Department of Electrical Engineering and Computer Science

Howard University

EECE 423: VLSI Design

FALL 2025

Final Project: 4-Bit CMOS Ripple-Carry Adder

By:

Isioma Nwansoh

Mohammed Akinbayo

Instructor: Dr. Hassan Salmani

Introduction

The goal of this project was to design, simulate, and analyze a 4-bit ripple-carry adder (RCA) at the transistor level using HSPICE. The assignment required:

- Building a static CMOS NAND gate as a reusable subcircuit.
- Constructing a 1-bit full adder using only NAND gates.
- Cascading four full adders into a 4-bit RCA using subcircuit instantiation.
- Performing dynamic simulations to measure worst-case delay, power, and Power-Delay Product (PDP) under different transistor sizing strategies.

This report explains the design logic, shows how the simulations were set up, presents the measured results (delay, power, PDP), and discusses the power–performance trade-offs for the three sizing strategies:

1. PMOS-only scaling
2. NMOS-only scaling
3. Proportional scaling of both PMOS and NMOS

Circuit Design and Hierarchy

https://github.com/Mohammed532/VLSILab_2025/blob/main/FINAL/

The testbench included voltage sources for VDD and GND, and PWL input sources for A[3:0] and B[3:0] that step through a sequence of input combinations to exercise carry propagation and high switching activity.

2-Input CMOS NAND Subcircuit

The basic building block is a 2-input static CMOS NAND gate defined as a `.SUBCKT`:

```
.subckt NAND a b y vdd gnd
MMN1 y a x   gnd n105_HVT l=0.03u w=0.54u*WN
MNN2 x b gnd gnd n105_HVT l=0.03u w=0.54u*WN
MMP1 y a vdd vdd p105_HVT l=0.03u w=0.27u*WP
MMP2 y b vdd vdd p105_HVT l=0.03u w=0.27u*WP
.ends
```

The parameters `WP` and `WN` are global scaling factors we change later (.alter statements) to implement the different sizing experiments.

NAND-Only 1-Bit Full Adder (Indian RCA Style)

The 1-bit full adder is implemented using only NAND gates and instantiated as a subcircuit:

```
.subckt adder a b cin sum cout vdd gnd
X1 a   b   n1  vdd gnd NAND
X2 a   n1  n2  vdd gnd NAND
X3 b   n1  n3  vdd gnd NAND
X4 n2  n3  n4  vdd gnd NAND   ; A ⊕ B

X5 n4  cin n5  vdd gnd NAND
X6 n4  n5  n6  vdd gnd NAND
X7 cin n5  n7  vdd gnd NAND
X8 n6  n7  sum vdd gnd NAND   ; (A ⊕ B) ⊕ Cin

X9 n1  n5 cout vdd gnd NAND   ; carry majority function
.ends adder
```

Logic explanation:

Gates X1–X4 implement A ⊕ B using a standard NAND-only XOR construction.

Gates X5–X8 take this intermediate XOR and combine it with `Cin` to implement

SUM = A ⊕ B ⊕ Cin.

Gate X9 combines internal nodes `n1` (related to A and B) and `n5` (related to Cin and A⊕B) to generate COUT, effectively implementing the majority function of the three inputs.

Because this design is fully built from NAND gates, it maps naturally to the previously defined subcircuit and makes resizing straightforward (just change `WP` and `WN` globally).

4-Bit Ripple-Carry Adder

The 4-bit RCA is formed by cascading four full-adder instances:

```
Xrc1 A0 B0 CIN S0 C0 vdd gnd adder N='N' P='P'
Xrc2 A1 B1 C0  S1 C1 vdd gnd adder N='N' P='P'
Xrc3 A2 B2 C1  S2 C2 vdd gnd adder N='N' P='P'
Xrc4 A3 B3 C2  S3 C3 vdd gnd adder N='N' P='P'
```

Carry-out from each stage becomes carry-in to the next. This means the worst-case path is a carry transition that ripples from the least significant bit (S0) all the way to C3 (COUT).

Testbench and Input Patterns

Inputs A[3:0] and B[3:0] are driven by PWL sources that emulate adding several different operand pairs over time. The sequence covers cases such as:

0+0->15+0->15+15->0+15->0+1

Each bit toggles at a different frequency, ensuring high switching activity and forcing carry propagation through different bit positions.

Signals from top to bottom - A0, A1, A2, A3, B1, B2, B3, S0, C0, S1, C1, S2, C2, S3, C3

The initial carry-in `CIN` is held at 0 V in this project, which is typical for a basic RCA test.

