ESE3700 Project 1

Lucas Krippendorff

Contents

1 Boolean Functions and Design Approach

To design an adder, it is natural to start with the simplest case: adding two single bits. The possible combinations are:

$$0+0=0, \qquad 0+1=1+0=1, \qquad 1+1=10$$

When $A = B = 1$, the result is 2, which in binary is a two-digit number: the first digit (the sum bit) is 0, and the second digit (the carry) is 1. These two outputs are labeled S (Sum) and C_{out} (Carry-out).

To add numbers wider than one bit, each bit position must also account for the carry produced by the bit position below it. For example, $01+01=10$: the upper bit, despite being $0+0$, becomes 1 because the lower bit produced $1+1=0$ with a carry of 1 into the next position. Each bit-slice therefore has three inputs – A, B, and C_{in}, where C_{in} is the C_{out} of the previous bit-slice – and two outputs, S and C_{out}. This is the full adder.

The Boolean expressions for S and C_{out} as functions of A, B, and C_{in} are most easily read off from Karnaugh maps, shown in Figure 1.

C_{in} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

No groupings exist; the alternating checkerboard pattern corresponds to parity (XOR).

(a) $S = A \oplus B \oplus C_{in}$

C_{in} \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Three prime implicants: AB (col 11, both rows), BC_{in} (row 1, cols 01–11), AC_{in} (row 1, cols 11–10).

(b) $C_{out} = AB + AC_{in} + BC_{in}$

Figure 1: Karnaugh maps for Sum and Carry-out. C_{in} varies across rows; AB varies across columns in Gray-code order (00, 01, 11, 10).

Reading off the maps directly yields:

$$S = A \oplus B \oplus C_{in} \tag{1}$$

$$C_{out} = AB + AC_{in} + BC_{in} \tag{2}$$

These two expressions are the starting point. The remainder of this section explores how each can be implemented in CMOS, why several natural choices are ultimately rejected, and how recognizing a shared sub-expression leads to the chosen topology. Sections 1.1 and 1.2 together

constitute the design space exploration, investigating options S1–S4 for Sum and C1–C3 for Carry, plus an all-NAND implementation, before arriving at the selected design.

1.1 Implementations of Sum

Throughout this section, transistor count is used as the primary metric for comparing implementations. A rigorous delay estimate would use the Elmore delay model augmented with diffusion capacitance, but that analysis is implementation-specific and requires transistor sizing – a degree of freedom not yet explored. Transistor count is fast to determine directly from the Boolean expression and serves as a reasonable filter: when one implementation uses two or three times as many transistors as another, the difference in capacitance loading is large enough that no amount of sizing is likely to close the gap. The more careful Elmore analysis is deferred to Section 2.2.

Equation 1 is a three-input XOR, which admits several gate-level readings.

Option S1 — Single XOR3 gate. The most direct reading of Equation 1 is a single 3-input XOR gate. A transmission-gate XOR3 can be realized compactly, but its output is not signal-restored: the output voltage is degraded by a threshold drop across the pass transistors, making it unreliable as a logic input to the next stage without an additional buffer. A static complementary CMOS XOR3 avoids this but requires a deeply stacked pull-down network that is inherently more complex than decomposing the function into smaller, well-understood primitives. Either way, a three-input XOR gate is a bespoke cell that offers no structural reuse with the carry path. *Rejected: non-restored output (TG form) or unnecessary gate complexity (static form), with no sharing with* C_{out}.

Option S2 — Two cascaded XOR2 gates. Because XOR is associative,

$$S = (A \oplus B) \oplus C_{in} = A \oplus (B \oplus C_{in}) = (A \oplus C_{in}) \oplus B \quad \text{(all equivalent)}.$$

Any grouping produces two identical XOR2 stages. A transmission-gate XOR2 produces a fully restored output and is a standard, well-characterized cell. The grouping $(A \oplus B)$ first is preferred because, as shown in Section 1.2, that intermediate is also needed by C_{out}, so no extra hardware is incurred.

Option S3 — NAND-only XOR2 chain. NAND gates are the most natural primitive in complementary CMOS because they require no output inverter – the pull-down network directly produces an inverted result. It is worth asking whether $A \oplus B$ can be expressed purely in terms of NAND2 gates before committing to transmission gates.

The key identity is that $\overline{AB}$ lets us extract $A\bar{B}$ and $\bar{A}B$ without an explicit NOT gate. Starting from the canonical XOR expansion:

$$A \oplus B = A\bar{B} + \bar{A}B$$

Noting that $A \cdot \overline{AB} = A(\bar{A} + \bar{B}) = A\bar{B}$ and $B \cdot \overline{AB} = \bar{A}B$, both terms can be rewritten using the single NAND output $\overline{AB}$:

$$A \oplus B = A\bar{B} + \bar{A}B = A \cdot \overline{AB} + B \cdot \overline{AB}$$

Applying De Morgan to the OR gives the all-NAND form:

$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

This is four NAND2 gates, each costing 4 transistors: **16 T per XOR2 stage**. Computing S requires two such stages – one for $P = A \oplus B$ and one for $S = P \oplus C_{in}$ – giving **32 T** for Sum alone, before counting the carry logic.

Among static gate implementations, the all-NAND construction is the most transistor-efficient way to build an XOR2. Alternative approaches using NOR gates introduce redundant inverters – for example, factoring $A \oplus B = \overline{AB} \cdot (A+B)$ and realizing $(A+B)$ via a NOR followed by an inverter requires 14 T but mixes gate types without a meaningful transistor saving, and the added NOR pull-up delay offsets any reduction in count. NAND gates avoid this because the $\overline{AB}$ intermediate is already the correct polarity for the subsequent stages, requiring no extra inversion.

Option S4 — Transmission-gate XOR2. A fundamentally different approach uses transmission gates (TGs) rather than static pull-up/pull-down networks. A TG is a parallel NMOS/PMOS pair: when its control signals are asserted it passes its input to its output with near-ideal conductance; when de-asserted it is open-circuit. XOR can be read as a 2-to-1 multiplexer controlled by B: when $B = 0$, the output should equal A; when $B = 1$, the output should equal $\bar{A}$. With B and $\bar{B}$ controlling two TGs:

TG_1 (B=0): pass A	2 T
TG_2 (B=1): pass $\bar{A}$	2 T
INV(B): generate $\bar{B}$	2 T
INV(A): generate $\bar{A}$	2 T

Total: **8 T per XOR2 stage**, or **16 T for Sum** – half the count of S4 and a quarter of S3.

The tradeoff is that a TG does not restore the signal: it passes a voltage, not a logic level reinforced by a pull-up or pull-down network. In a chain of two cascaded TG-XOR2 stages, the degraded output of the first stage feeds directly into the second, potentially compounding the signal integrity issue. This concern motivates reserving S4 as an optimization candidate to be validated by SPICE, rather than adopting it immediately as the baseline.

Interim selection: The transistor counts for Sum across the options are: S1 (signal integrity issues, rejected), S3 (32 T), and S4 (16 T). On transistor count alone S4 wins, but the signal restoration concern across two cascaded TG stages is a legitimate risk for a baseline design. Option S3 (four-NAND2 XOR2, 32 T) is therefore selected as the **baseline** sum implementation – fully restored outputs at every node, straightforward to analyse. Option S4 is carried forward as a promising optimization candidate and evaluated in Section 3.

1.2 Implementations of C_{out}

Starting from the expression in Equation 2, a sequence of algebraic steps reveals progressively better implementations.

