University of Pennsylvania ESE 5700

Project 1, Part 3: the MAC unit

Tyrone Marhguy • Victor Wanjohi

Due 27 October 2026

GPDK 18 nm $V_{DD} = 0.8$ V $D = 28$ $T_{CLK} = 2.5$ ns $t_{dot} = 42.5$ ns $E_A = 8.9$ pJ $\text{FOM}_{MAC} = 1.1 \times 10^5$

All simulations are schematic level (no extracted parasitics). Bitline capacitance is $C_{BL} = 3.5$ fF per bitline (`cbw`, Part 2). Each block output under test drives one `inv_2x`. The SRAM is treated as a black box with the Part 2 interface and $t_{access} = 488$ ps.

Contents

1 Team members and contributions . . . 2
1.1 Architecture . . . 2
1.2 Block inventory . . . 4
2 Sequential elements . . . 4
2.1 Style and justification . . . 4
2.2 Clock-to-Q . . . 6
2.3 Setup time . . . 7
2.4 Hold time . . . 14
2.5 Energy per clock . . . 16
2.6 Clock-input capacitance . . . 17
2.7 Enabled registers . . . 20
3 Datapath . . . 23
3.1 Leaf cells . . . 23
3.2 Multiplier . . . 23
3.3 Accumulator adder . . . 24
3.4 Logical-effort analysis . . . 26
3.5 Unit tests . . . 27
4 Sequencer . . . 34
4.1 State diagram . . . 34
4.2 Counter and controller . . . 34
4.3 One-dot-product timing . . . 39
5 Timing closure . . . 40
5.1 Critical path . . . 40
5.2 Period sweep . . . 41
5.3 Hold check . . . 44
6 Verification . . . 44
6.1 Test vectors . . . 44
6.2 Read-latch fix . . . 46
7 Metrics and figure of merit . . . 48
7.1 Reported metrics . . . 48
7.2 Energy . . . 48
7.3 Memory vs. datapath energy . . . 50
8 What limited the design . . . 51
8.1 Delay . . . 51
8.2 Area and energy . . . 51
8.3 Choices we kept . . . 51

1. Team members and contributions

Member	Contribution
Victor Wanjohi	Part 1 write-stability analysis; Part 2 column circuits, decoder, and SRAM-macro benches (T_{CLK}, power, access); Part 3 leaf cells (`tg`, `xor2`, `mux2`), 4×4 array multiplier (256/256, 1.14–1.24 ns), 4-bit counter (38/38) and `controller_mac` (60/60); sequencer schematics; logical-effort sizing; datapath and sequencer write-up.
Tyrone Marhguy	Part 1 6T cell: schematic, β sweep, pull-up/pass/pull-down sizing, custom layout, DRC and LVS; Part 2 SRAM macro, instantiated twice as W and X; Part 3 `top` schematic; transistor-level register (`d_ff`) and characterization (§2); 12-bit adder, `register_12b` and accumulator; `tb_top` Tests A/B/C, clock-period sweep, VDD energy integrals and FOM_{MAC}.

1.1 Architecture

Two copies of the Part 2 macro hold the weights W and activations X; a static-CMOS array multiplier, a 12-bit ripple-carry accumulator, a 4-bit address counter and a two-bit controller (all registers are transmission-gate master-slave flip-flops) compute $\text{ACC} = \sum_{i=0}^{15} W[i]\,X[i]$. The read is pipelined against the multiply-accumulate (read in cycle n, accumulate in $n+1$), so a dot product takes 17 cycles from START to DONE. Tests A, B and C all pass at 10, 4.0 and 2.5 ns. At 2.0 ns, A finishes at 0x34D and C at 0x320 (pass) while B finishes at 0xC10 instead of 0xE10 (fail). At 1.75 ns and 1.5 ns all three fail. The selected period is therefore $T_{CLK} = 2.5$ ns (400 MHz), $t_{dot} = 42.5$ ns; the 2.0–2.5 ns interval was not swept further. Energy and FOM use that same period: Test A 8.870 pJ, $\text{FOM}_{MAC} = 1.1 \times 10^5$ μm^2·pJ·ns. The multiplier's worst case is about 1.2 ns, so it, not the 488 ps SRAM read, sets the period. Area 291.8 μm^2 *(est.)*.