Sizing Experiments

To study the power–delay trade-offs, we performed three sizing experiments using `.alter` statements in the testbench:

1. PMOS-only scaling :

```
.alter p_scale_2  .param P = 2 N = 1
.alter p_scale_3  .param P = 3 N = 1
.alter p_scale_4  .param P = 4 N = 1
```

2. NMOS-only scaling:

```
 .alter n_scale_2  .param P = 1 N = 2
.alter n_scale_3  .param P = 1 N = 3
.alter n_scale_4  .param P = 1 N = 4
```

3. Proportional scaling (both):

```
.alter c_scale_2  .param P = 2 N = 2
.alter c_scale_3  .param P = 3 N = 3
.alter c_scale_4  .param P = 4 N = 4
```

`P` and `N` scale PMOS and NMOS widths in the NAND subcircuit. Every device in the entire 4-bit adder is resized consistently, satisfying the assignment requirement that all PMOS or all NMOS across the design share the same W/L for each case.

Measurement Setup

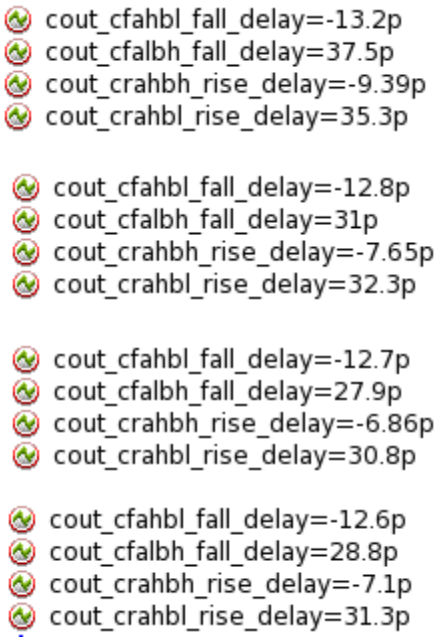
Delay Measurements

Propagation delay is measured using `.measure` statements that trigger when an input crosses `VDD/2` and target when an output crosses `VDD/2`. For example, one of the S0 measurements:

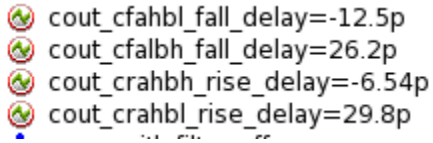
```
.measure tran S0_AhBl_RISE_DELAY
 TRIG V(A0) VAL='SUPPLY/2' RISE=1
 TARG V(S0) VAL='SUPPLY/2' RISE=1
```

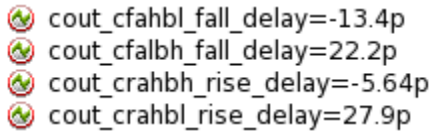
For COUT we focus on worst-case transitions involving carry rippling

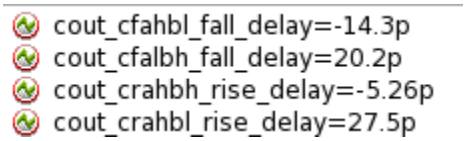
Proportional Scaling



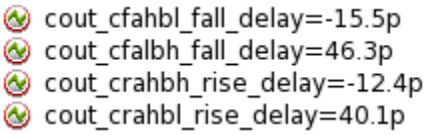
PMOS only scaling

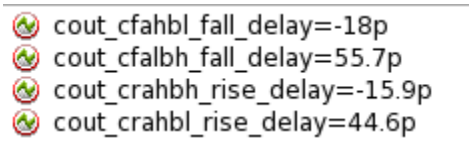


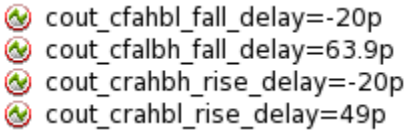




NMOS only scaling







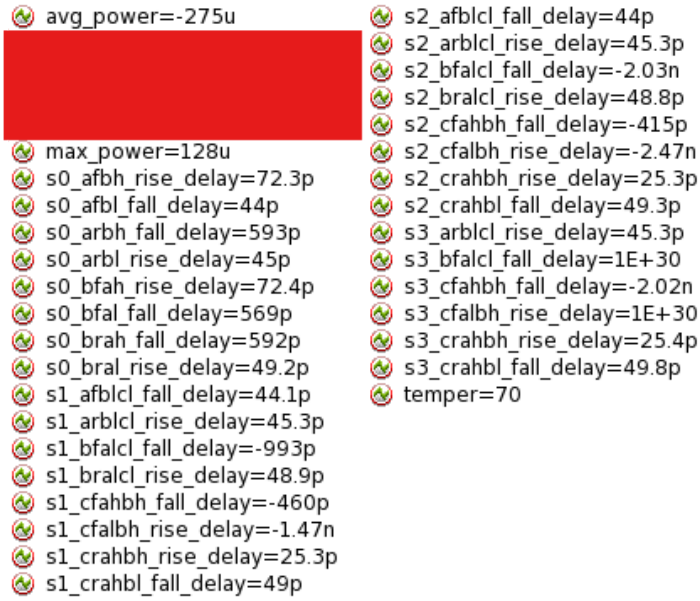
From these we extract worst-case rise and fall delays for COUT for each sizing case.