Option C1 — AND/OR/INV gates, direct from the K-map. The K-map yields three prime implicants (AB, AC_{in}, BC_{in}), each implementable as an AND2 gate, whose outputs are OR'd together. Since OR is not a primitive in complementary CMOS (it requires an internal inversion), this maps to three AND2 gates feeding an OR3, followed by no additional inverter since OR is realized as NOR + INV. No intermediate signal is shared with the sum path, and the gate count is high. *Rejected: three levels of logic, no sharing with Sum.*

Option C2 — Factor out C_{in}.

$$\begin{aligned} C_{out} &= AB + AC_{in} + BC_{in} \\ &= AB + C_{in}(A + B) \end{aligned} \tag{3}$$

C1 had two stages: the three AND gates compute in parallel, then their results feed a single OR3. C2 at face value looks worse – the OR, AND, and final OR must execute serially, giving three stages rather than two. However, in a full adder A and B are available immediately, while C_{in} is delayed, arriving only after the previous bit-slice has resolved its carry. $(A+B)$ is therefore already computed and waiting long before C_{in} arrives – it is $(A + B)$ waiting on C_{in}, not the other way around. The effective delay once C_{in} arrives is just the AND and OR it passes through, making this implementation two stages in practice, the same as C1, with the added benefit that the C_{in} path is now explicitly isolated as a single branch. *Carried forward: the factored structure directly motivates the next step.*

Option C3 — Replace $(A + B)$ with $(A \oplus B)$. The substitution $A + B \to A \oplus B$ is valid here because the only case where $A + B \neq A \oplus B$ is $A = B = 1$, but in that case $AB = 1$ already forces $C_{out} = 1$ regardless of C_{in}, making the $C_{in}(A + B)$ term redundant. Formally:

$$\begin{aligned} C_{out} &= AB + C_{in}(A + B) \\ &= AB + C_{in}(A \oplus B) + C_{in} \cdot AB \\ &= AB(1 + C_{in}) + C_{in}(A \oplus B) \\ &= AB + C_{in}(A \oplus B) \end{aligned} \tag{4}$$

$$\boxed{C_{out} = AB + C_{in}(A \oplus B)}$$

Notably, $A \oplus B$ is the same intermediate already computed by the first XOR2 stage in Option S2. The significance of this is explored in Section 1.3.

The OR between AB and $C_{in}(A \oplus B)$ can be converted to a NAND using De Morgan's law:

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{AB} \cdot \overline{C_{in}(A \oplus B)}} = \overline{\overline{AB} \cdot \overline{P \cdot C_{in}}}$$

This is three NAND2 gates: $\overline{AB}$, $\overline{P \cdot C_{in}}$, and their NAND. As shown in Table 4, NAND pull-up (143 fs) is at least 2× faster than NOR pull-up (288 fs) in the worst case, making this the preferred carry implementation.

1.3 Selected Implementation

The carry implementation is settled: C3 with three NAND2 gates avoids all series-PMOS paths and reuses $P = A \oplus B$ from the sum path.

For the XOR2 itself, the analysis in Sections 1.1 pointed to two viable candidates. Option S4 (TG-XOR2) offers fewer transistors, but the non-restoring output is a legitimate concern – a degraded signal feeding into the next stage compounds across the chain in ways the first-order Elmore model does not capture. Option S3 (four-NAND2 XOR2) uses fully restored outputs at every stage, is straightforwardly analyzable, and is the conservative starting point for a baseline design.

The baseline therefore uses Option S3 for the XOR2: 16 T per stage, all outputs fully restored, all minimum-sized. Option S4 remains a promising optimization candidate – its lower transistor count and single-pass delay structure could outperform S3 on carry propagation despite the signal integrity concern, which SPICE is well-suited to resolve. This is explored in Section 3.

2 Baseline CMOS Design

2.1 Logic and Operation

Sum. The sum is computed using Option S3 (Section 1.1): two cascaded four-NAND2 XOR2 stages. This implementation was chosen for the baseline because every output is fully restored – each NAND2 drives into a rail-to-rail logic level – eliminating the signal degradation concern present in transmission-gate alternatives. Each NAND4-XOR2 stage costs 16 T, for a total of 32 T on the sum path.

1. **XOR2 (gate 1):** Computes $P = A \oplus B$ via a four-NAND2 XOR2 (16 T). This signal feeds both the sum and carry paths.
2. **XOR2 (gate 2):** Computes $S = P \oplus C_{in}$ via a second four-NAND2 XOR2 (16 T).

Carry. The carry is implemented using Option C3 (Section 1.2):

$$C_{out} = \overline{\overline{AB} \cdot \overline{P \cdot C_{in}}}$$

realized as three NAND2 gates. This implementation was chosen for two reasons. First, $P = A \oplus B$ is already available from gate 1, so the first XOR2 is shared between Sum and Carry at no additional transistor cost. Second, the all-NAND structure avoids series-PMOS pull-up paths entirely – as shown in Table 4, NAND pull-up (143 fs) is at least 2× faster than NOR pull-up (288 fs), making NAND the clear choice for the timing-critical carry chain.

All transistors are sized at the process minimum width $W = L = 22$ nm. Transistor count is used as the area proxy throughout this report; the justification is given in Section 5.

Half Adder (first bit-slice). The first bit-slice has no carry-in – $C_{in} = 0$ – and simplifies to a half adder. Substituting $C_{in} = 0$ into the full adder expressions:

$$S = A \oplus B \oplus 0 = A \oplus B$$
$$C_{out} = \overline{\overline{AB} \cdot \overline{P \cdot 0}} = \overline{\overline{AB} \cdot \overline{0}} = \overline{\overline{AB} \cdot 1} = \overline{\overline{AB}} = AB$$

The second XOR2 stage and two of the three NAND2 gates in the carry logic are therefore unnecessary for the first bit-slice.

2.2 Delay Analysis

The Elmore delay to output node i is defined as (Lecture 13):

$$\tau_{Di} = \sum_{k=1}^{N} C_k R_{ik}$$

where N is the number of nodes in the circuit, C_k is the capacitance at node k, and R_{ik} is the shared resistance between the path from the source s to node i and the path from s to node k:

$$R_{ik} = \sum_{j} R_j \quad \Rightarrow \quad \big(R_j \in [\,\text{path}(s \to i) \cap \text{path}(s \to k)\,]\big)$$

In words, R_{ik} sums only those resistances that are common to both paths. The total capacitance at each node is $C_0 + \gamma C_0$, where C_0 scales with the width of each transistor connected to that node and must be summed over all transistors sharing it.

Oxide Capacitance C_0. The gate oxide capacitance per unit area is:

$$C_0 = C_{ox} = \frac{\varepsilon_{rox}\,\varepsilon_0}{t_{oxe}}$$

Using model file parameters $\varepsilon_{rox} = 3.9$, $\varepsilon_0 = 8.854{\times}10^{-12}\,\mathrm{F/m}$, and the respective oxide thicknesses:

Table 1: Oxide capacitance extracted from 22 nm HP PTM model.

	t_{oxe} (nm)	ε_{rox}	C_0 (mF/m^2)
NMOS	1.05	3.9	32.9
PMOS	1.10	3.9	31.4

Diffusion Capacitance Ratio γ. Diffusion (source/drain) capacitance per unit area is modeled as $C_{diff} = \gamma C_0$, where $\gamma = C_{js}/C_0$ and C_{js} is the zero-bias junction capacitance from the model file. Both device types share $C_{js} = 0.5\,\mathrm{mF/m^2}$:

Table 2: Diffusion capacitance ratio γ extracted from 22 nm HP PTM model.

	C_{js} (mF/m^2)	C_0 (mF/m^2)	γ
NMOS	0.5	32.9	0.0152
PMOS	0.5	31.4	0.0159
Approximate (both)			≈ 0.015

$\gamma \approx 0.015$ for both device types, meaning diffusion capacitance contributes roughly 1.5% of gate capacitance per unit width – a small but non-negligible correction at minimum sizing.

Effective On-Resistance R_{eff}. The on-resistance must account for the effective channel length L_{eff}, which differs from the drawn gate length L due to lateral diffusion of the source/drain regions, parameterized in the model file as `xl`:

$$L_{eff} = L + x_l = 22\,\mathrm{nm} + (-9\,\mathrm{nm}) = 13\,\mathrm{nm}$$

The linear-region on-resistance for a minimum-size transistor ($W = L$) is then:

$$R_{on} = \frac{1}{\mu_0\, C_0\, (W/L_{eff})\, (V_{DD} - V_{th})} = \frac{1}{\mu_0\, C_0\, (V_{DD} - V_{th})} \cdot \frac{L_{eff}}{L}$$

with $V_{DD} = 0.8\,\mathrm{V}$ and scaling factor $L_{eff}/L = 13/22 = 0.59$. The effective resistance used in Elmore delay accounts for the nonlinear switching profile across a full transition, giving $R_{eff} = \frac{3}{4} R_{on}$:

Table 3: On-resistance and effective resistance for minimum-size transistors, corrected for $L_{eff} = 13\,\mathrm{nm}$.

	μ_0 (m^2/Vs)	C_0 (mF/m^2)	V_{th} (V)	$V_{DD} - V_{th}$ (V)	R_{on} (kΩ)	R_{eff} (kΩ)
NMOS	0.04	32.9	0.503	0.297	1.5	1.1
PMOS	0.0095	31.4	0.461	0.339	5.8	4.4

The PMOS effective resistance is approximately $4\times$ that of NMOS, reflecting the lower hole mobility. This ratio directly informs transistor sizing decisions in the optimized design.