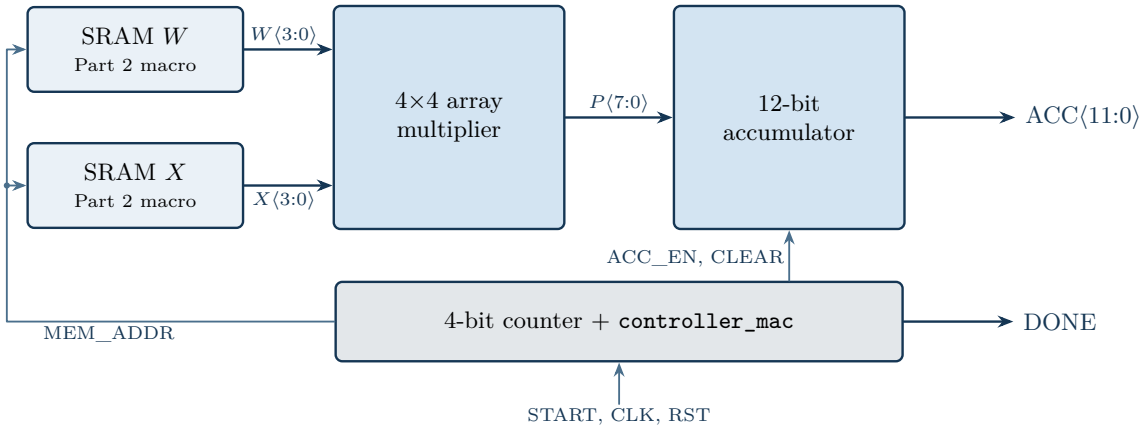


Figure 1: Top-level MAC: two Part 2 macros feed the array multiplier; the product accumulates for 16 cycles. The counter/controller sequences MEM_ADDR, ACC_EN and DONE (17 cycles START→DONE).