Power Measurements

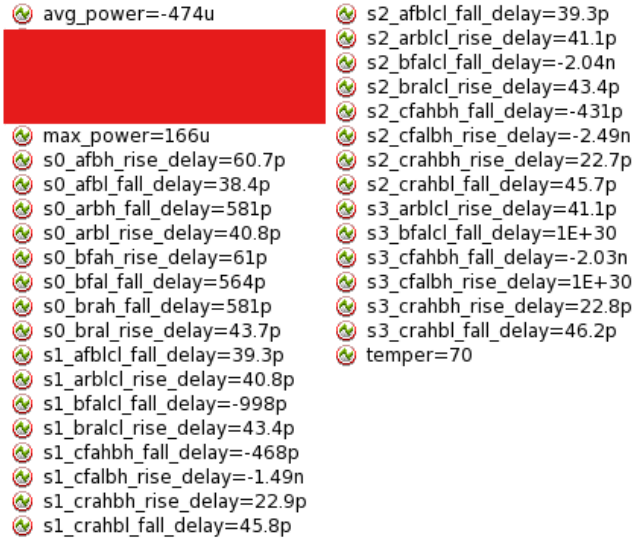
Average and peak power are obtained using `.measure` statements on the supply current:

`avg_power` – average power over the entire transient

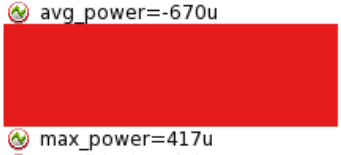
`max_power` – peak instantaneous power



Proportional Scaling



avg_power=-670u



max_power=417u
s0_afbh_rise_delay=57.1p
s0_afbl_fall_delay=36.7p
s0_arbh_fall_delay=577p
s0_arbl_rise_delay=39.3p
s0_bfah_rise_delay=58.1p
s0_bfal_fall_delay=562p
s0_brah_fall_delay=578p
s0_bral_rise_delay=42p
s1_afblcl_fall_delay=38.1p
s1_arblcl_rise_delay=39.6p
s1_bfalcl_fall_delay=-1n
s1_bralcl_rise_delay=42p
s1_cfahbh_fall_delay=-471p
s1_cfalbh_rise_delay=-1.49n
s1_crahbh_rise_delay=22p
s1_crahbl_fall_delay=44.8p
s2_afblcl_fall_delay=38.1p
s2_arblcl_rise_delay=39.7p
s2_bfalcl_fall_delay=-2.04n
s2_bralcl_rise_delay=41.8p
s2_cfahbh_fall_delay=-436p
s2_cfalbh_rise_delay=-2.5n
s2_crahbh_rise_delay=21.9p
s2_crahbl_fall_delay=45p
s3_arblcl_rise_delay=39.7p
s3_bfalcl_fall_delay=1E+30
s3_cfahbh_fall_delay=-2.03n
s3_cfalbh_rise_delay=1E+30
s3_crahbh_rise_delay=21.8p
s3_crahbl_fall_delay=45.5p
temper=70

avg_power=-864u
cout_cfahbl_fall_delay=8.31n
cout_cfalbh_fall_delay=2.07n
cout_crahbh_rise_delay=2.08n
cout_crahbl_rise_delay=30.6p
max_power=527u
s0_afbh_rise_delay=56.2p
s0_afbl_fall_delay=37.1p
s0_arbh_fall_delay=575p
s0_arbl_rise_delay=38.7p
s0_bfah_rise_delay=56.1p
s0_bfal_fall_delay=561p
s0_brah_fall_delay=576p
s0_bral_rise_delay=41.3p
s1_afblcl_fall_delay=35.8p
s1_arblcl_rise_delay=39.2p
s1_bfalcl_fall_delay=-1n
s1_bralcl_rise_delay=41.2p
s1_cfahbh_fall_delay=-473p
s1_cfalbh_rise_delay=-1.5n
s1_crahbh_rise_delay=21.5p
s1_crahbl_fall_delay=44.4p
s2_afblcl_fall_delay=37.4p
s2_arblcl_rise_delay=39.1p
s2_bfalcl_fall_delay=-2.04n
s2_bralcl_rise_delay=41.2p
s2_cfahbh_fall_delay=-438p
s2_cfalbh_rise_delay=-2.5n
s2_crahbh_rise_delay=21.5p
s2_crahbl_fall_delay=44.4p
s3_arblcl_rise_delay=39.1p
s3_bfalcl_fall_delay=1E+30
s3_cfahbh_fall_delay=-2.03n
s3_cfalbh_rise_delay=1E+30
s3_crahbh_rise_delay=21.2p
s3_crahbl_fall_delay=45.1p
temper=70

