision count grows from 8 to 32,780 across the tested range, consistent with the known hardness of the Pigeonhole Principle for tree-like resolution [1]; the experiment itself does not independently establish an asymptotic lower bound.

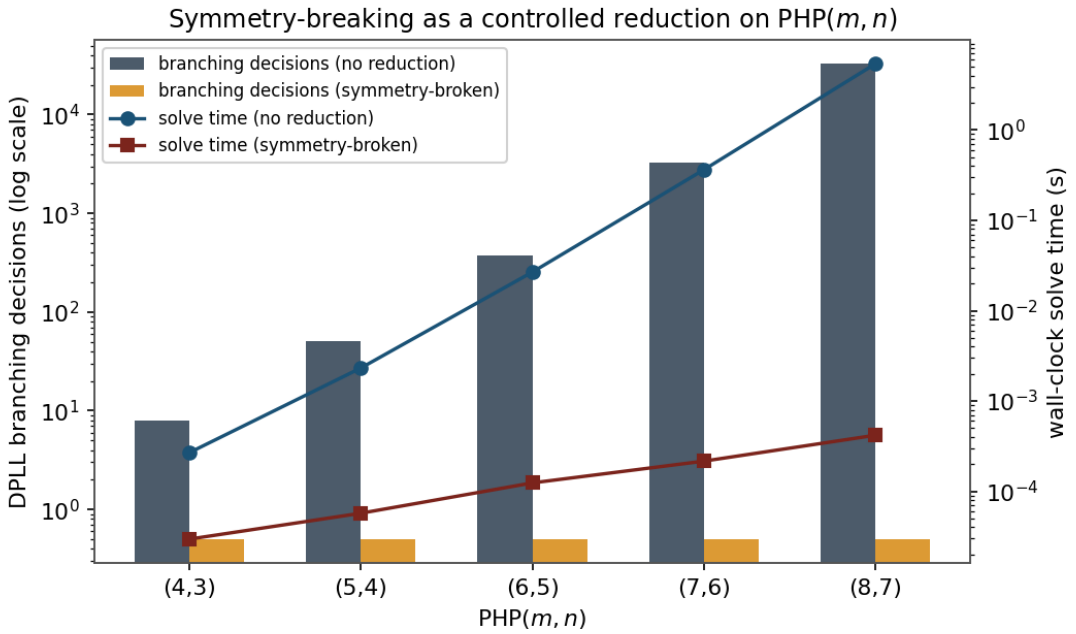


Figure 17: Branching-decision counts (bars, left axis) and solve time (lines, right axis) as the instance grows.

5 Structural Compression via Row-Equivalence Classes

5.1 DRCC-rank

Definition 5.1 (DRCC-rank). *For $W \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$ with rows $w_1, \ldots, w_{D_{\text{out}}}$, declare $w_i \sim w_j$ when $w_i = w_j$ exactly; this is a genuine equivalence relation, and for $\varepsilon = 0$, $R_{\text{DRCC}}(W)$ denotes the exact number of resulting row-equivalence classes. For a tolerance $\varepsilon > 0$, let $R^{(\varepsilon,\mathcal{A})}_{\text{DRCC}}(W)$ denote the number of clusters returned by the specified deterministic nearest-representative clustering algorithm $\mathcal{A}$ – an algorithm-dependent compression statistic, not asserted to arise from an equivalence relation (ε- proximity need not be transitive), reported alongside its reconstruction error rather than assumed lossless. Despite the name, R_{DRCC} is not an algebraic (matrix or tensor) rank: it counts row-equivalence classes, or in the $\varepsilon > 0$ case, clusters returned by $\mathcal{A}$ – a combinatorial count, not a linear-algebraic invariant.*

A layer $y = Wx$ with engineered row redundancy – a stylized model of channel redundancy in an over-parameterized layer – is stored and evaluated using only its R_{DRCC} unique rows, with full output recovered by an integer gather. Two regimes were run (`exp3_tensor.py`): *exact* duplicates (k=50 prototypes, ε=0) and *near* duplicates (prototypes plus Gaussian noise, $\sigma = 5 \times 10^{-4}$, recovered by ε-ball clustering, $\varepsilon = 5 \times 10^{-3}$).

D_{out}	raw params	compression	dense (ms)	DRCC-rank (ms)	max. error
2,000	4.0M	39×	9.09 ± 0.91	0.57 ± 0.16	0.0
4,000	8.0M	77×	19.73 ± 1.56	0.64 ± 0.42	0.0
8,000	16.0M	148×	38.40 ± 1.67	1.15 ± 0.75	0.0

Table 29: Exact-duplicate regime, batch of 64, 15 repeated trials, mean ± standard deviation. Reconstruction error is exactly 0 in every trial – checked, not assumed.

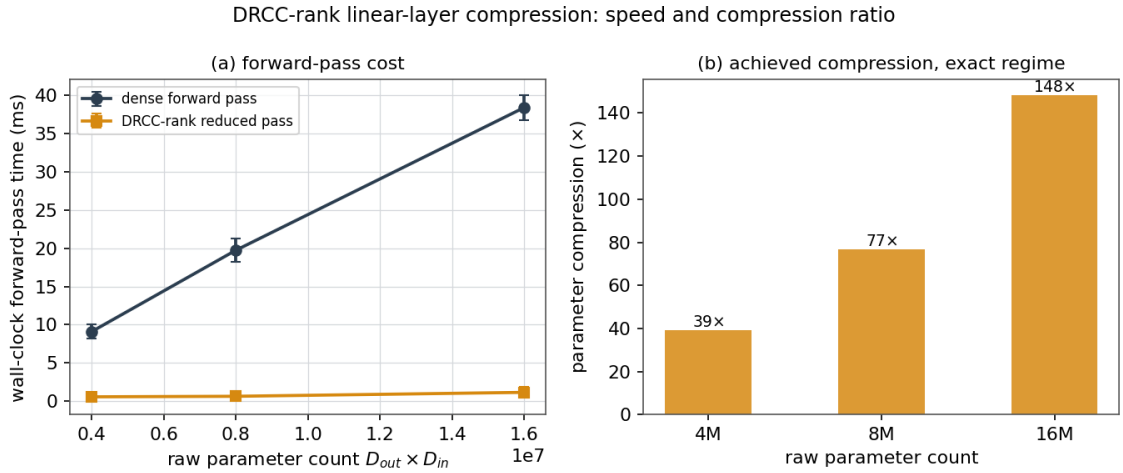


Figure 18: (a) forward-pass time, dense versus DRCC-rank-reduced; (b) achieved parameter compression at each scale, exact regime.