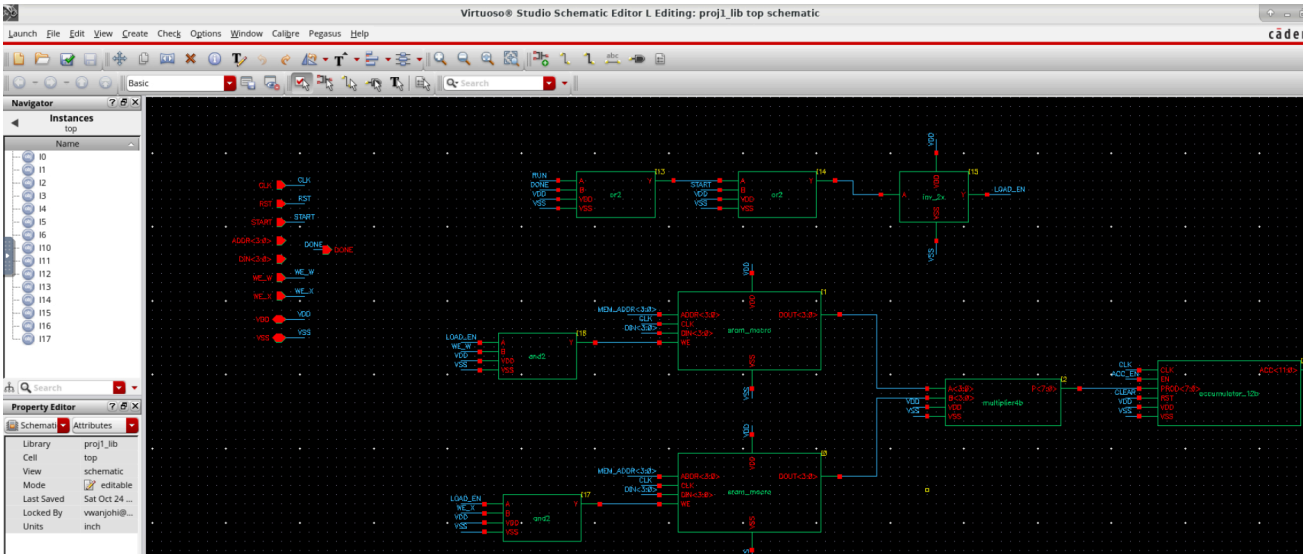


Figure 2: `top`, left half: the two `sram_macro` instances (I1 weights, written when LOAD_EN·WE_W; I0 activations, LOAD_EN·WE_X), sharing MEM_ADDR, CLK and DIN; their DOUT buses feed `multiplier4b` (I2), whose product feeds `accumulator_12b` (I3). Top: LOAD_EN = $\overline{\text{RUN} + \text{DONE} + \text{START}}$ (two `or2`, `inv_2x`).

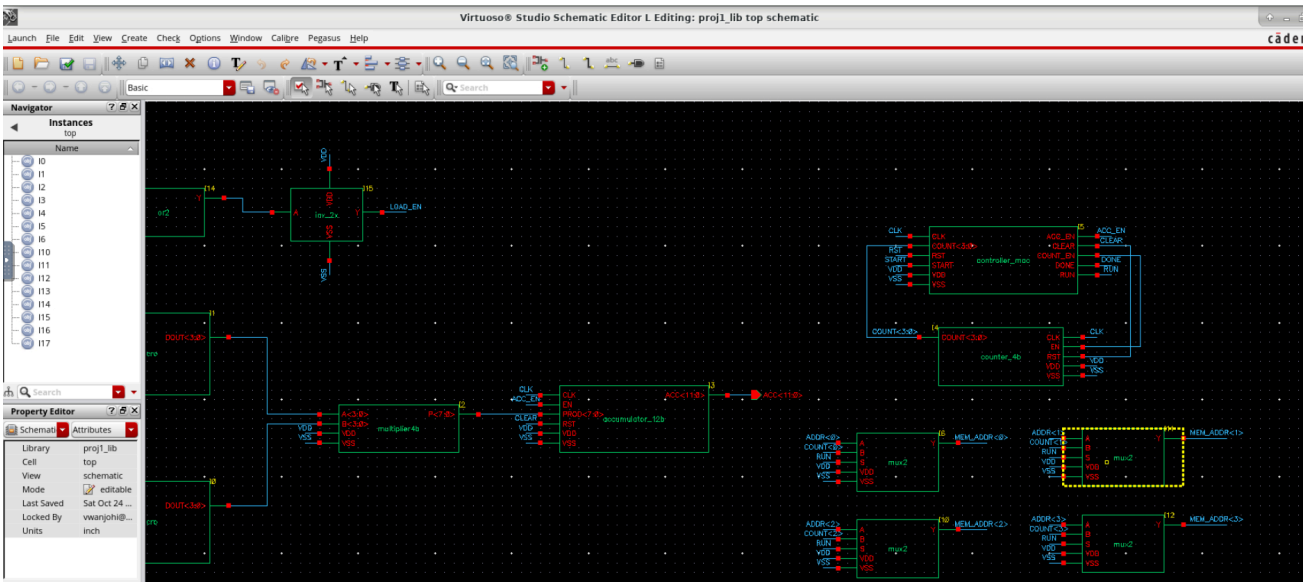


Figure 3: `top`, right half: `controller_mac` (I5) and `counter_4b` (I4) in a loop (COUNT_EN, CLEAR, COUNT⟨3:0⟩), and four `mux2` that send COUNT to MEM_ADDR while RUN = 1 and the external ADDR otherwise.