Gate Delay Estimates – INV, NAND2, NOR2, and TG XOR2. Using $W = L = 22\,\text{nm}$, the per-transistor gate capacitance is $C_0 = 32.9\,\text{mF/m}^2 \times (22\,\text{nm})^2 = 15.9\,\text{aF}$ and the diffusion capacitance is $\gamma C_0 = 0.24\,\text{aF}$. Each gate output drives one identical gate input ($2C_0$) plus drain diffusions from all transistors connected to that node. For INV this is $2\gamma C_0$ (one PMOS drain, one NMOS drain); for NAND2 and NOR2 it is $3\gamma C_0$ (two parallel PMOS drains + one NMOS drain for NAND2; two parallel NMOS drains + one PMOS drain for NOR2). Source-side junction capacitances are excluded throughout.

INV pull-up:

$$\tau_{pu} = R_{eff,p}(2C_0 + 2\gamma C_0) = 4400 \times 31.7\,\text{aF} \approx 139\,\text{fs}$$

INV pull-down:

$$\tau_{pd} = R_{eff,n}(2C_0 + 2\gamma C_0) = 1100 \times 31.7\,\text{aF} \approx 35\,\text{fs}$$

NAND2 pull-up (worst case: one PMOS conducting):

$$\tau_{pu} = R_{eff,p}(2C_0 + 3\gamma C_0) = 4400 \times 32.5\,\text{aF} \approx 143\,\text{fs}$$

NAND2 pull-down (worst case: two series NMOS):

$$\tau_{pd} = R_{eff,n} \cdot 2\gamma C_0 + 2R_{eff,n}(2C_0 + 3\gamma C_0) = 1100 \times 0.48\,\text{aF} + 2200 \times 32.5\,\text{aF} \approx 72\,\text{fs}$$

NOR2 pull-up (worst case: two series PMOS):

$$\tau_{pu} = R_{eff,p} \cdot 2\gamma C_0 + 2R_{eff,p}(2C_0 + 3\gamma C_0) = 4400 \times 0.48\,\text{aF} + 8800 \times 32.5\,\text{aF} \approx 288\,\text{fs}$$

NOR2 pull-down (worst case: one NMOS conducting):

$$\tau_{pd} = R_{eff,n}(2C_0 + 3\gamma C_0) = 1100 \times 32.5\,\text{aF} \approx 36\,\text{fs}$$

The TG XOR2 is more involved than NAND2 or NOR2 because the value of B determines which of two structurally different signal paths carries A to the output Y. When $B = 1$, A passes through one inverter then directly through TG1. When $B = 0$, A passes through two inverters then through TG2. The intermediate node $\sim A$ is heavily loaded in both cases since it drives the second inverter's gate regardless of B, while the restored A node after INV2 is lightly loaded since it only sees TG2 diffusions. The node capacitances are:

$$\begin{aligned} C(\sim A) &= 2C_0 + 3\gamma C_0 = 32.5\,\text{aF} \\ C(A_{res}) &= 3\gamma C_0 = 0.72\,\text{aF} \\ C(Y) &= 2C_0 + 2\gamma C_0 = 32.3\,\text{aF} \end{aligned}$$

and $R_{TG} = R_{eff,n} \| R_{eff,p} = 0.88\,\text{k}\Omega$.

TG XOR2 worst-case pull-up (B=1, A: 1→0): INV1 PMOS charges $\sim A$, which propagates through TG1 to Y:

$$\tau_{pu} = C(\sim A) \cdot R_{eff,p} + C(Y) \cdot (R_{eff,p} + R_{TG}) = 32.5 \times 4400 + 32.3 \times 5280 \approx 314\,\text{fs}$$

TG XOR2 worst-case pull-down (B=0, A: 1→0): INV1 PMOS first charges $\sim A$ (sequential), then INV2 NMOS discharges A_{res} through TG2 to Y:

$$\tau_{pd} = C(\sim A) \cdot R_{eff,p} + C(A_{res}) \cdot R_{eff,n} + C(Y) \cdot (R_{eff,n} + R_{TG}) = 143 + 1 + 64 \approx 208\,\text{fs}$$

Table 4: Worst-case Elmore delay for minimum-size INV, NAND2, NOR2, and TG XOR2 ($W = L = 22\,\text{nm}$, self-loaded).

Gate	**Transition**	τ **expression**	τ **(fs)**
INV	Pull-up	$R_{eff,p}(2C_0 + 2\gamma C_0)$	139
INV	Pull-down	$R_{eff,n}(2C_0 + 2\gamma C_0)$	35
NAND2	Pull-up	$R_{eff,p}(2C_0 + 3\gamma C_0)$	143
NAND2	Pull-down	$2\gamma C_0 R_{eff,n} + 2R_{eff,n}(2C_0 + 3\gamma C_0)$	72
NOR2	Pull-up	$2\gamma C_0 R_{eff,p} + 2R_{eff,p}(2C_0 + 3\gamma C_0)$	288
NOR2	Pull-down	$R_{eff,n}(2C_0 + 3\gamma C_0)$	36
TG XOR2	Pull-up	$C(\sim A)\, R_{eff,p} + C(Y)(R_{eff,p} + R_{TG})$	314
TG XOR2	Pull-down	$C(\sim A)\, R_{eff,p} + C(A_{res})\, R_{eff,n} + C(Y)(R_{eff,n} + R_{TG})$	208

The delay difference between NAND and NOR is significant – NOR pull-up is 2× slower than NAND pull-up, and NOR pull-down is only fast because it uses a single NMOS. This is a direct consequence of the series PMOS stack in NOR, compounded by the inherently lower hole mobility. These estimates reinforce the design choice made in Section 1.2 to implement the carry using NAND gates wherever possible, avoiding series PMOS paths on the critical $C_{in} \to C_{out}$ chain.

Worst-Case Adder Delay Estimates. Using the gate delays from Table 4, worst-case delays for each adder stage are assembled by taking the maximum pull-up or pull-down delay at each gate along the critical path. These are Elmore estimates and should be understood as relative comparisons between topologies rather than absolute predictions – SPICE results are expected to be significantly larger due to signal slew effects on transmission gates and the inherent limitations of the first-order RC model.

Half adder (first bit-slice, $C_{in} = 0$):

$$t_{A,B\to S} = t_{\text{NAND4-XOR2}} = 3 \times 143 = 429\,\text{fs}$$
$$t_{A,B\to C_{out}} = t_{NAND2} + t_{INV} = 143 + 139 = 282\,\text{fs}$$

Critical path: $A, B \to S$ at **429 fs**.

Full adder (all subsequent bit-slices):

$$t_{C_{in}\to C_{out}} = t_{NAND2} + t_{NAND2} = 143 + 143 = 286\,\text{fs}$$
$$t_{C_{in}\to S} = t_{\text{NAND4-XOR2}} = 429\,\text{fs}$$

Critical path for the ripple chain: $C_{in} \to C_{out}$ at **286 fs**.

8-bit adder (1 half adder + 7 full adders, final sum):

$$t_{total} = t_{HA\to C_{out}} + 7 \times t_{FA,\,C_{in}\to C_{out}} + t_{FA,\,C_{in}\to S} = 282 + 7 \times 286 + 429 = \mathbf{2713}\,\text{fs} \approx \mathbf{2.7}\,\text{ps}$$

Table 5: Worst-case Elmore delay estimates for the baseline adder designs.

Design	**Critical Path**	**Delay (fs)**
Half adder	$A, B \to S$	429
Full adder	$C_{in} \to C_{out}$	286
Full adder	$C_{in} \to S$	429
8-bit adder	Full carry ripple + final sum	2713

2.3 Schematic

The baseline design is built hierarchically. Figure 2 shows the transistor-level schematic of the minimum-sized NAND2 cell ($W = L = 22\,\text{nm}$ for all four transistors) used throughout the carry logic and XOR2 stages.