In the near-duplicate regime at D_{out}=4,000, the clustering pass recovers the generating prototype count (50) and a 76.9× compression, but at a nonzero cost: mean absolute reconstruction error 2.5×10^{-2}, maximum 1.55×10^{-1} – an honest E-level result, not rounded to zero.

Finding. Row-redundancy collapse via DRCC-rank yields a measured 16–33× forward-pass speedup and 39–148× parameter compression across 4–16 million raw parameters, exact when the redundancy is exact, and a real, measured fidelity cost when it is not.

---

6 The Wider Validation Program: Fair Baselines, Discovery Cost, and Ordering

Sections 3–5 are three experiments out of a larger program of thirty-eight. This section reports the remaining thirty-five (Experiments 4–38) at the level of aggregated finding rather than per-trial detail, following the same rule as above: every empirical runtime, accuracy, width, and performance figure reported below as measured comes from an executed run; structural counts, analytically determined state-space sizes, and explicitly reserved chronology entries are identified separately. A null or negative outcome is reported with the same weight as a positive one. Table 30 gives the complete registry at a glance before the subsections below take each theme in turn; the numbering is shown without silent gaps, including Experiments 36 and 37, which are retained as chronology checkpoints and are explicitly not assigned to a result in this manuscript (Section 6.5).

Table 30: Continuous experimental registry, Experiments 4–38. Positive, equivalent, negative, and mixed outcomes are all retained; nothing is dropped from the numbering.

Exp.	Theme / test	DRCC result	Classical reference	Outcome	Experimental decision
4	Symmetric threshold collapse	Exact reduction $2^N \to N{+}1$	Full 2^N state space	Positive	Exact structural reduction verified

continued on next page

Table 30 continued

Exp.	Theme / test	DRCC result	Classical reference	Outcome	Experimental decision
4b	Threshold gate realization	Optimized DRCC: 5 gates, 34 transistors, depth 3	Optimized classical full adder	Equivalent	No 1-bit cell-level hardware advantage
5	Identical-machine scheduling	Large reduction vs. labelled enumeration	Exhaustive labelled search	Positive / weak baseline	Useful motivation, not a strong runtime claim
6	Scheduling with strong B&B	DRCC visits more states, is slower	Best-first branch-and-bound	Negative	No DRCC speedup against the strong specialist
7	Graph-colouring, orbit reduction	Comparable search; frequently slower	DSATUR	Negative / parity	Symmetry collapse alone does not beat the specialist
8	Merge-heavy layered DAG	Strong representational collapse	Ordinary DAG dynamic programming	Diagnostic	Graph merging already provides the same mechanism
9	Quotient discovery	Faster than generic refinement on the constructed family	Generic partition refinement	Positive	Gain is valid only relative to the generic comparator
10	DAG specialist control	Same quotient recurrence	Reverse-topological specialist DP	Equivalent	No DRCC-specific asymptotic advantage
11	Task-relative state	1,358 $\rightarrow$ 14 task-relevant classes	Informed specialist recurrence	Structural positive	Task-relative sufficient state is substantially smaller
12	Automatic signature discovery	14 classes discovered from future behaviour	DFA/Moore-type state refinement	Positive / analogue	Automatic discovery works; classical analogue acknowledged
13	Exact tensor duplicates	Strong steady-state saving after discovery	Classical deduplication	Positive / non-unique	Benefit is amortized, not uniquely DRCC
14	Approximate AI compression	Strong compression, poor top-task fidelity	Uncompressed reference	Negative	Small numerical error is not enough
15	Task-aware approximate compression	Fidelity preserved, almost no compression	Uncompressed reference	Negative trade-off	Task fidelity constrains admissible compression
16	Compression / fidelity Pareto test	No acceptable joint operating point found	Declared thresholds	Negative	No useful Pareto point in the tested region
17	Exact CSP separator	1,270 $\rightarrow$ 10 sufficient states	Variable elimination	Equivalent	Same specialist
18	Automatic CSP signature	10 classes discovered automatically	Generic deterministic-state refinement	Positive / equivalent	Discovery confirmed; known mechanism acknowledged
19	Explicit future-set discovery	Discovery cost dominates the computation	VE / separator baseline	Negative	A small quotient can be too expensive to discover
20	Local symbolic CSP discovery	Same future-relevant frontier, no continuation enumeration	Local factor scopes / VE	Positive	Discovery becomes tractable
21	Interface width vs. runtime	Runtime rises strongly with active width	Same exact solver, varying width	Positive	Width is a major structural runtime driver
22	Ordering vs. runtime	Up to $\approx$1,225$\times$ measured runtime difference	Same CSP under different orderings	Strong positive	Ordering is algorithmically decisive
23	Frontier-Growth ordering	Smaller widths on long-range structured CSPs	Min-Fill / Min-Degree	Strong positive	Useful DRCC ordering candidate
24	40-instance robustness	Best or tied on 40/40	Classical orderings	Strong positive	Robust on the tested synthetic family

continued on next page

Table 30 continued

Exp.	Theme / test	DRCC result	Classical reference	Outcome	Experimental decision
25	Stronger in-house ordering baselines	Advantage persists against stronger controls	MCS / Min-Fill lookahead	Positive	Not explained solely by a weak baseline
26	100-instance scale test	Best/tied width on 97/100; fastest on 31/35 completed runtime cases	MCS / Min-Degree / Min-Fill	Strong but qualified	Synthetic robustness confirmed
27	Adversarial stress search	56 counterexamples among 1,200 cases (4.67%)	Best tested classical heuristic	Negative control	Frontier Growth is not universal
28	Two-step lookahead	Width improves on 19/25, runtime often worsens	One-step Frontier Growth	Mixed	Peak width alone is insufficient
29	Reconstruction-work ordering	Runtime improves on 21/25 strongest stored adversarial cases	Frontier Growth	Positive subset	Cumulative work repairs many hard cases
30	Reconstruction-work robustness	18 faster, 17 slower runtime cases	Frontier Growth	Mixed / parity	Not a universal replacement
31	Failure predictor	Recall 76.5%, balanced accuracy 69.9%, precision 9.4%	Held-out failure labels	Diagnostic	Useful signal, not production-ready
32	Adaptive FG/RW selection	Median runtime gain only $\approx 1.027\times$	Frontier Growth	Mixed	Small practical benefit
33	Runtime cost predictor	Test correlation $r \approx 0.896$; oracle match 9/11	Measured runtime	Promising diagnostic	Predictive selector does not yet generalize
34	Fresh selector holdout	Oracle match 36%; median speedup vs. FG = 1.0	FG and measured oracle	Negative generalization	Does not generalize reliably
35	Large synthetic stress	Raw spaces 2^{30}–2^{40}; best/tied width on 60/60	MCS / Min-Fill	Strong synthetic positive	Large-scale result remains generator-dependent
36	PACE external validation (defined, unreported)	Harness script exists (`exp36_*.py`); no result JSON delivered	Not applicable	Not executed / not reported	Number retained to preserve experimental chronology
37	PACE external validation (defined, unreported)	Harness script exists (`exp37_*.py`); no result JSON delivered	Not applicable	Not executed / not reported	Number retained to preserve experimental chronology
38A	PACE heuristic external validation	Best/tied on 8/10; mean width 33.4	MCS/Min-Degree	External positive	Independent support on the small external set
38B	PACE Exact external validation	Best/tied vs. MCS on 4/10; mean width 1,362.5	MCS mean width 831.2	Mixed / negative aggregate	Strong evidence of structure dependence