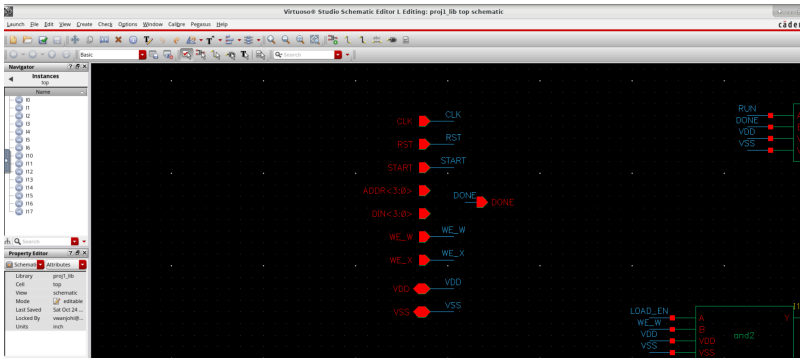


Figure 4: `top` ports: CLK, RST, START, ADDR⟨3:0⟩, DIN⟨3:0⟩, WE_W, WE_X in; DONE and ACC⟨11:0⟩ out; VDD, VSS.

1.2 Block inventory

Block	Built from	Test bench	Status
Register bit (DFF)	latch pair, T-gates	`tb_dff`, `tb_dff_cclk`	$t_{cq} = 68.51/62.18$ ps; $t_{su} = 40$ ps; $t_h = 10/\leq 0$ ps; $C_{CLK} = 2.822$ fF; $E = 3.895/9.309$ fJ
4×4 multiplier	AND array, adders	`tb_multiplier4b`	256/256 correct; worst case 1.14–1.24 ns
12-bit adder / acc.	`fulladder_12b`, `register_12b`	`tb_fulladder_12b` `tb_accumulator_12b`	adder sums match; acc. reaches 0xE10
4-bit counter	4 DFFs, toggle chain	`tb_counter4b`	38/38 vector checks pass
Controller	4 registers, gates	`tb_controller_mac`	60/60 vector checks; DONE 17 edges after START
MAC top	all of the above and 2 macros	`tb_top`	A/B/C pass at 10/4/2.5 ns; at 2.0 ns A/C pass, B = 0xC10 fail; all fail at 1.75 ns and 1.5 ns

2. Sequential elements

2.1 Style and justification

Positive-edge master-slave flip-flop built from two static transmission-gate latches (`dlatch`): the master is transparent while CLK = 0, the slave while CLK = 1, so D is captured on the rising edge. Each latch has an input `tg`, a storage node driven by two `inv_2x`, a hold `tg` that closes the loop from Q, and its own clock inverter; the flip-flop adds one more `inv_2x` to make the master's clock (100 fins: `tg` 2+2 fins, `inv_2x` 8/4 fins). It was chosen because it reuses the verified `tg` and inverter cells, is fully static (holds with the clock stopped, unlike a dynamic C^{2}MOS latch), and needs only the single external clock. Its cost is the local clock inverters (clock load, and a small skew between the two phases, which hold must cover). `register_1b` adds a `mux2` load enable ($EN = 1$ loads D, $EN = 0$ recirculates Q) and an `and2` synchronous reset (152 fins). A C^{2}MOS or pulsed latch would cut the local clock load ($C_{CLK} = 2.822$ fF) and some of the 68.51 ps t_{cq}, but cannot hold with the clock stopped, which the sequencer does in IDLE and DONE.

Design note. The first `dlatch` fed the node after one inversion back through the hold gate, an inverting loop that cannot store a value; the hold `tg` now returns Q (two inversions) to the storage node and conducts while CLK = 0 (fixed 10/25, before any register simulation).