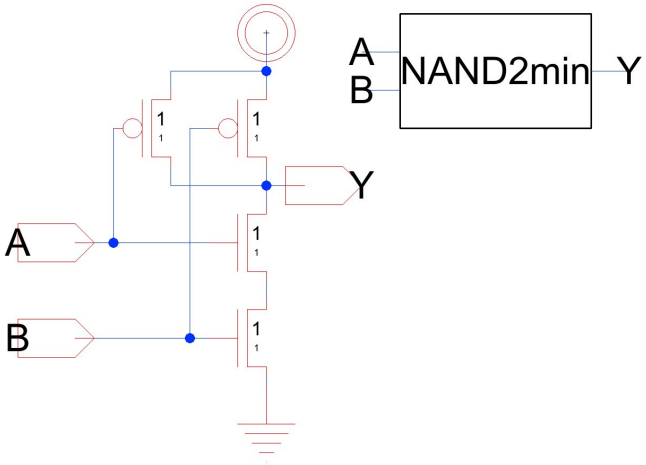

Figure 2: Minimum-sized NAND2 transistor-level schematic. Two PMOS in parallel (pull-up) and two NMOS in series (pull-down), $W = L = 22\,\text{nm}$.

Figure 3 shows the XOR2 gate built from four NAND2 cells (Option S3). Each NAND2 block is the cell in Figure 2.

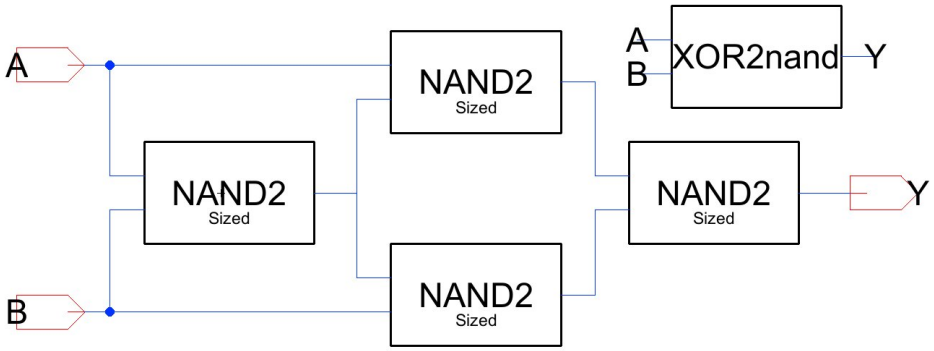

Figure 3: XOR2 gate implemented as four NAND2 cells (Option S3, 16 T). $n_1 = \overline{AB}$; $n_2 = \overline{A \cdot n_1}$; $n_3 = \overline{B \cdot n_1}$; $Y = \overline{n_2 \cdot n_3} = A \oplus B$.

Figure 4 shows the full adder bit-slice, composed of one NAND2 (generating $\overline{AB}$), two XOR2nand stages (for P and S), and two additional NAND2 gates for the carry.

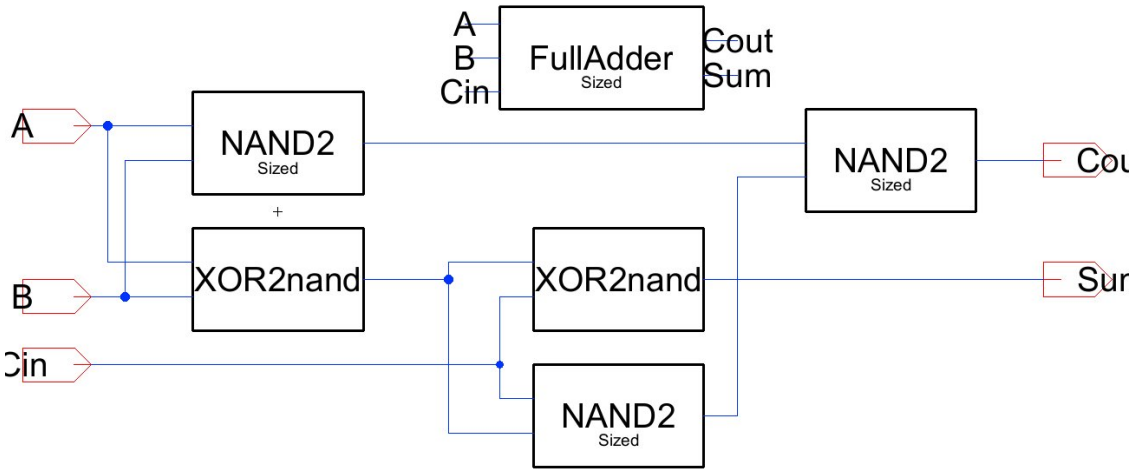

Figure 4: Full adder bit-slice schematic. NAND2 computes $\overline{AB}$ in parallel with XOR2nand computing $P = A \oplus B$. The second XOR2nand computes $S = P \oplus C_{in}$; the two right-hand NAND2s implement $C_{out} = \overline{\overline{AB} \cdot \overline{P \cdot C_{in}}}$.

Figure 5 shows the complete 8-bit ripple-carry adder: a half adder at bit position 0 (no C_{in}) cascaded with seven full adder bit-slices.

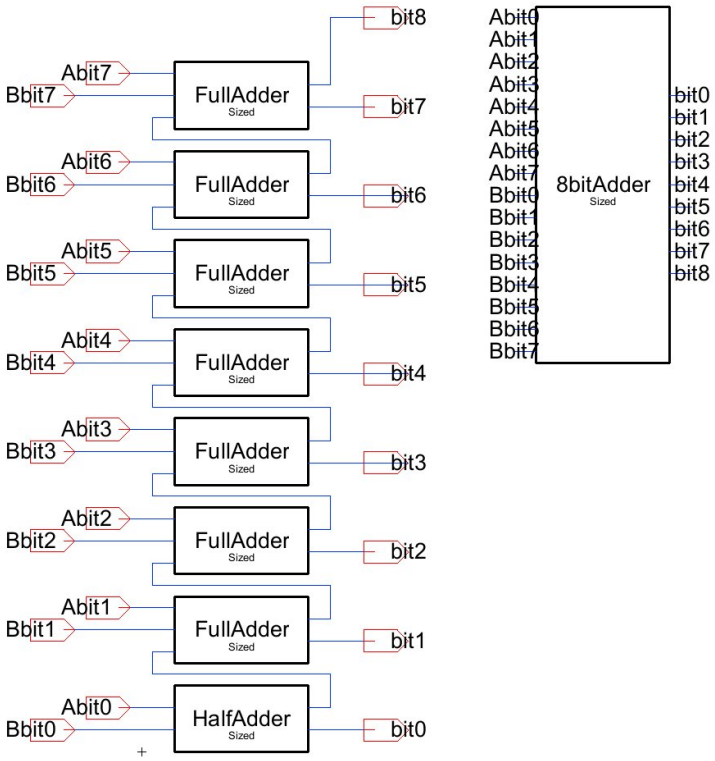

Figure 5: 8-bit ripple-carry adder schematic. Bit position 0 uses a HalfAdder cell (no C_{in}); bit positions 1–7 use the FullAdder cell. The carry chain propagates upward from `bit0` to `bit8`.

2.4 Design Metrics Summary

Logical correctness validation. Functional correctness was verified in two steps. First, all $2^3 = 8$ input combinations $(A, B, C_{in}) \in \{0,1\}^3$ were applied to a single full adder bit-slice and the outputs S and C_{out} confirmed against the truth table derived from Equations 1 and 2. This exhaustively covers the bit-slice logic. Second, the full 8-bit adder was driven with $A = B =$ `0xFF`, $C_{in,0} = 1$ (i.e. $255+255+1$), which forces every bit-slice to produce a carry, exercising the complete carry chain. Together these two tests demonstrate correctness for all input cases.

Testbench conditions. Inputs are driven by a source with drive strength equivalent to one adder output, and all outputs are loaded by the equivalent of one adder input, consistent with the handout requirement for adder-tree use.

Test case justification. The $255 + 1$ carry propagation test is the worst-case delay scenario because it maximises the number of stages through which carry must ripple: every bit-slice is in propagate mode ($A_i \oplus B_i = 1$), so carry must traverse all 8 stages before the final output settles. No other input combination produces a longer critical path.