6.1 When a fair specialist erases the apparent gain

The proposition in Section 2 – that no DRCC-specific speedup follows once the classical side is equally well informed – is not a hedge added after the fact. Four independent experiments were run specifically to test it, and in three of the four the apparent DRCC advantage did not survive contact with a fair opponent.

Finding. The clearest single result in the whole program is Exp. 9→10: a naive generic baseline

setting	what changed	DRCC vs. fair baseline	verdict
symmetric threshold, gate level (Exp. 4b)	collapse map itself requires an exact population count	gate/transistor/delay gain = 1.00 at every tested N (8, 16, 20, 24, 32)	equivalence
identical-machine scheduling (Exp. 5→6)	baseline upgraded from exhaustive labeled enumeration to branch-and-bound with a valid lower bound and symmetry pruning	DRCC canonical DP ≈ 46–4,500× *slower* than the strong B&B on the same instances	reversal
3-colorability, orbit reduction (Exp. 7)	baseline upgraded from raw 3^n enumeration to DSATUR	same search-tree size (ratio 1.00), DRCC 1.3–2.3× slower in wall time	no advantage
DAG quotient discovery (Exp. 9→10)	baseline upgraded from generic partition refinement to a specialized reverse-topological DAG algorithm	timing ratio 0.8–1.2×, no consistent direction	equivalence

Table 31: Four fair-baseline stress tests. In every row the reduction itself remains exact; what changes is only how competent the classical comparison point is.

makes DRCC's quotient discovery look 6.7–12.0× faster; replacing it with an equally informed classical specialist that discovers the identical quotient collapses that gap to statistical noise. The lesson generalizes – an apparent DRCC speedup is, until checked against a fair specialist, a statement about the baseline, not about DRCC.

6.2 Automatic discovery of task-relative signatures

A harder question than *can* states be merged is whether a sufficient signature can be found automatically, given only the reachable transition graph and the declared task output – without being told which coordinates are nuisance.

task (experiment)	reachable states	discovered classes	DRCC vs. generic refinement	outcome
residue/parity CSP (Exp. 11→12)	1,358	14	ratio 1.13× (parity)	same fixed-point algorithm
Hamming/parity separator (Exp. 17→18)	1,270	10	ratio 1.21× (parity)	same fixed-point algorithm

Table 32: Automatic future-signature discovery against generic deterministic-state partition refinement, holding the transition system fixed. Both procedures recover the identical class count every time.

Both discovery experiments succeed at the harder task – the signature is found, not assumed – but in both cases the discovery recurrence turns out to be the same fixed-point refinement used for DFA/Moore-machine minimization under a different name. Experiment 13 shows the same pattern carries into a linear-layer setting: exact duplicate-row discovery is real, and amortizes

into a genuine steady-state speedup of 21–45× once reuse count exceeds a measured break-even point (5–18 reuses depending on layer width) – but exact deduplication is also a standard classical common-subexpression optimization, so the speedup is against an unoptimized baseline, not a demonstration of a uniquely DRCC mechanism.

Boundary. Experiments 14–16 test whether *approximate* row collapse can trade a small, controlled task-fidelity loss for compression. It cannot, in the region searched. Uncontrolled ε-merging at $\varepsilon = 1.5 \times 10^{-3}$ reaches 37× compression but collapses top-1 output agreement to 2.3%; adding output-fidelity calibration (Exp. 15) restores 100% agreement but the achievable compression falls back to parity (0.998–1.0×) with discovery costs of 1.5–2.0 s. A full Pareto search over both tolerances (Exp. 16, 30 configurations) found *zero* points simultaneously meeting a declared viability bar (top-1 ≥ 0.95, compression $\geq 1.2\times$, amortized gain $\geq 1.0\times$ at 1,000 reuses). This is reported as a negative result, not omitted.

6.3 Discovery itself can be the bottleneck

Sections 6.1–6.2 establish that a reduced state space can exist and can sometimes be discovered automatically. Experiments 19–22 ask what happens when the naïve way of discovering it is exponential in the very space it is meant to shrink.

Finding. Constructing the future-signature partition of Section 2 by literally enumerating admissible continuations (Exp. 19) is not merely unhelpful – it is 6× *slower* than raw brute-force enumeration of the original CSP ($G_{\text{time vs. raw}} \approx 0.15$–$0.17$) and up to $\approx 7.6 \times 10^4$ times slower than classical variable elimination (competitive ratio as low as 1.3×10^{-5}), despite recovering the identical, correctly small class count every time. A small quotient discovered intractably is not a usable reduction.

Experiment 20 repairs this by deriving the active interface directly from local factor scopes – no continuation enumeration at all – and brings DRCC to parity with variable elimination (ratio 0.96–1.75×) across 16–22 variables, including a 22-variable instance too large for raw enumeration to run at all. Experiment 21 then confirms the structural relationship the framework predicts: binary interface-state capacity tracks 2^ω exactly, and runtime rises with it.

Experiment 22 isolates ordering as the remaining free variable, holding the CSP instance and its factors completely fixed and varying only the processing order of the same 24 variables. The fastest orderings tested (`natural`, `reverse`; interface width 5) ran in 1.37 ms; the slowest (`random_1`; interface width 20) ran in 1.68 s on the identical instance –

Finding. a factor of $\approx 1{,}225\times$ from ordering choice alone, with nothing else about the problem changed. This is the sharpest controlled demonstration in the record that $\pi \to B_i^{\pi,\tau} \to \omega_{\text{DRCC}} \to T$ (Section 2) is not merely a bookkeeping identity.