PMOS only scaling

avg_power=-382u

max_power=241u
s0_afbh_rise_delay=53.1p
s0_afbl_fall_delay=36.1p
s0_arbh_fall_delay=573p
s0_arbl_rise_delay=37.6p
s0_bfah_rise_delay=53.2p
s0_bfal_fall_delay=560p
s0_brah_fall_delay=573p
s0_bral_rise_delay=39.9p
s1_afblcl_fall_delay=36.2p
s1_arblcl_rise_delay=38.1p
s1_bfalcl_fall_delay=-1n
s1_bralcl_rise_delay=39.8p
s1_cfahbh_fall_delay=-474p
s1_cfalbh_rise_delay=-1.5n
s1_crahbh_rise_delay=20.7p
s1_crahbl_fall_delay=43.3p
s2_afblcl_fall_delay=36.2p
s2_arblcl_rise_delay=38.1p
s2_bfalcl_fall_delay=-2.04n
s2_bralcl_rise_delay=39.8p
s2_cfahbh_fall_delay=-441p
s2_cfalbh_rise_delay=-2.51n
s2_crahbh_rise_delay=20.7p
s2_crahbl_fall_delay=43.3p
s3_arblcl_rise_delay=38.1p
s3_bfalcl_fall_delay=1E+30
s3_cfahbh_fall_delay=-2.04n
s3_cfalbh_rise_delay=1E+30
s3_crahbh_rise_delay=20.6p
s3_crahbl_fall_delay=43.9p
temper=70

avg_power=-480u

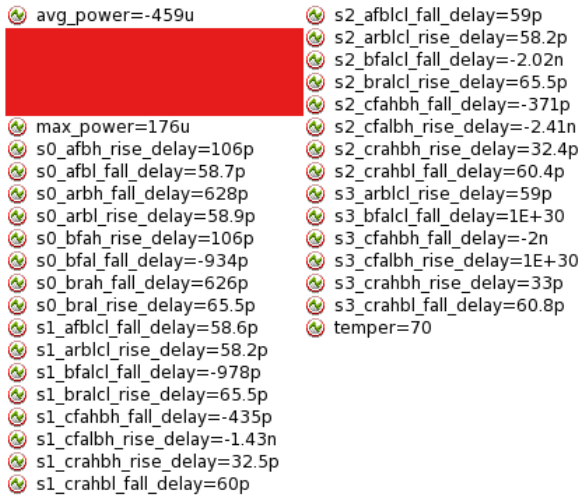
max_power=142u
s0_afbh_rise_delay=44.7p
s0_afbl_fall_delay=33.1p
s0_arbh_fall_delay=567p
s0_arbl_rise_delay=36p
s0_bfah_rise_delay=45.8p
s0_bfal_fall_delay=558p
s0_brah_fall_delay=566p
s0_bral_rise_delay=37.6p
s1_afblcl_fall_delay=33.1p
s1_arblcl_rise_delay=36.8p
s1_bfalcl_fall_delay=-1n
s1_bralcl_rise_delay=37.6p
s1_cfahbh_fall_delay=-478p
s1_cfalbh_rise_delay=-1.5n
s1_crahbh_rise_delay=19.6p
s1_crahbl_fall_delay=42.1p
s2_afblcl_fall_delay=33.1p
s2_arblcl_rise_delay=36.8p
s2_bfalcl_fall_delay=-2.04n
s2_bralcl_rise_delay=37.6p
s2_cfahbh_fall_delay=-448p
s2_cfalbh_rise_delay=-2.52n
s2_crahbh_rise_delay=19.4p
s2_crahbl_fall_delay=42.4p
s3_arblcl_rise_delay=36.8p
s3_bfalcl_fall_delay=1E+30
s3_cfahbh_fall_delay=-2.04n
s3_cfalbh_rise_delay=1E+30
s3_crahbh_rise_delay=19.5p
s3_crahbl_fall_delay=42.6p
temper=70