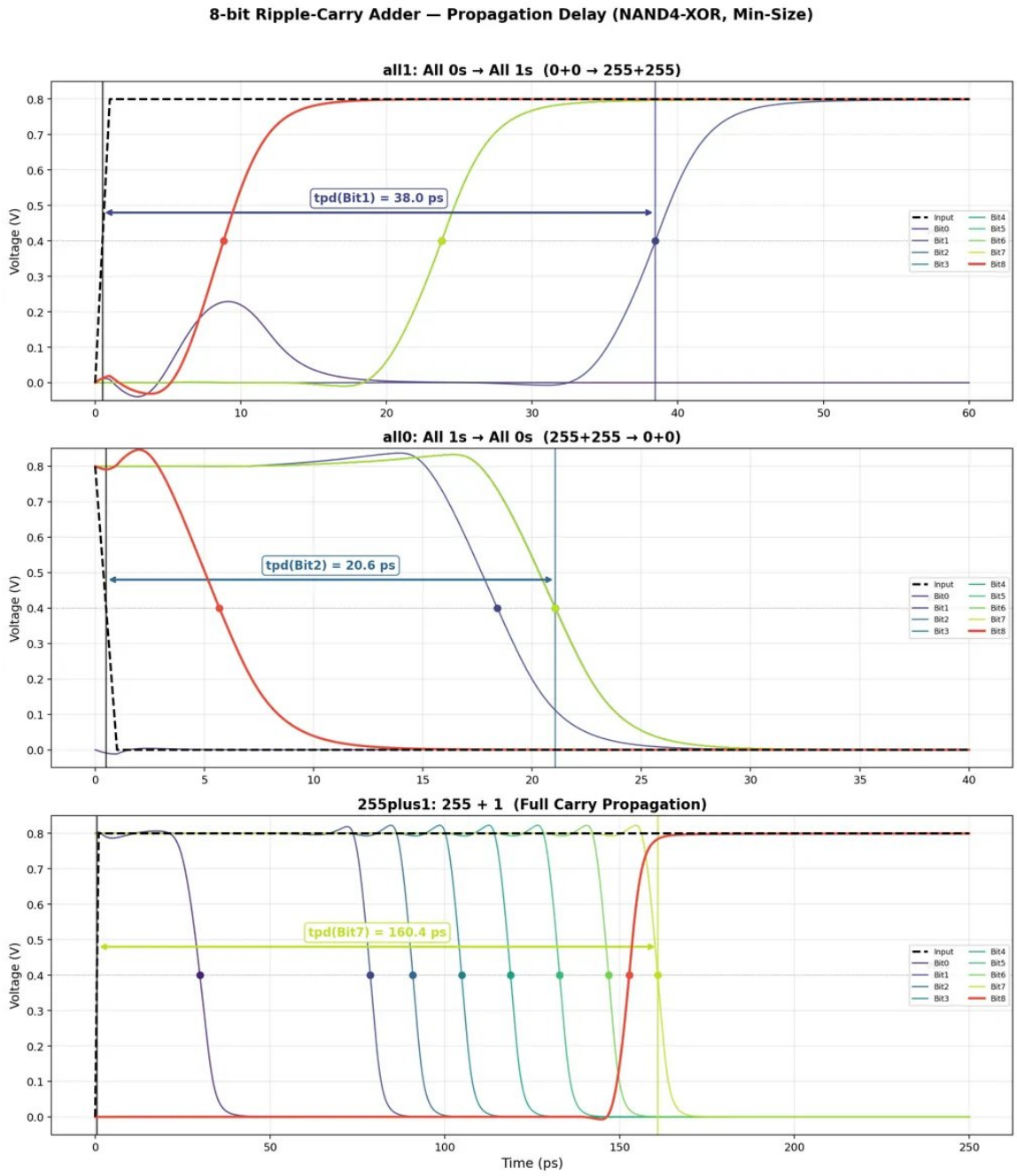

Figure 6: SPICE propagation delay for the baseline 8-bit ripple-carry adder (NAND4-XOR, minimum-size transistors). Top: all inputs $0 \rightarrow 1$, $t_{pd}(\text{Bit1}) = 38.0\,\text{ps}$. Middle: all inputs $1 \rightarrow 0$, $t_{pd}(\text{Bit2}) = 20.6\,\text{ps}$. Bottom: full carry propagation $(255 + 1)$, $t_{pd}(\text{Bit7}) = 160.4\,\text{ps}$.

Table 6: Design metrics summary for the baseline design (NAND4-XOR, 8-bit adder, $V_{DD} = 0.8\,\text{V}$, minimum-size transistors). All values to 2 significant figures.

Metric	Value	Conditions
Worst-case delay	160 ps	$255 + 1$, full carry propagation
Active energy (max)	2.3 fJ	All inputs $0 \to 1$
Active energy (avg)	1.2 fJ	Half inputs switch
Leakage energy (max)	0.025 fJ	$A = B = 1$, scaled to 160 ps delay period*
Leakage energy (min)	0.016 fJ	$A = B = 0$, scaled to 160 ps delay period*
Total transistor count	330 T	Area proxy

*Scaled from 100 ps measurement to the baseline delay period (160 ps). See Section 4 for methodology.

Leakage is maximised at $A = B = 1$ because this turns on the NMOS pull-down network in each NAND2, placing multiple transistors in a high-leakage sub-threshold state simultaneously. Minimum leakage at $A = B = 0$ cuts the pull-down paths and leaves the PMOS devices dominant, which have lower sub-threshold current in this process.

3 Delay-Optimized Design

3.1 Logic and Operation

The carry logic is unchanged from the baseline – three NAND2 gates implementing $C_{out} = \overline{\overline{AB} \cdot \overline{P \cdot C_{in}}}$, with P shared from the sum path.

The single change in the optimized design is the XOR2 implementation: the four-NAND2 construction (Option S3) is replaced by the transmission-gate XOR2 (Option S4). As discussed in Section 1.3, S4 was deferred to the optimized design because the non-restoring output was a theoretical concern. SPICE simulation resolves this: the TG output driving into static NAND2 gates provides sufficient restoration in practice, and the lower transistor count and reduced gate delay of S4 produce a meaningful improvement on the critical $C_{in} \to C_{out}$ carry propagation path.

For reference, the output expressions are unchanged:

$$S = (A \oplus B) \oplus C_{in} \tag{5}$$

$$C_{out} = \overline{\overline{AB} \cdot \overline{P \cdot C_{in}}} \quad \text{where } P = A \oplus B \tag{6}$$

All transistors are minimum-sized ($W = L = 22\,\text{nm}$). Since the logic expressions are identical to the baseline, correctness was verified using the same test cases: all $2^3 = 8$ single-bit-slice input combinations and the full 8-bit $255 + 255 + 1$ carry propagation test.

3.2 Delay Analysis

The carry chain is structurally identical to the baseline – three NAND2 gates with P and $\overline{AB}$ precomputed – so the Elmore delay analysis of Section 2.2 applies directly to the $C_{in} \to C_{out}$ path. The change is in the XOR2 cell: the TG-XOR2 worst-case delays (314 fs pull-up, 208 fs pull-down, from Table 4) replace the NAND4-XOR critical path of $3 \times 143 = 429\,\text{fs}$. Elmore therefore predicts the TG-XOR sum path is faster, and the carry path is unchanged – consistent with the SPICE results below.

The optimized design was explored in two steps.

Step 1 — PMOS sizing. Analytically, widening the NAND2 PMOS to $W_p = 2W_{min}$ was predicted to reduce pull-up delay from 145 fs to 73 fs when driving the TG-XOR2 input, based on the Elmore analysis in Section 2.2. However, SPICE simulation showed that this sizing actually worsened carry propagation delay compared to minimum-sized. The Elmore model captures the resistive improvement but underestimates the additional capacitance added to internal nodes by the wider PMOS, which the full nonlinear simulation captures. PMOS sizing was therefore reverted to minimum width.

Step 2 — TG-XOR2. Replacing the four-NAND2 XOR2 with the transmission-gate XOR2 (Option S4, 8 T per stage) at minimum sizing was then evaluated in SPICE. The TG-XOR2 is roughly 2× slower on rising transitions and marginally faster on falling transitions compared to the NAND4-XOR. However, on carry propagation – where C_{in} ripples through the NAND carry chain stage by stage and the XOR only computes the final sum – the TG-XOR2 design is significantly faster. This is the metric that dominates for an 8-bit ripple-carry adder.

SPICE results for the full carry propagation test ($255 + 1$, forcing carry through all 8 stages) confirm this decisively:

The carry propagation delay improves from 160.4 ps to 116.3 ps – a 27% reduction – at the cost of a 60% slowdown on pure rising transitions. Since worst-case adder performance is governed by carry propagation, the TG-XOR is the superior choice. The signal restoration concern that motivated using NAND4-XOR for the baseline did not manifest as a practical problem in simulation.