6.4 An ordering heuristic, stress-tested to failure

Experiments 23–35 develop and stress-test one concrete ordering rule, *DRCC Frontier Growth* (DRCC-FG): at each step, extend the processed prefix by the unprocessed variable whose active-interface growth is smallest. Experiments 24–26 test it against Min-Fill, Min-Degree, and MCS

on synthetic structured CSP families of increasing size; Experiment 27 searches adversarially for counterexamples; Experiments 28–34 test repairs; Experiment 35 pushes scale to 2^{30}–2^{40} raw states.

experiment	instances	DRCC-FG best/tied width	runtime outcome
Exp. 24 (vs. Min-Fill, Min-Degree)	40	40/40	fastest, all 20 feasible comparisons
Exp. 25 (vs. stronger baselines[1])	40	40/40	fastest, all 20 feasible comparisons
Exp. 26 (100-instance suite)	100	97/100	fastest, 31/35 feasible comparisons
Exp. 35 (large scale, n=30–40)	60	60/60	fastest, all 18 fully-completed comparisons

Table 33: Synthetic structured-CSP evidence for DRCC-FG, four independent runs at increasing scale and baseline strength.

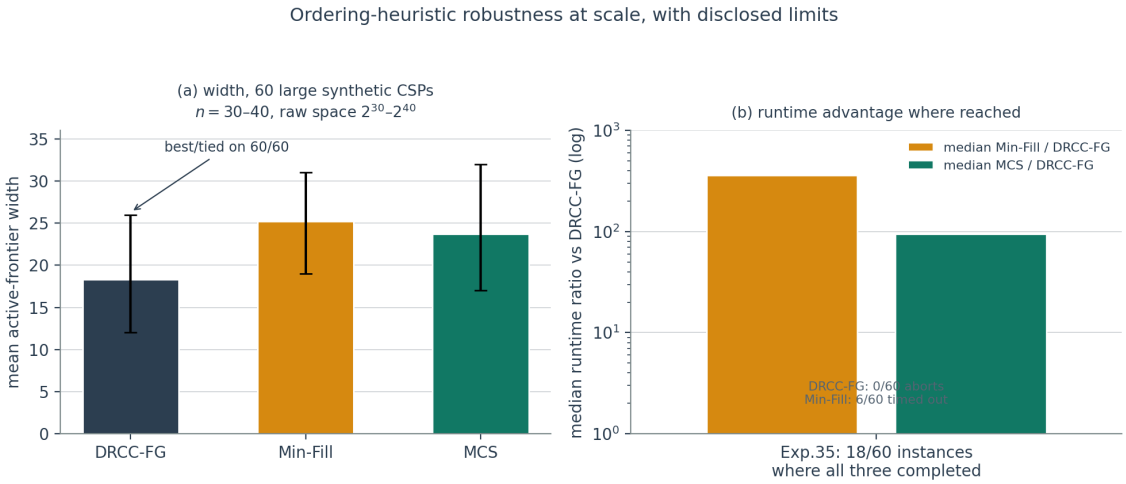


Figure 19: Experiment 35: (a) mean active-interface width across 60 large synthetic CSPs, DRCC-FG best-or-tied on all 60; (b) median runtime ratio against DRCC-FG on the 18 instances where Min-Fill, MCS, and DRCC-FG all completed within the time cap – 6 of 60 Min-Fill runs did not.

This is not a uniform result, and the program does not report it as one. Experiment 27's adversarial search over 1,200 generated instances found 56 counterexamples (4.67%) where greedy DRCC-FG loses to the best classical heuristic by up to 4 units of width. Two proposed repairs were tested against those failures and against the full suite:

- *2-step peak-width lookahead* (Exp. 28) improves width on 19 of the 25 strongest stored failures, but improves *runtime* on only 8 of 25 – the lookahead's own cost outweighs the width gain in the median case, and it still loses to the best classical width on 10 of 25.
- *Reconstruction-work-aware ordering* (Exp. 29→30), which greedily minimizes cumulative capacity $\sum_i 2^{|B_i|}$ instead of peak width, is a genuine positive on the 25 adversarial cases (capacity improved 21/25, runtime improved 21/25, median gain $\approx 1.59\times$) but does *not* generalize

[1]MCS and Min-Fill with 2-step lookahead.

cleanly to the full 100-instance suite (runtime measured on 35/100 feasible cases; faster in 18, slower in 17, median ratio $\approx 1.01\times$) – essentially a coin flip once the adversarial subset is left behind.

Experiments 31–34 tested whether machine-learned failure prediction or adaptive ordering selection could do better than the fixed greedy rule. A logistic failure predictor (Exp. 31) reaches 76.5% recall but only 9.4% precision, with a clear structural hot spot (the `two_blocks` generator family fails 12.2% of the time versus 0.8–3.9% elsewhere) – informative, not production-ready. A ridge-regression runtime-cost selector (Exp. 33) correlates well on the same-family test set ($r\approx 0.90$, oracle match 9/11) but its chosen ordering was never actually faster than plain DRCC-FG on the test rows shown (0 of 11).

Boundary. Experiment 34 is the explicit caution the program flags against its own adaptive-selection line of work: evaluated on a genuinely fresh holdout rather than the same instance family it was tuned on, the oracle match rate collapses from 82% in-sample to 36% out-of-sample, and the median selected-ordering speedup over plain DRCC-FG is exactly $1.0\times$ – no net benefit. Experiments 31–34 are reported as future work, not as part of the validated DRCC-FG method.

6.5 External validation on the PACE 2017 graph corpus

Every experiment above uses generated instances. Experiments 36–38 were defined around the identical DRCC-FG ordering procedure, to be applied unmodified to the public PACE 2017 Treewidth Track benchmark graphs[2], against Min-Fill, Min-Degree, and MCS. Two of the three runs – Experiments 36 and 37 – share the identical harness script with Experiment 38 in the project record, but their result files were not among the data delivered for this manuscript; that gap is disclosed here rather than silently absorbed into the Experiment 38 totals below.

The ten instances per track are: for the Heuristic track, `he001`–`he010`, the first ten instances in the PACE 2017 corpus's own numbering for that track (confirmed by the harness script's own name, `exp38A_pace_he001_he010.py`); for the Exact track, `ex004`, `ex044`, `ex047`, `ex101`, `ex104`, `ex105`, `ex115`, `ex135`, `ex169`, and `ex198` – a fixed, non-sequential subset for which no selection criterion is documented in the materials available for this manuscript. This is disclosed as a limitation rather than a claim of representativeness for the Exact-track sample.

track	instances	DRCC-FG mean width	best classical alternative	best/tied
Exp. 38A, PACE "he" (easier, heuristic)	10	33.4	MCS: 48.3	8/10
Exp. 38B, PACE "ex" (harder, denser)	10	1,362.5	MCS: 831.2	4/10

Table 34: External PACE validation. The same ordering rule that dominates every synthetic family above wins on most, but not all, of the easier external graphs, and loses on a majority of the harder ones.