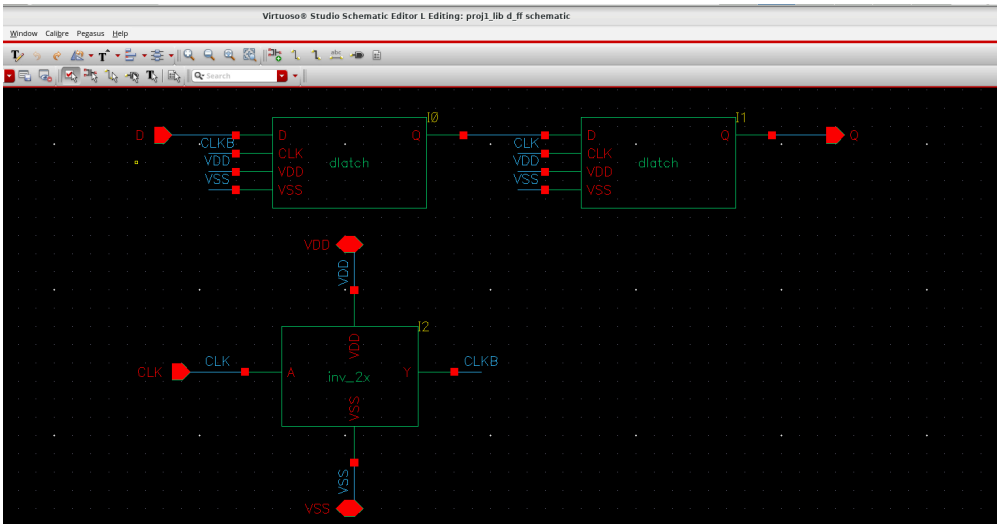


Figure 5: `d_ff`: master `dlatch` I0 clocked by CLKB (transparent while CLK = 0), slave I1 clocked by CLK, and the clock inverter I2 (`inv_2x`, 8/4 fins). Each `dlatch`: input and hold `tg` (2+2 fins), two `inv_2x` in the loop and one for its local clock; 100 fins in total.

Test bench (`tb_dff`). DUT is `d_ff`. CLK and D are driven from `dff_active.vec`; VDD is 0.8 V (V0); Q is loaded by $C_Q = 1$ fF. Each data value is stable for a full cycle before the capturing rising CLK edge.

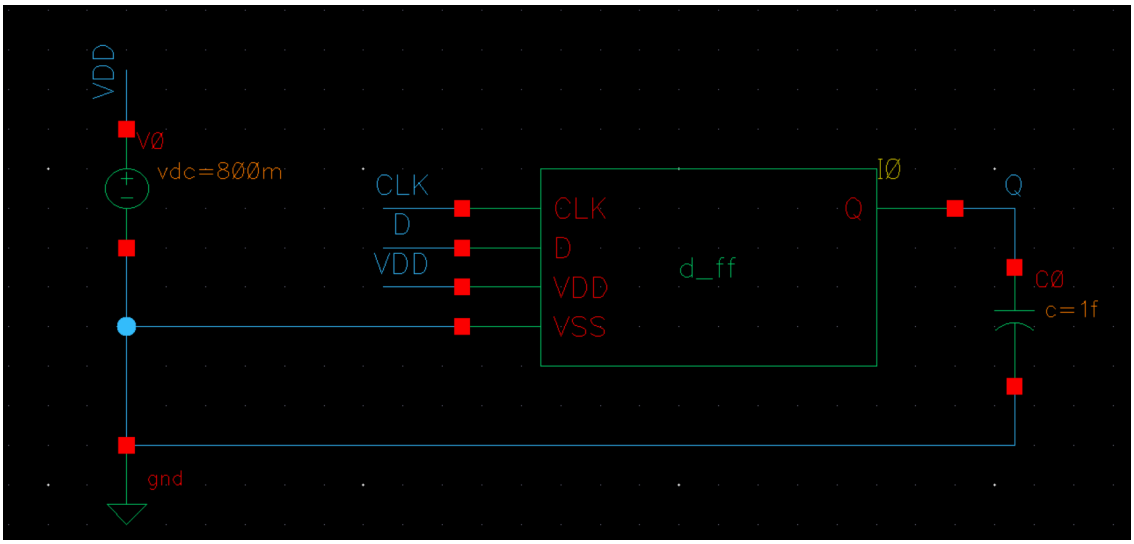


Figure 6: `tb_dff`: `d_ff` with CLK and D from the vector file, $V_{DD} = 0.8$ V (V0), $C_Q = 1$ fF.