3.3 Schematic

The optimized design is structurally identical to the baseline – same NAND2 carry logic, same hierarchical full adder and 8-bit adder composition, same minimum sizing throughout. The sole difference is the XOR2 cell: the four-NAND2 construction is replaced by a transmission-gate XOR2 followed by two inverters in series to restore the signal. Figure 7 shows the transistor-level schematic of this cell. The two output inverters ensure that the output is a fully rail-to-rail logic level before driving the next stage, addressing the signal degradation concern that motivated using the NAND4-XOR for the baseline.

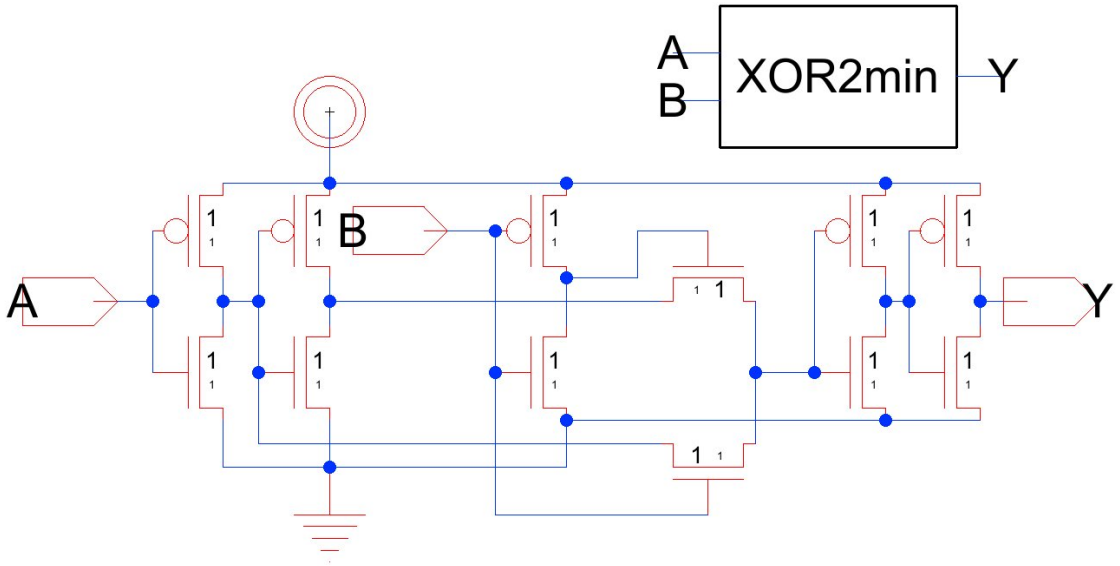

Figure 7: Optimized XOR2 cell: transmission-gate XOR2 followed by two series inverters for signal restoration. All transistors minimum-sized ($W = L = 22\,\text{nm}$).

3.4 Design Metrics Summary

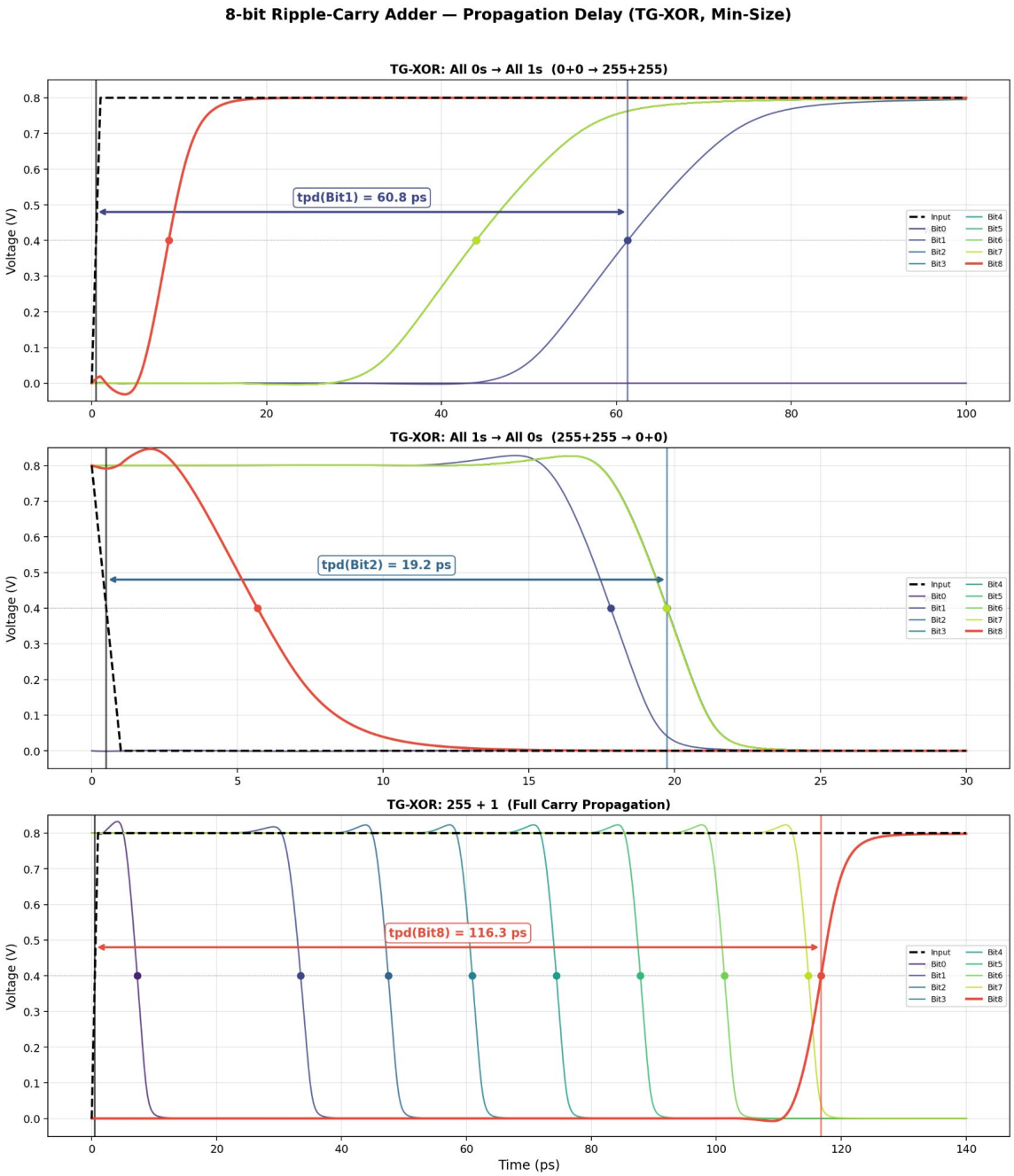

Figure 8: SPICE propagation delay for the optimized 8-bit ripple-carry adder (TG-XOR, minimum-size transistors). Top: all inputs $0 \rightarrow 1$, t_{pd}(Bit1) = 60.8 ps. Middle: all inputs $1 \rightarrow 0$, t_{pd}(Bit2) = 19.2 ps. Bottom: full carry propagation $(255 + 1)$, t_{pd}(Bit8) = 116.3 ps.

Table 7: Design metrics summary for the optimized design (TG-XOR, 8-bit adder, $V_{DD} = 0.8\,\mathrm{V}$, minimum-size transistors). All values to 2 significant figures.

Metric	**Value**	**Conditions**
Worst-case delay	120 ps	$255 + 1$, full carry propagation
Active energy (max)	2.3 fJ	All inputs $0 \to 1$
Active energy (avg)	1.3 fJ	Half inputs switch
Leakage energy (max)	0.21 fJ	$A = B = 1$, scaled to 116 ps delay period*
Leakage energy (min)	0.019 fJ	$A = B = 0$, scaled to 116 ps delay period*
Total transistor count	240 T	Area proxy

*Scaled from 100 ps measurement to the optimized delay period (116 ps). See Section 4 for methodology.

Leakage is maximised at $A = B = 1$ because with both inputs high, both transmission gates are in a conducting state simultaneously: the NMOS is gated on by the high input and the PMOS is gated on by its complemented signal. This creates a resistive path from supply to ground through the pass transistors that does not exist in fully static CMOS gates, which always have a well-defined off path. Minimum leakage at $A = B = 0$ leaves the TGs in the opposite switching state where the sub-threshold paths are suppressed.

4 Design Comparison and Selection Rationale

4.1 Delay

Figure 9 shows the SPICE waveforms for both designs side by side across all three test cases.