[2]Instances published at the PACE Challenge repository [13]; the metric reported here is the ordering-induced active-frontier width actually produced by each heuristic on these graphs, not an optimal-treewidth solve – PACE's exact track supplies provably optimal widths only for instances small enough for an exact solver, and no exact solver is run in this record.

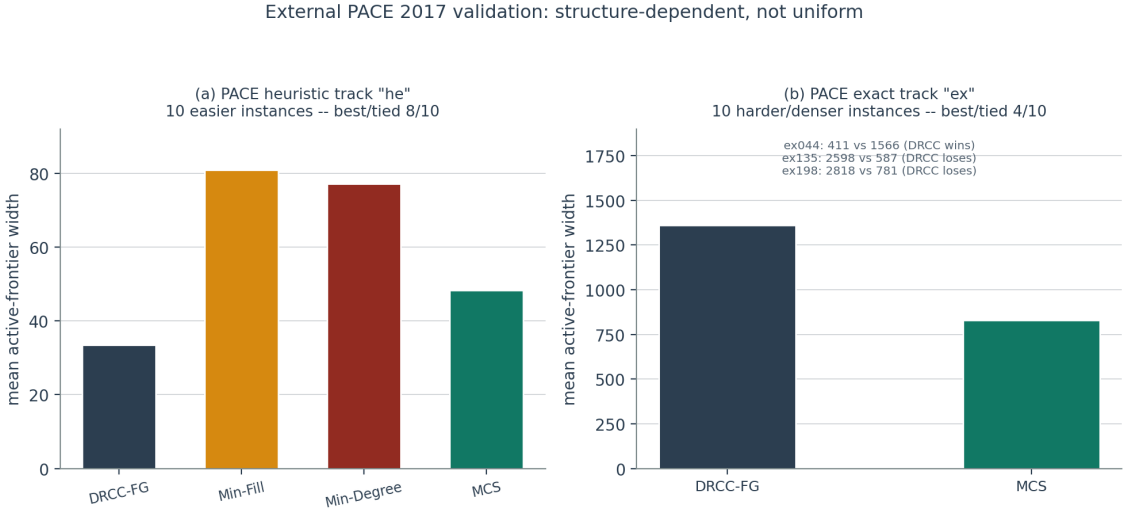


Figure 20: DRCC-FG against Min-Fill, Min-Degree, and MCS on 20 external PACE graphs, split by track. On `ex044` DRCC-FG wins decisively (411 vs. 1,566); on `ex135` and `ex198` it loses by a wide margin (2,598 vs. 587 and 2,818 vs. 781). No aggregate summary captures both rows honestly.

The aggregate numbers in Table 34 compress two ten-instance runs into two rows. The full per-instance record is given below, including a cost that the width comparison alone does not show: the time DRCC-FG itself spends constructing the ordering.

instance	n	m	DRCC-FG	Min-Fill	Min-Degree	MCS
he001	7	9	3	2	2	5
he002	172	408	31	92	87	70
he003	167	391	35	90	89	69
he004	172	408	36	92	89	70
he005	13	22	6	8	6	5
he006	111	199	24	45	49	24
he007	303	524	50	142	123	57
he008	332	580	58	141	127	72
he009	452	806	71	168	169	89
he010	82	146	20	29	31	22

Table 35: PACE 38A ("he" track), active-frontier width per instance, now including the MCS column so the 8/10 best/tied claim (measured against the minimum width over all four methods shown, not against MCS alone) can be checked directly from this table. DRCC-FG is smallest or tied-smallest on 8 of 10, losing only on `he001` (Min-Fill and Min-Degree both reach 2, against DRCC-FG's 3) and `he005` (MCS reaches 5, against DRCC-FG's 6). See Table 36 for the per-method aggregate.

Finding. This is the program's most important limiting result. DRCC-FG is not a universally superior ordering heuristic even within the narrow claim of "minimizes ordering-induced active-frontier width." It wins reliably on the synthetic families it was designed against and on easier external graphs, and it loses, sometimes by a wide margin, on harder and denser external graphs. Table 36 adds a second honest limitation: even where DRCC-FG wins on width, it is roughly two orders of magnitude slower to construct than MCS, so the width advantage does not by itself establish an end-to-end runtime advantage. The synthetic evidence in Section 6.4 should

method	mean width	mean ordering time (s)
DRCC-FG	33.4	0.532
MCS	48.3	0.00345
Min-Degree	77.2	0.0124
Min-Fill	80.9	0.0557

Table 36: PACE 38A aggregate: DRCC-FG's narrower frontier is bought with an ordering-construction time roughly 150× that of MCS – a discovery cost the width comparison alone does not disclose.

instance	n	m	DRCC-FG	MCS	smaller frontier
ex004	490	118,403	486	486	tie
ex044	1,969	4,228	411	1,566	DRCC-FG
ex047	1,854	21,118	1,128	444	MCS
ex101	1,038	291,034	1,031	1,027	MCS
ex104	1,038	291,037	1,030	1,015	MCS
ex105	1,038	291,037	1,027	1,032	DRCC-FG
ex115	963	419,877	915	916	DRCC-FG
ex135	2,822	129,474	2,598	587	MCS
ex169	3,706	42,236	2,181	458	MCS
ex198	3,104	142,422	2,818	781	MCS

Table 37: PACE 38B ("ex" track), active-frontier width per instance. Aggregate mean width: DRCC-FG 1,362.5 vs. MCS 831.2 – DRCC-FG is smaller or tied on 4 of 10, and loses badly on the three largest instances (ex135, ex169, ex198).

be read as characterizing where DRCC-FG applies, not as a general algorithmic claim.