The 0–50 ns transient is a functional check: Q tracks D on every rising CLK. C_{CLK} and energy per clock are not in this run.

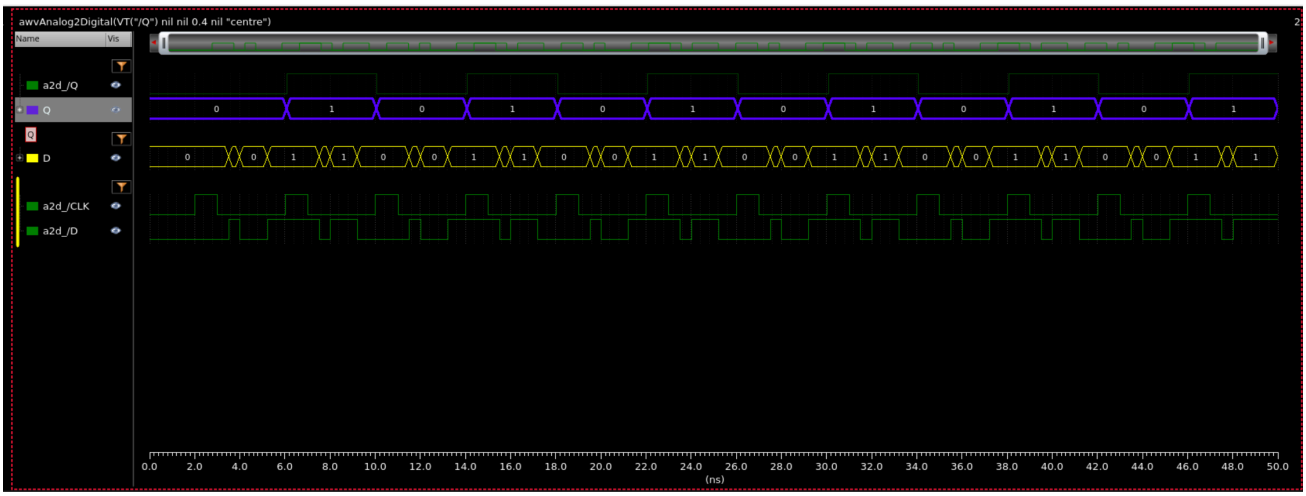


Figure 7: `tb_dff` with `dff_active.vec`, 0–50 ns: Q (bus) follows D on the rising CLK. Positive-edge capture.

2.2 Clock-to-Q

Q rising. Markers in ViVA at the $V_{DD}/2 = 0.4$ V threshold on the capturing CLK edge and the subsequent Q rise: CLK (A) at 6.010000 ns, Q (B) at 6.078507 ns, dx = 68.50691 ps. D is held high through this edge.