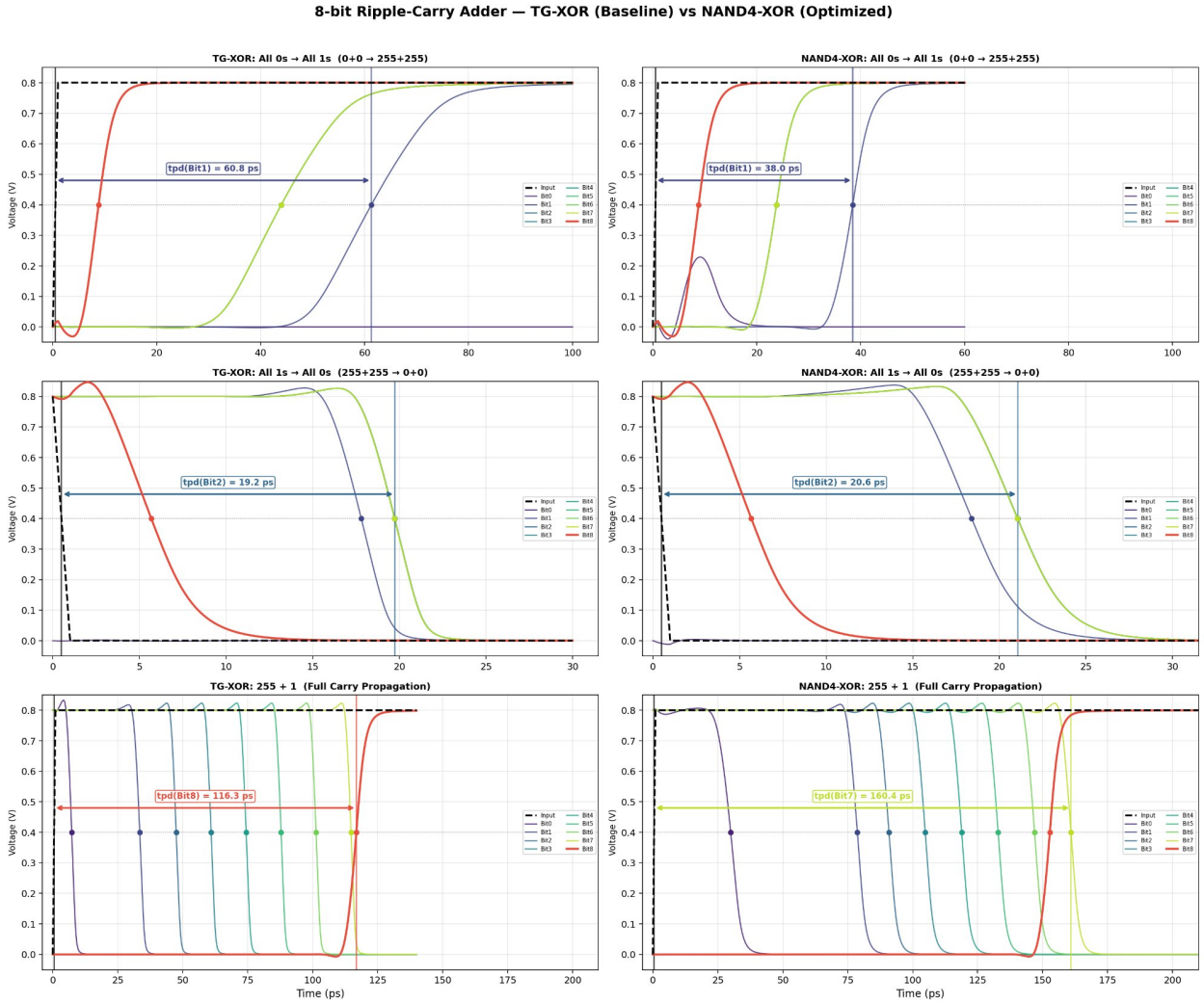

Figure 9: Side-by-side SPICE propagation delay comparison: TG-XOR (left) vs. NAND4-XOR (right). Rows from top to bottom: all $0 \to 1$, all $1 \to 0$, and full carry propagation $(255 + 1)$.

Table 8: Quantitative delay comparison: TG-XOR (optimized) vs. NAND4-XOR (baseline), 8-bit ripple-carry adder, minimum sizing.

Test case	**TG-XOR (ps)**	**NAND4-XOR (ps)**	**Δ (ps)**
All $0 \to 1$ $(0 + 0 \to 255 + 255)$	60.8	38.0	+22.8
All $1 \to 0$ $(255 + 255 \to 0 + 0)$	19.2	20.6	−1.4
Full carry prop. $(255 + 1)$	116.3	160.4	−44.1

The TG-XOR design is slower on rising transitions by 22.8 ps and marginally faster on falling transitions. On carry propagation – the worst-case scenario for a ripple-carry adder – it is 44.1 ps faster, a 27% reduction. Since the carry propagation delay determines the maximum operating frequency of the adder in a real system, the TG-XOR is the superior design despite its disadvantage on non-critical transition types.

4.2 Active Energy

Figure 10 shows the supply current and cumulative energy $E(t) = V_{DD} \int |I|\, dt$ for both designs under max switching (all $0 \to 1$) and average switching (half inputs switch).

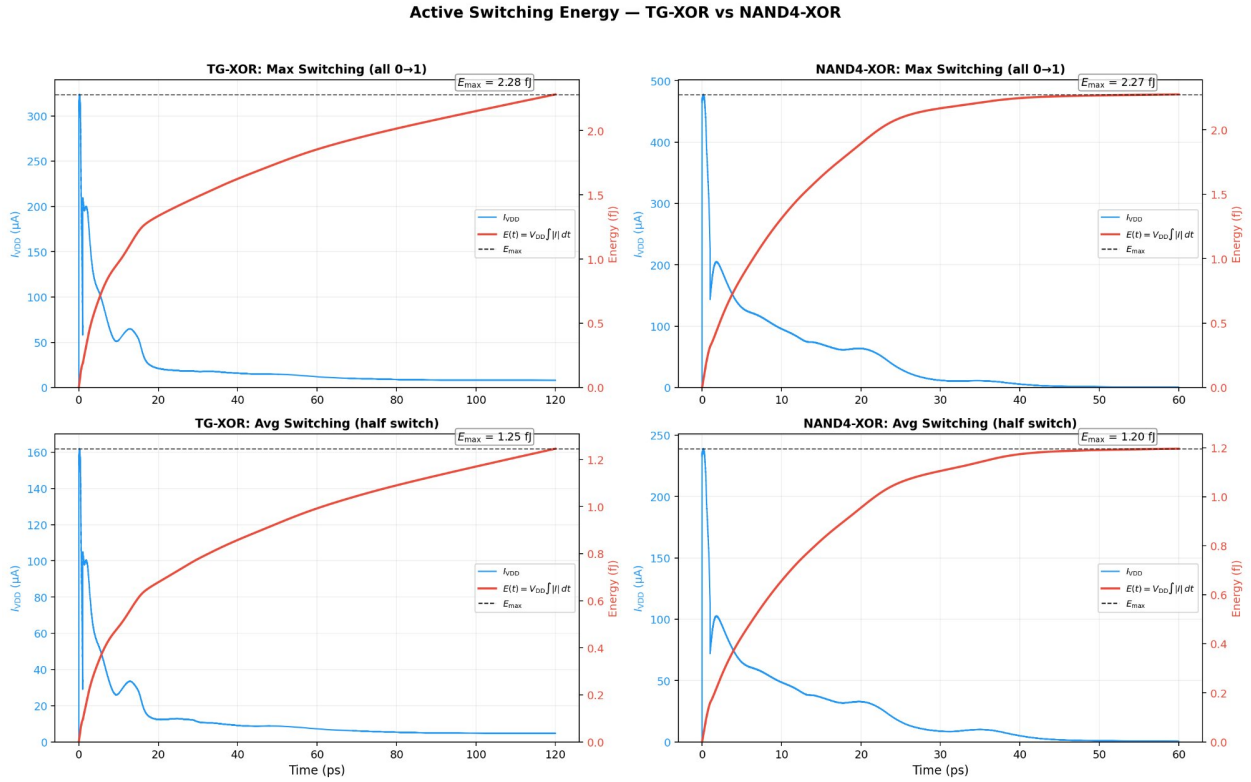

Figure 10: Active switching energy for TG-XOR (left) and NAND4-XOR (right). Top row: max switching (all inputs $0 \to 1$). Bottom row: average switching (half inputs switch).