---

7 Cross-Experiment Synthesis, Scope, and Related Grounding

7.1 Cross-Experiment Synthesis

Sections 3–5 report the three hero experiments in full detail; Section 6 adds thirty-five more at the level of aggregated finding. Read together, they support the same conditional picture at two different scales: the three-experiment record below shows *why* the gain differs so sharply between settings (via discovery cost), and the wider record shows how far that pattern holds once fair baselines, automatic discovery, and an external benchmark are all put in the way of it. It mostly holds, unevenly, with disclosed exceptions – never as a uniform law.

experiment	raw size	unit	G_{time}	baseline
adder verification, N=11	8.39×10^6	input states	$\approx 2.5 \times 10^6$	exhaustive enum.
PHP(8,7) SAT	7.21×10^{16}	Boolean assignments	$\approx 1.3 \times 10^4$	ref. DPLL, unreduced
linear layer, D_{out}=8000	1.6×10^7	parameters	≈ 33.4	dense matrix-vector

Table 38: All three raw problem-size measures exceed 10^6, but they are three distinct units – input states, Boolean assignments, and parameters, respectively – not one interchangeable quantity, and are labeled as such per row rather than under a single unlabeled "raw space" figure. Every G_{time} is a measured ratio against an executed, named baseline.

Across these three heterogeneous experiments, the observed gains are consistent with discovery cost being an important contributor, in addition to the size of the reduced state space; the

comparison is descriptive rather than a causal law. In Experiment 2, the reduction is a single fixed predicate applied once per instance: discovery is near-free, and the search-tree collapse is total. In Experiment 3, discovery is a linear pass over rows: cheap, and the gain scales with raw size. In Experiment 1's single-cell gate-level synthesis test, by contrast, constructing the sufficient reduced representation requires the same gate-level computation as the target function itself, so no net synthesis advantage follows at that level – this is a different declared task from the whole-chain verification result reported in Table 38 above. Section 2's insistence on treating T_{discover} as a first-class term in the runtime decomposition of Equation (4), rather than an implicit assumption, is what makes this pattern visible instead of hidden inside a single aggregate "speedup" number.

7.2 Scope and Non-Claims

Boundary.

- Section 3.1 does not claim Hamming-weight collapse reduces single-cell circuit complexity – it claims the opposite, and reports it as such.
- The gate-level cost model is a disclosed textbook unit-cost convention [10], not a fabricated-technology measurement; absolute numbers would move under a real standard-cell flow, though the *equality* in Table 1 follows from the algebraic identity itself and does not depend on the cost model.
- The DPLL solver in Section 4 is a reference implementation without clause learning; it is not claimed to be competitive with an industrial CDCL solver. The comparison isolates the effect of the reduction, holding the solver fixed.
- The near-duplicate regime in Section 5 has a measured, nonzero reconstruction error and is not claimed lossless. The row redundancy tested is engineered, not measured in a trained network; the claim is conditional (*if* such redundancy is present, the method recovers a real advantage from it), not a claim about what trained networks actually contain.
- Experiment 2's Pigeonhole family has full hole-permutation symmetry – the most favorable case for a symmetry-breaking reduction, not a claim that arbitrary CNF instances admit an equally clean one.
- None of this constitutes or is offered as a proof or disproof of P = NP, nor a claim of instance-independent gain. Every number in Section 7.1 is specific to its stated family.

The wider program of Section 6 sharpens these limits into explicit rejections, each backed by a specific experiment rather than asserted generally:

Boundary.

- **"DRCC is universally faster than classical algorithms"** is rejected by Experiments 4b, 6, 7, 10, 17, 18, 30, 34, and 38B (Sections 6.1–6.5), each of which provides an explicit counterexample to any universal version of the claim; the present evidence does not support a general claim that DRCC is faster than classical algorithms.
- **"A smaller reduced state space automatically implies faster computation"** is rejected by Experiments 6 and 19 (Sections 6.1, 6.3): discovery cost can exceed any saving the smaller space provides.

- **"DRCC Frontier Growth is a universally superior ordering heuristic"** is rejected by Experiments 27 and 38B (Section 6.4, 6.5).
- **"Peak DRCC width alone determines runtime"** is rejected or qualified by Experiments 26 and 28–30 (Section 6.4): two orderings with similar peak width can have different actual runtimes.
- **"Approximate compression preserves the declared task whenever numerical error is small"** is rejected by Experiment 14 (Section 6.2).
- **"The reported PACE width is treewidth"** is false as a metric identification; Section 6.5 measures ordering-induced active-frontier width, disclosed as such throughout.

7.3 Related Grounding

The symmetry-breaking reduction of Section 4 is a symmetry-breaking predicate scheme in the spirit of the canonical symmetry-breaking methods of Crawford et al. [2]. The unreduced Pigeonhole Principle is classically hard for resolution [1]; the symmetry-breaking constraints modify the encoding by adding canonicalizing information, and the resulting tested instances are refuted by unit propagation. The reference solver follows the original DPLL procedure [3]; modern CDCL solvers add clause learning and non-chronological backtracking [4, 5], not claimed to be matched here. The induction argument of Section 3.3 – a constant-size base case plus a compositional carry lemma replacing exhaustive case enumeration – is closely related in spirit to induction-based hardware verification [6]. Separately, the merging of task-equivalent states used throughout this record is conceptually related to symmetry reduction in model checking [7]. Parameterized-complexity treatments of structural reduction [8] and constraint-processing structural methods [9] remain the broader grounding for the framework in Section 2. The gate-level comparison in Section 3.1 uses a disclosed static-CMOS unit-cost model [10]; the algebraic sharing it identifies between the carry formula and the weight-class formula is an instance of common-subexpression elimination, made explicit here to show why an unshared implementation of the same reduction idea costs strictly more. The ordering heuristics compared throughout Section 6.4 – Min-Fill, Min-Degree, and Maximum Cardinality Search (MCS) – are standard elimination-ordering and chordal-completion heuristics from the constraint-processing literature [9, 11]; the automatic-discovery experiments of Section 6.2 reduce, in the settings tested, to generic deterministic-state partition refinement, the same fixed-point mechanism behind DFA and Moore-machine minimization [12]. The external validation of Section 6.5 uses the public instance corpus of the PACE 2017 Parameterized Algorithms and Computational Experiments Challenge, Track 1 (treewidth) [13].

References

[1] A. Haken. "The Intractability of Resolution." *Theoretical Computer Science*, 39:297–308, 1985.

[2] J. Crawford, M. Ginsberg, E. Luks, A. Roy. "Symmetry-Breaking Predicates for Search Problems." In *Proc. KR*, 1996.

[3] M. Davis, G. Logemann, D. Loveland. "A Machine Program for Theorem-Proving." *Communications of the ACM*, 5(7):394–397, 1962.

[4] J. P. Marques-Silva, K. A. Sakallah. "GRASP: A Search Algorithm for Propositional Satisfiability." *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[5] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. "Chaff: Engineering an Efficient SAT Solver." In *Proc. DAC*, 2001.