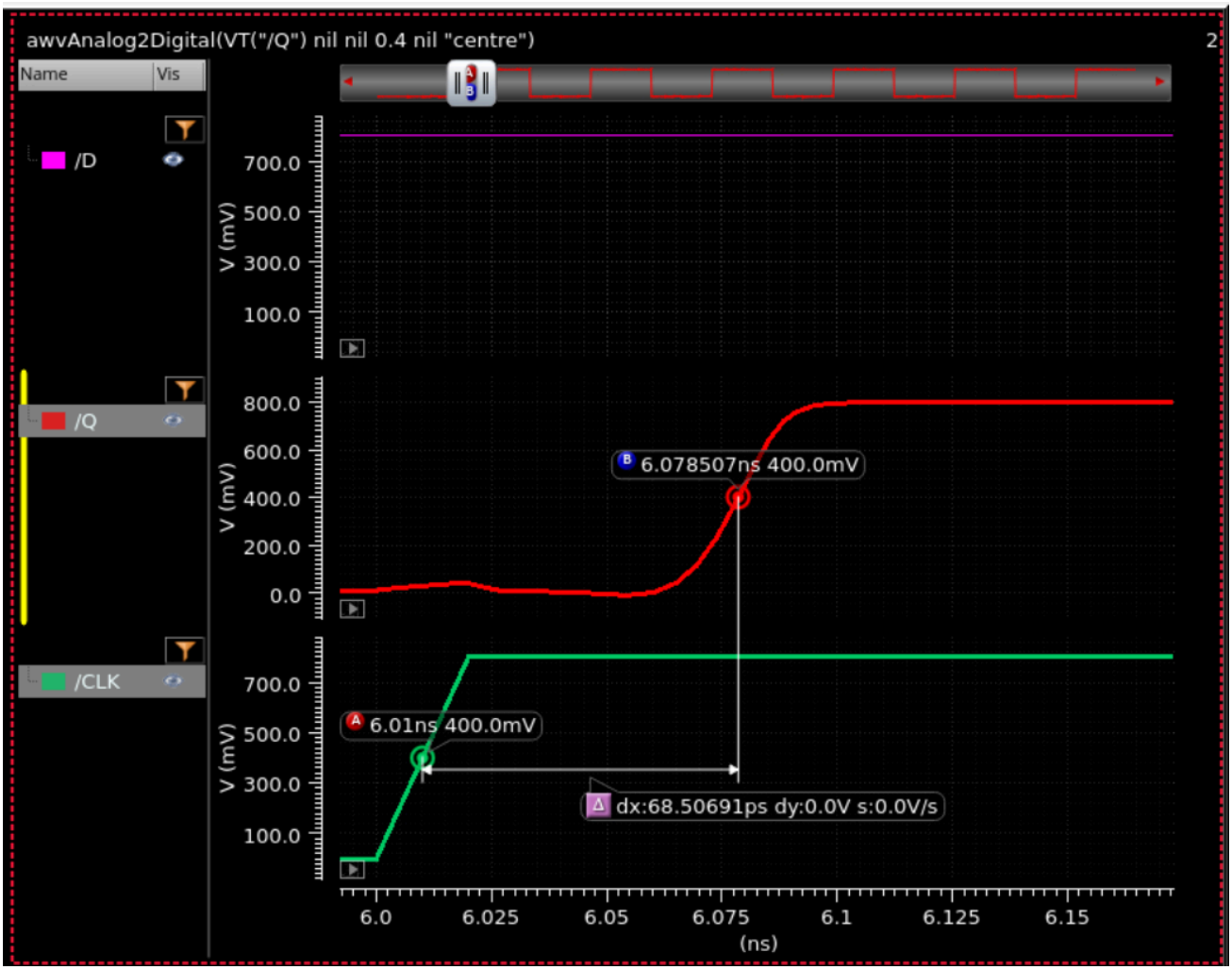


Figure 8: `tb_dff` rising clock-to-Q at 0.4 V: CLK (green, A) at 6.010 ns, 400 mV; Q (red, B) at 6.078507 ns, 400 mV; dx = 68.51 ps. D (magenta) held high. $V_{DD} = 0.8$ V, $C_Q = 1$ fF.

Clock-to-Q, Q falling. Same 0.4 V markers on `dff_setup_fall.vec` at the 8 ns edge (D falling with ≈ 1 ns of slack): CLK (A) at 8.01 ns, Q (B) at 8.07218 ns, dx = 62.17548 ps. $V_{DD} = 0.8$ V, $C_Q = 1$ fF.