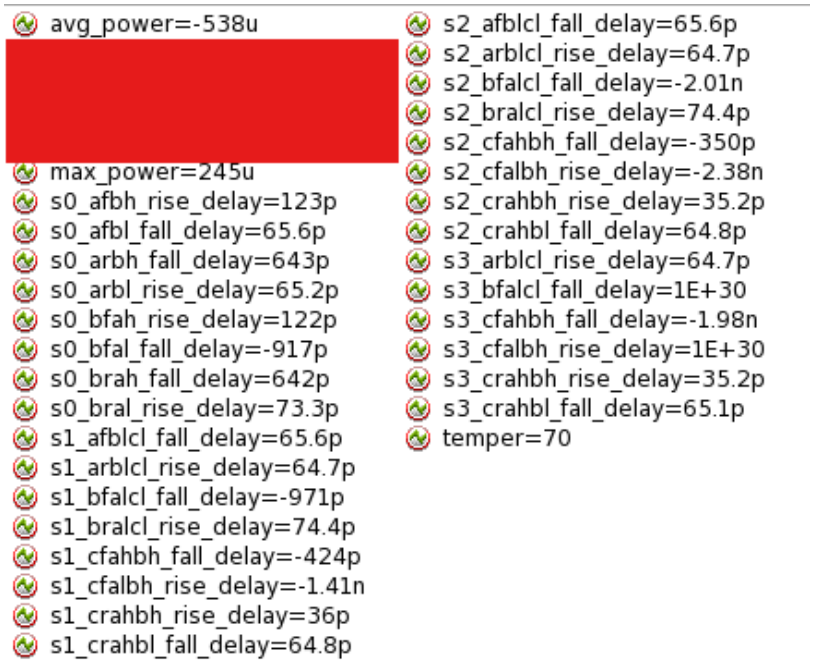
---

avg_power=-564u

max_power=158u
s0_afbh_rise_delay=42.6p
s0_afbl_fall_delay=32.5p
s0_arbh_fall_delay=563p
s0_arbl_rise_delay=36p
s0_bfah_rise_delay=45.1p
s0_bfal_fall_delay=557p
s0_brah_fall_delay=565p
s0_bral_rise_delay=37.1p
s1_afblcl_fall_delay=32.3p
s1_arblcl_rise_delay=36.8p
s1_bfalcl_fall_delay=-1.01n
s1_bralcl_rise_delay=37.1p
s1_cfahbh_fall_delay=-481p
s1_cfalbh_rise_delay=-1.51n
s1_crahbh_rise_delay=19.4p
s1_crahbl_fall_delay=42.7p
s2_afblcl_fall_delay=32.4p
s2_arblcl_rise_delay=36.7p
s2_bfalcl_fall_delay=-2.05n
s2_bralcl_rise_delay=37.1p
s2_cfahbh_fall_delay=-453p
s2_cfalbh_rise_delay=-2.52n
s2_crahbh_rise_delay=19.3p
s2_crahbl_fall_delay=42.5p
s3_arblcl_rise_delay=36.8p
s3_bfalcl_fall_delay=1E+30
s3_cfahbh_fall_delay=-2.04n
s3_cfalbh_rise_delay=1E+30
s3_crahbh_rise_delay=19.2p
s3_crahbl_fall_delay=43.2p
temper=70

NMOS scaling





Results

Functional Verification

The full waveform plot (see page 6) shows that:

S0–S3 and C3 (COUT) correctly reflect the binary sum of A and B for all test vectors.

When the sum exceeds 15, the carry ripples through the chain and C3 asserts.

This confirms the logical correctness of the NAND-based 4-bit RCA.

Worst-Case COUT Delay

From the COUT measurement screenshots, the worst positive propagation delays for COUT were extracted. Example subset:

Baseline (1× both): rise ≈ 32.3 ps, fall ≈ 31.0 ps

Both scaled 2×: rise ≈ 30.8 ps, fall ≈ 27.9 ps

Both scaled 3×: rise ≈ 40.1 ps, fall ≈ 46.3 ps

Both scaled 4×: rise ≈ 49.0 ps, fall ≈ 63.9 ps

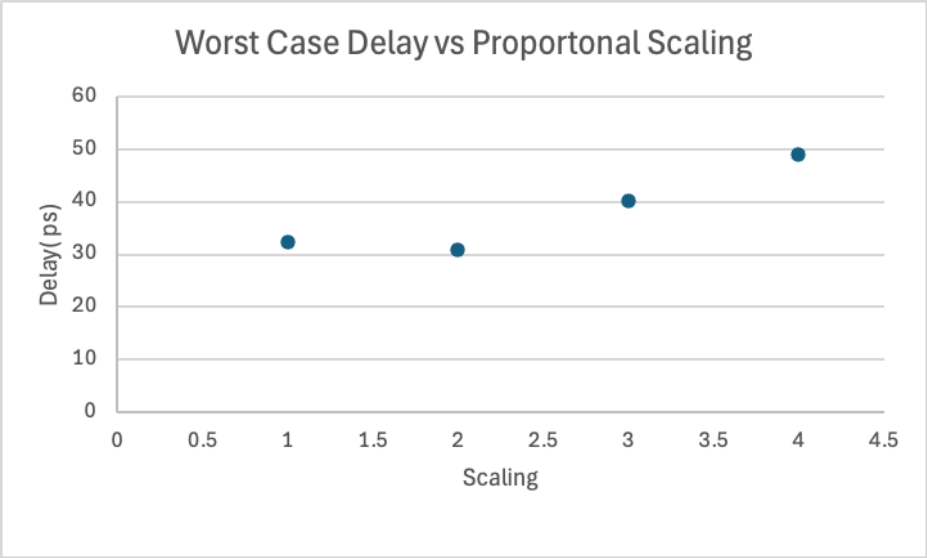
For PMOS-only scaling, delays steadily improve:

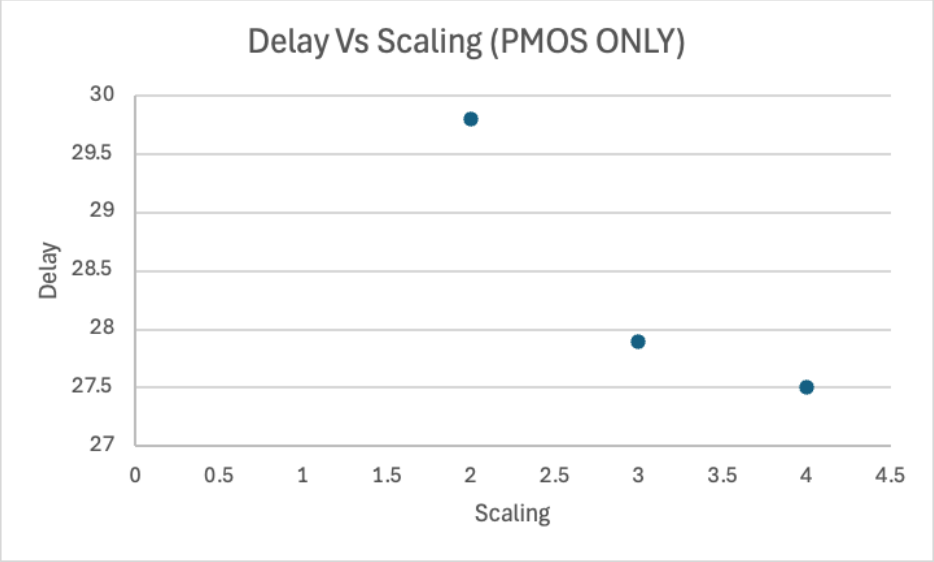
P2×: rise ≈ 29.8 ps, fall ≈ 26.2 ps

P3×: rise ≈ 27.9 ps, fall ≈ 22.2 ps

P4×: rise ≈ 27.5 ps, fall ≈ 20.2 ps

For NMOS-only scaling, delays actually worsen, reaching ~44–56 ps.





Proportional scaling initially reduces delay (1×→2×) but beyond that, increasing capacitance dominates and delay grows again.

PMOS-only scaling gives the best and most consistent delay improvement.

NMOS-only scaling is the worst; the PMOS network becomes the bottleneck.

Power Results

From the `avg_power` values:

Baseline: 275 µW

Both scaled:

2×: 474 µW

3×: 670 µW

4×: 864 µW

PMOS-only:

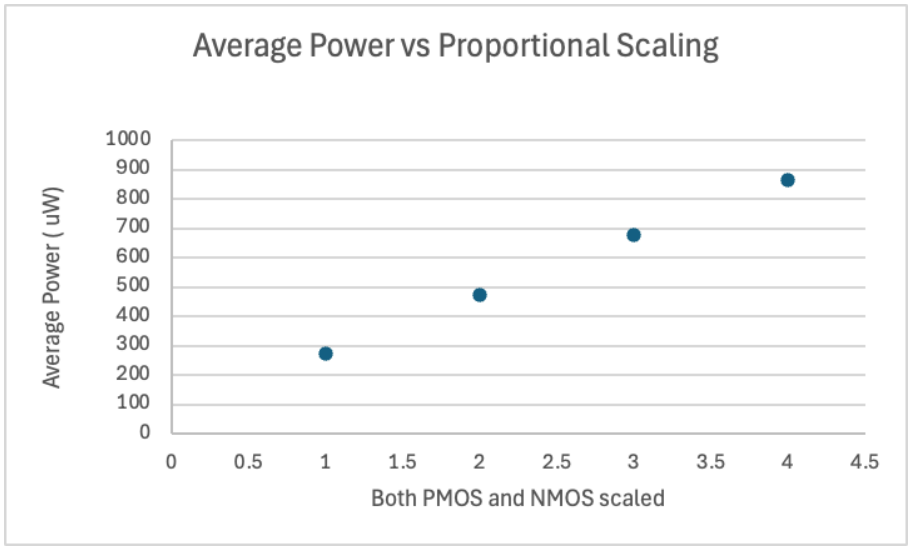
2×: 538 µW

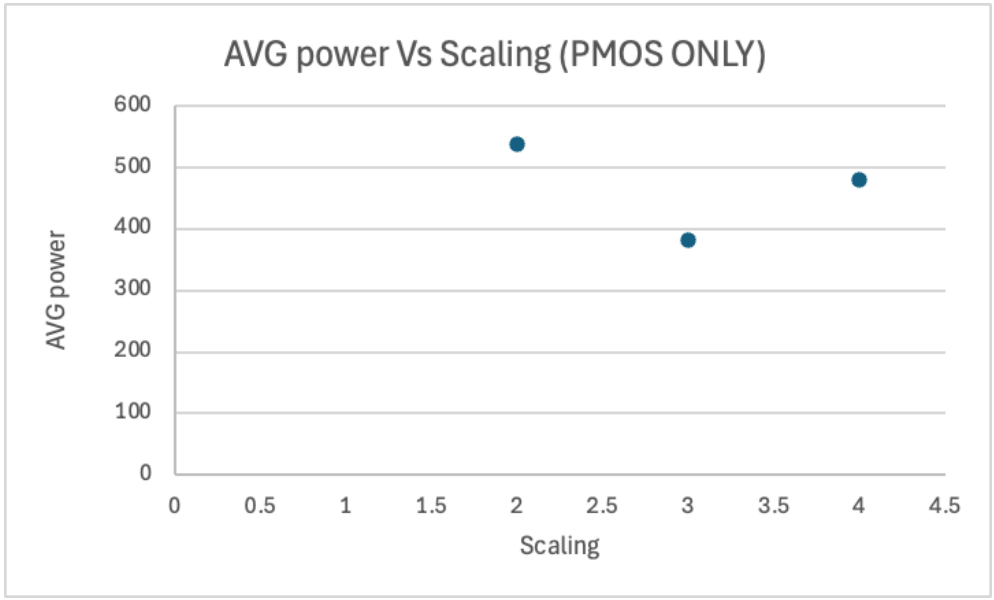
3×: 382 µW

4×: 480 µW

NMOS-only

4×: 564 µW


Average Power vs Proportional Scaling
Average Power (uW)
1000
900
800
700
600
500
400
300
200
100
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Both PMOS and NMOS scaled


AVG power Vs Scaling (PMOS ONLY)
AVG power
600
500
400
300
200
100
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Scaling

Interpretation:

Proportional scaling significantly increases dynamic power: more width → higher load capacitance on every node → more switching energy.

PMOS-only scaling has a moderate power overhead, especially at 3× where average power remains comparable while delay improves.

NMOS-only scaling is again inefficient: relatively high power with no speed benefit.

Power-Delay Product (PDP)

The PDP was computed as:

[PDP ={Average Power} * {Worst-Case COUT Delay}]

(Using COUT rise delay and average power for each case.)

For proportional scaling:

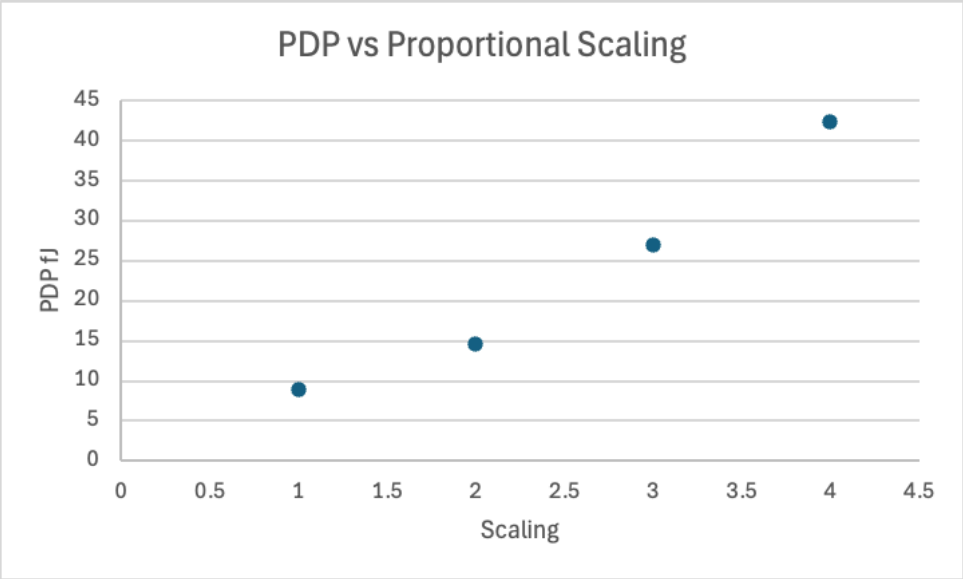
Scale	**PDP (fJ)**
1x	8.9
2x	14.6
3x	26.9
4x	42.3