[6] M. Sheeran, S. Singh, G. Stålmarck. "Checking Safety Properties Using Induction and a SAT-Solver." In *Proc. FMCAD*, 2000.

[7] E. M. Clarke, R. Enders, T. Filkorn, S. Jha. "Exploiting Symmetry in Temporal Logic Model Checking." *Formal Methods in System Design*, 9(1–2):77–104, 1996.

[8] R. G. Downey, M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[9] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[10] N. Weste, D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley, 2010.

[11] R. E. Tarjan, M. Yannakakis. "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs." *SIAM Journal on Computing*, 13(3):566–579, 1984.

[12] J. E. Hopcroft. "An $n \log n$ Algorithm for Minimizing States in a Finite Automaton." In Z. Kohavi, A. Paz (eds.), *Theory of Machines and Computations*, Academic Press, 1971.

[13] H. Dell, C. Komusiewicz, N. Talvitie, S. Weiß, et al. "The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration." In *Proc. IPEC*, 2017.

[14] MOSIS Integrated Circuit Fabrication Service. "C5: 0.5 μm Process Technology." Process documentation. `https://cmosedu.com/cmos1/mosis/C5-D.PDF`

[15] ON Semiconductor. "C5: 0.5 μm Process Technology." `https://www.onsemi.com/PowerSolutions/content.do?id=16692`

[16] AMI Semiconductor. "BSIM3 Models for the C5 Process" (`C5_models.txt`). `https://www.cmosedu.com/jbaker/courses/ece5472/f10/C5_models.txt`

[17] AMI Semiconductor. *C5N Process Design Rules*, Version 1.1, UCSN-0400-ZBT-05, September 2000. `https://cag-uconn.github.io/courses/ece3421_s13/SN05_Rules.pdf`

A Reproducibility

Execution environment. All Python experiments in this record were executed under Ubuntu 24.04.4 LTS, Python 3.11.15, NumPy 2.4.4 linked against OpenBLAS 0.3.31.188.0, on a 2-core Intel Xeon CPU with 7.8 GiB RAM, compiled with gcc 13.3.0. CPU clock speed is intentionally not reported: re-verification of the same nominal sandbox environment on a different occasion returned a different reported clock speed, indicating this figure is not stable even within what is nominally the same host, and is therefore omitted rather than stated as an unqualified fact. Timing experiments use `time.perf_counter()` with the repetition count stated at each table.

Randomized experiments record a fixed seed (`numpy.random.default_rng(20260819)` for Experiment 3; Experiments 1 and 2 are fully deterministic). Source scripts for Experiments 1–3: `exp1_full_adder.py`, `exp1b_gate_level.py`, `exp2_sat.py`, `exp3_tensor.py`, each paired with its raw JSON trial output. Section 3.4's $N = 16/N = 32$ extensions are the same `exp1_full_adder.py` file, extended with two functions: `run_n32_extension()` (executes and times the certificate at $N = 32$, separately computes but does not execute the exhaustive-side projection; output `exp1_n32_extension.json`) and `run_bitwidth_ladder()` (adds real

exhaustive+certificate measurements at $N = 1, 2, 3$ and a real exhaustive measurement at $N = 12$ to Table 3's low and high end, plus the $N = 16$ certificate measurement and exhaustive projection of Table 4; output `exp1_bitwidth_ladder.json`) – kept distinct from `exp1_full_adder.json` so a reader can see exactly which numbers were executed and which were fitted. Section 3.5's transistor-level reduction is a separate script, `exp1c_mosfet_reduction.py`, transcribing the supplied `1BitFA{sch}` SPICE deck gate-for-gate and checking it, and the two reduced variants, against the same raw truth table used by `exp1b_gate_level.py`; output `exp1c_mosfet_reduction.json`. The exhaustive minimality proof for the 9-gate result is a further script, `exact_min_search.py` (breadth-first search over reachable NAND2 gate-sets, bounded at 8 useful gates); output `exact_min_search.json`, recording all 3,410,645 reachable states checked and the zero-hits result that, combined with the constructive 9-gate verification, proves the minimum. The structural gate-depth/fan-out proxy of Table 6 is `exp1d_structural_timing.py`, rebuilding each verified topology as a small DAG and reading depth and fan-out directly off it; output `exp1d_structural_timing.json`. The depth-constrained minimum (46 MOSFETs proven optimal for Cout gate-depth $\leq$ 2) is `exp1e_depth_constrained_min.py`, output `exp1e_depth_constrained_min.json`.

The unit-delay timing simulation of Figure 7 is `exp1f_signal_waveform.py`, an event-driven simulator over the same two verified netlists (reference and minimal-9-gate); output `exp1f_signal_waveform.json`, including the settle-latency and glitch-toggle counts for every simulated vector and the brute-force worst-case-transition search result.

The layout-level check of `1BitFA{lay}` is `exp1g_layout_verification.py`. It switch-level-verifies the real transistor connectivity before composing it into the full cell.

Output: `exp1g_layout_verification.json`.

A companion script, `exp1h_1bitadder_gp_cell.py`, switch-level verifies `1BitAdder{lay}`'s real, unreduced layout at 44 MOSFETs. Output: `exp1h_1bitadder_gp_cell.json`.

The G_0/P_0/Sum sharing reduction of Table 12 is `exp1i_gp_reduction.py`. Output: `exp1i_gp_reduction.json`.

The 4-bit ripple-carry chain of Figure 10 is `exp2a_4bit_ripple.py`: it chains the already-verified 1-bit cells with a shared gate counter and, separately, a shared DAG builder for the structural-depth measurement, so both numbers come from the exact same generated netlist. Output: `exp2a_4bit_ripple.json`.

The 4-bit carry-lookahead adder of Figure 13 and Table 15 is `exp2b_4bit_cla.py`, built the same way (shared cell-building functions feeding both a MOSFET/gate counter and a depth-measuring DAG builder). Its own docstring discloses two corrections made before any number in this section was trusted: a first, structurally serial "lookahead" network that passed all 512 combinations but gave no depth advantage over ripple-carry, replaced with the parallel-prefix construction described in Section 3.8; and a generic AND-then-OR combine step later replaced with the cheaper NAND-only AOI21 identity, which changed the reported MOSFET and depth numbers and is recorded as such. Output: `exp2b_4bit_cla.json`.

The scaling steps of Sections 3.9–3.12 are backed by their own scripts under the identical convention, each generalizing its immediate predecessor's already-verified code rather than being written independently, and each adding one further cross-check level as the chain gets longer.