Active energy is nearly identical between the two designs despite the TG-XOR having 27% fewer transistors: 2.28 fJ vs. 2.27 fJ at max switching, and 1.25 fJ vs. 1.20 fJ at average switching. The TG-XOR's lower transistor count does not translate to lower active energy because the transmission gate pass transistors dissipate energy differently from static pull-up/pull-down networks.

4.3 Leakage Energy

Figure 11 shows leakage energy over a 100 ps static window for both designs at max ($A = B = 1$) and min ($A = B = 0$) leakage input states.

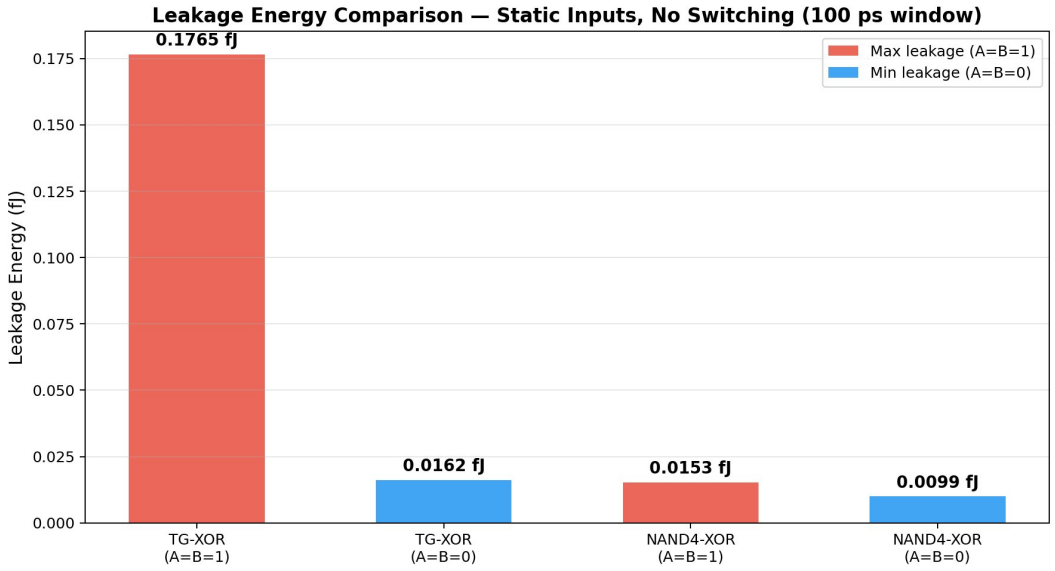

Figure 11: Leakage energy comparison over a 100 ps static window. Max leakage at $A = B = 1$; min leakage at $A = B = 0$.

The handout specifies leakage energy over one adder delay period, which differs between designs (160.4 ps for baseline, 116.3 ps for optimized) and neither equals the 100 ps simulation window. Since leakage current is constant with static inputs, energy scales linearly with time and the measured values can be scaled directly:

$$E_{leakage} = E_{100\,\mathrm{ps}} \times \frac{t_{delay}}{100\,\mathrm{ps}}$$

Table 9: Leakage energy scaled to each design's worst-case delay period.

Design	**Input state**	**Measured (fJ)**	**Scale factor**	**Scaled (fJ)**
Baseline (NAND4-XOR)	$A = B = 1$ (max)	0.0153	$\times 1.604$	0.025
Baseline (NAND4-XOR)	$A = B = 0$ (min)	0.0099	$\times 1.604$	0.016
Optimized (TG-XOR)	$A = B = 1$ (max)	0.1765	$\times 1.163$	0.21
Optimized (TG-XOR)	$A = B = 0$ (min)	0.0162	$\times 1.163$	0.019

The TG-XOR max leakage (0.21 fJ) is over $10\times$ the NAND4-XOR max leakage (0.025 fJ) – the most significant design tradeoff between the two implementations. When $A = B = 1$, both transmission gates in the TG-XOR are partially conducting: the NMOS is gated on by the high input and the PMOS by its complement, creating a resistive sub-threshold path from supply to ground through the pass transistors. This path does not exist in fully static CMOS, where every node is driven by a restoring gate with a well-defined off state. The NAND4-XOR avoids this entirely since all nodes are held by complementary pull-up/pull-down networks. At minimum leakage ($A = B = 0$) the two designs are comparable: 0.019 fJ (TG-XOR) vs. 0.016 fJ (NAND4-XOR).

In summary, the TG-XOR wins on carry propagation delay (27% faster) and area (27% fewer transistors), ties on active energy, but pays a $10\times$ leakage penalty at worst-case inputs – a tradeoff worth considering for always-on or low-power applications.

5 Adder Precision Scaling

The simulation waveforms in Sections 2 and 3 label outputs as Bit0–Bit8, where Bit0–Bit7 are the sum outputs and Bit8 is the final carry-out.

Area and transistor count. Since every transistor in both designs is minimum-sized ($W = L = 22\,\text{nm}$), each transistor occupies the same minimum feature area on silicon. Total circuit area is therefore directly proportional to transistor count – doubling the transistor count exactly doubles the area. This makes transistor count a valid and simple area proxy, and it is used as such throughout this report.

Area scaling. Both designs are built hierarchically from primitive cells. For the baseline (NAND4-XOR):

$$\begin{aligned} \text{NAND2} &= 4\,\text{T} \quad \text{(2 PMOS parallel, 2 NMOS series)} \\ \text{XOR2nand} &= 4 \times \text{NAND2} = 16\,\text{T} \\ \text{INV} &= 2\,\text{T} \\ \text{HA} &= \text{NAND2} + \text{XOR2nand} + \text{INV} = 4 + 16 + 2 = 22\,\text{T} \\ \text{FA} &= 3 \times \text{NAND2} + 2 \times \text{XOR2nand} = 12 + 32 = 44\,\text{T} \end{aligned}$$

For the optimized (TG-XOR), the only change is the XOR2 cell:

$$\begin{aligned} \text{TG-XOR2} &= 6\,\text{T (TG structure)} + 2 \times \text{INV} = 6 + 4 = 10\,\text{T} \\ \text{HA} &= \text{NAND2} + \text{TG-XOR2} + \text{INV} = 4 + 10 + 2 = 16\,\text{T} \\ \text{FA} &= 3 \times \text{NAND2} + 2 \times \text{TG-XOR2} = 12 + 20 = 32\,\text{T} \end{aligned}$$

An N-bit adder uses one HA and $(N-1)$ FAs, giving total transistor counts of $44N - 22$ (baseline) and $32N - 16$ (optimized). The 27% area reduction from the TG-XOR substitution is constant at all widths since the same transistors are saved in every cell regardless of N.

Delay scaling. The per-stage carry delay extracted from the full carry propagation simulation is nearly identical between designs – 13.7 ps per stage (baseline) vs. 13.5 ps (optimized) – because both share the same three-NAND2 carry chain. The speedup in the optimized design comes almost entirely from the first two stages: the TG-XOR half adder settles in 6.8 ps vs. 29.5 ps at Bit0, and the Bit0→Bit1 transition is 26 ps vs. 49 ps. After that, the two carry chains propagate at essentially the same rate. For the baseline, the worst-case output is always the final sum bit S_{N-1}, which must wait for carry ripple and then pass through the final XOR stage. For the optimized design, C_{out} (Bit$N$) arrives approximately 2 ps after S_{N-1}, making the carry endpoint the critical output.

Table 10: Worst-case delay and transistor count vs. adder bit-width, derived from the 8-bit SPICE carry propagation staircase.

Bit width	Baseline delay (ps)	Optimized delay (ps)	Baseline area (T)	Optimized area (T)
1	29.5	6.8	22	16
2	49.0	28.0	66	48
4	96.5	74.0	154	112
8	160.4	116.3	330	240

Both delay and area scale approximately linearly with N. Halving the precision from 8 bits to 4 bits reduces worst-case delay by 35–40% and area by roughly 53%, depending on the design. This near-linear scaling makes low-precision arithmetic units attractive for ML and AI accelerators, where massive parallelism matters more than per-operation precision. A chip that would fit one 32-bit adder tree can instead pack eight 4-bit adder trees in comparable area, enabling far higher throughput for inference workloads where quantized weights and activations (INT8, INT4) maintain acceptable model accuracy. The delay savings compound in multiplier–adder trees, where the critical path traverses multiple addition stages.

Academic Integrity Statement

I, **Lucas Krippendorff**, certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this project.