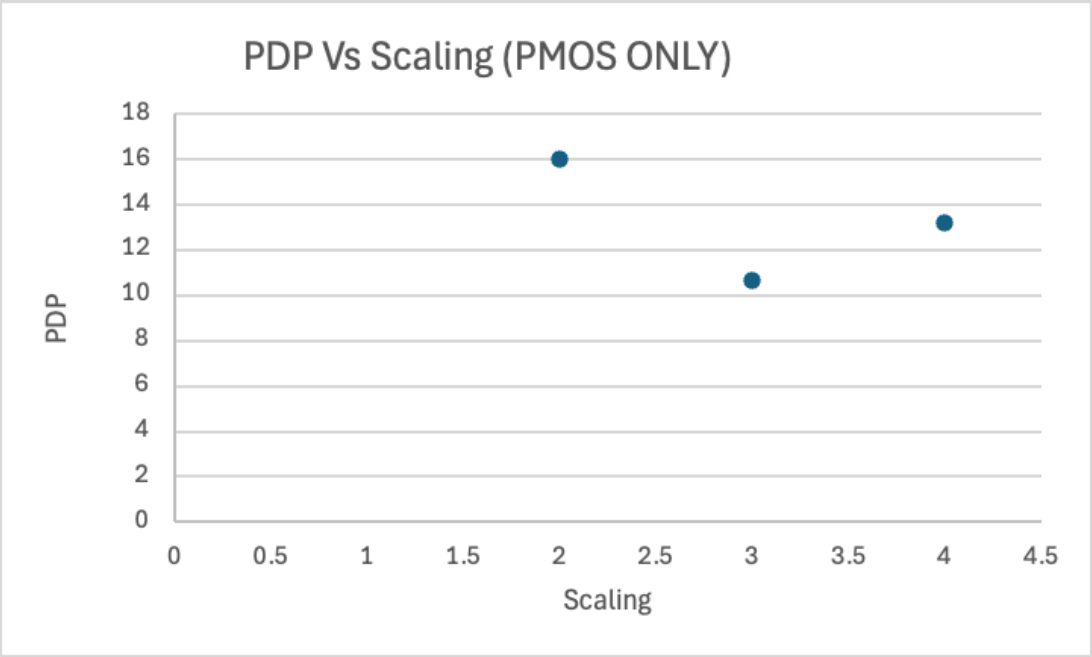
For PMOS-only scaling:

Scale	PDP (fJ)
2x	16.0
3x	10.7
4x	13.2

NMOS-only

4×: PDP ≈ 25.1 fJ.





Important points:

For both-scaled, PDP increases monotonically with scaling.

For PMOS-only, PDP shows a minimum at P=3×, making it the most energy-efficient design.

NMOS-only again performs poorly, with high PDP.

Discussion

Which Sizing Strategy Optimized Speed?

The fastest propagation delay is obtained when both PMOS and NMOS are modestly scaled (2×) or when PMOS is scaled more aggressively (3–4×). This is because:

- Wider NMOS transistors speed up the pull-down path.
- Wider PMOS transistors compensate for low hole mobility, speeding up the pull-up path.
- Balanced scaling (both) initially reduces delay as drive strength dominates, but beyond 2× the extra capacitance slows down edges.

So if the absolute minimum delay is the main goal (ignoring power), moderate proportional scaling is the best choice.

Which Sizing Strategy Minimized PDP?

PDP balances speed and power. The data shows the lowest PDP occurs at PMOS scaling 3×:

- Delay is significantly improved compared to baseline.
- Average power remains low compared with heavy proportional scaling.

This means PMOS-only 3× scale provides the best energy per operation, making it the most efficient solution in practice.

PMOS vs NMOS Strength Effects

In a NAND gate:

The pull-up network (PMOS in parallel) sets the speed of rising edges.

The pull-down network (series NMOS) sets the speed of falling edges.

Therefore:

If we only scale NMOS, falling edges may become slightly faster, but rising edges remain limited by weak PMOS. The resulting imbalance and added capacitance can even increase overall delay .

If we scale PMOS, both rising delay and overall balance improve, producing better delay and PDP.

This explains why PMOS-only scaling outperforms NMOS-only scaling in both delay and PDP experiments.

Overall Power–Performance Trade-Offs

This project illustrates a classic VLSI trade-off:

Larger transistors → lower resistance → lower delay

But also larger capacitance → higher dynamic power and eventually higher delay beyond a point.

Our results show:

There is a sweet spot where slight upsizing improves performance without excessive power (PMOS 3×).

Beyond that, performance actually degrades, and energy per operation increases

Conclusion

Proportional scaling minimizes delay only for small factors but greatly increases power and PDP for larger factors.

NMOS-only scaling is ineffective; it increases power and does not improve delay.

PMOS-only scaling , particularly at 3× width , offers the best energy efficiency (lowest PDP) while still improving delay relative to baseline.

This matches theoretical expectations about CMOS transistor behavior and demonstrates how careful device sizing is essential for balancing speed, power, and energy in modern VLSI design.