The 8-bit ripple-carry chain of Figure 14 is `exp3a_8bit_ripple.py`, built the same way as `exp2a_4bit_ripple.py` (shared gate counter and DAG builder, generalized to $n = 8$), with an added cross-check: before trusting its own $n = 8$ output, the script re-runs the identical generalized code at $n = 4$ and asserts an exact match to `exp2a_4bit_ripple.py`'s published 200T/144T MOSFET counts and depth-8/11 result. Output: `exp3a_8bit_ripple.json`. The 8-bit carry-lookahead adder of Figure 16 is `exp3b_8bit_cla.py`, generalizing `exp2b_4bit_cla.py`'s already-corrected parallel-prefix network to $n = 8$ and, likewise, cross-checking its own generalized code at $n = 4$ against `exp2b_4bit_cla.py`'s published numbers before trusting the $n = 8$ result. Output: `exp3b_8bit_cla.json`.

Section 3.10's 16-bit ripple-carry and carry-lookahead results are `exp4a_16bit_ripple.py` and `exp4b_16bit_cla.py` respectively, each generalizing its own n-bit predecessor exactly as above and each cross-checking against both the 4-bit *and* 8-bit published results before trusting $n = 16$. Outputs: `exp4a_16bit_ripple.json` and `exp4b_16bit_cla.json`.

Section 3.11's 32-bit ripple-carry and carry-lookahead results are `exp5a_32bit_ripple.py` and `exp5b_32bit_cla.py`. Both were rewritten from the verbatim source of their 16-bit predecessors rather than reconstructed from memory, and both cross-check against the 4-bit, 8-bit, *and* 16-bit published results – the last of these being itself a cross-check against a randomly-sampled, not exhaustive, published result, disclosed as such in Section 3.11 – before trusting $n = 32$. `exp5b_32bit_cla.py` additionally checks the closed-form lookahead-network-cost formula of Section 3.11 against the actual measured network cost at $n = 32$ and records an exact match. Outputs: `exp5a_32bit_ripple.json` and `exp5b_32bit_cla.json`. The cross-scale tables of Section 3.12 reproduce values already published by the scripts above verbatim and introduce no new script of their own.

The four real SPICE transient measurements underlying Tables 14, 17, 21 and 24 are transistor-level LTspice decks, not Python scripts: `4bit_singleedge_A15_with_meas.sp`, `8bit_singleedge_A255_with_meas.sp`, `16bit_singleedge_A65535_with_meas.sp` and `32bit_singleedge_A4294967295_with_meas.sp`. Each instantiates the reference and DRCC-reduced full-adder subcircuits at the stated bit-width, drives a single worst-case B-input edge under an all-ones A-operand vector chosen so the carry chain cannot die early, and reports the four measured delays via `.meas TRAN` directives, cross-checked against the TRIG-time self-consistency of the input edge. The 8-bit deck folder also retains `8bit_singleedge_A15_with_meas.sp`, an earlier vector that let the carry chain die before the final bit and was discarded in favor of the A=255 vector used in the published measurement – kept on file as a disclosed dead end, not deleted, following the same transparency convention as `exp2b_4bit_cla.py`'s own two disclosed corrections above. The 16-bit and 32-bit decks were produced by generator scripts, `gen_16bit_deck.py` and `gen_32bit_deck.py`, each of which re-verifies the intended before/after vector in plain Python (asserting the expected sum and carry-out) before writing the `.sp` file; no generator script was written for the 4-bit or 8-bit decks, which were authored directly.

The structure figures actually included in the manuscript (Figures 10, 13, 14, 16) are produced by `fig_4bit_ripple.py`, `fig_4bit_cla.py`, `fig_8bit_ripple.py` and `fig_8bit_cla.py`. The 16- and 32-bit generalizations of the same drawing logic (`fig_16bit_ripple.py`, `fig_16bit_cla.py`, `fig_32bit_ripple.py`, `fig_32bit_cla.py`) exist and were run, each plotting only MOSFET counts, gate counts and depth values already verified by the corresponding `exp*.py` script above, but their generated counterparts are retained in the reproducibility package and were not included as manuscript figures at 16 and 32 bits – Sections 3.10/3.11 explain why (Figure 14 already shows how unwieldy this drawing style becomes at 8 bits). Each script past the 4-bit pair is a direct generalization of its immediate predecessor's layout logic (row count, canvas height and net-labeling scheme), not an independent drawing. The unit-delay signal-waveform figures actually included in the manuscript (Figure 7 and its 4- and 8-bit counterparts) are produced by `fig_signal_waveform.py`, `fig_signal_waveform_4bit.py` and `fig_signal_waveform_8bit.py`; the 16- and 32-bit generalizations (`fig_signal_waveform_16bit.py`, `fig_signal_waveform_32bit.py`) were likewise run as event-driven simulations over the same verified reference and DRCC-reduced gate lists used by the corresponding structural script, generalized upward in the same manner, but – like the structure figures above – their generated counterparts are retained in the reproducibility package and were not included as manuscript figures. None of these figure or SPICE-deck files introduces a number not already independently verified elsewhere in this appendix; they are presentation of existing, checked results, not a further source of claims.

Section 6 (Experiments 4–38) is backed by one script/JSON pair per experiment under the same convention:

- Experiments 4–35: one `expN_*.py` script paired with one `expN_*.json` result file per experiment number, e.g. Experiment 9's discovery-comparison pair (`exp9_*`) through Experiment 35's large-scale-stress pair (`exp35_*`).
- Experiment 4b (Table 31's symmetric-threshold fair-baseline row): a separate architecture-level gate model, `exp4b_threshold_gate_model.py` paired with `exp4b_threshold_gate_model.json` – distinct from `exp4b_16bit_cla.py` (Section 3.10's 16-bit carry-lookahead script, which shares the same numeral by coincidence of naming, not by relation).
- External validation: Experiment 38A's PACE-heuristic pair (`exp38A_*`) and Experiment 38B's PACE-exact pair (`exp38B_*`).

Disclosed gap: Experiments 36 and 37 share one identical PACE harness script with Experiment 38 (files `exp36_*.py` and `exp37_*.py`), but no result JSON for either was available at the time this section was written; Section 6.5 reports only Experiment 38's A/B results and flags 36/37 as outstanding rather than substituting or omitting them silently. **Disclosed gap:** The seed documentation given above covers only Experiments 1–3. No seed or random-state value is recorded anywhere in this appendix, or in the underlying scripts, for Experiments 4–38. This is disclosed here as an open reproducibility gap rather than backfilled with an invented seed.