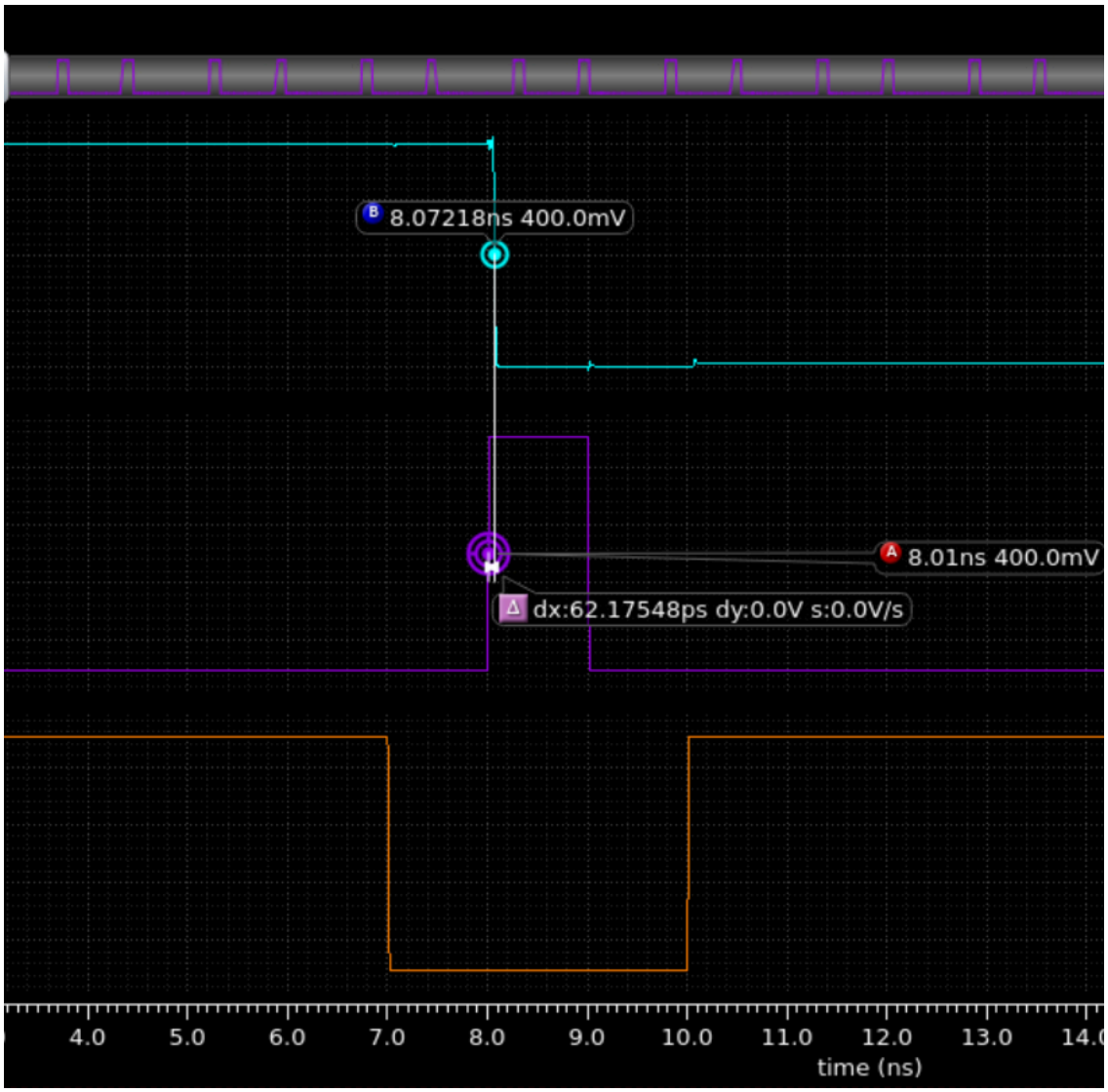


Figure 9: Falling clock-to-Q at 0.4 V: CLK (A) at 8.01 ns, 400 mV; Q (B) at 8.07218 ns, 400 mV; dx = 62.18 ps. Unconstrained slack (≈ 1 ns).

The earlier 0–15 ns close-up ($t_{cq} \approx 100$ ps at the 3 ns / 7 ns edges) was a coarse read of the same bench and is superseded by the 0.4 V marker measurements.

2.3 Setup time

D rising (`dff_setup_rise.vec`). t_{su} is the smallest D-to-CLK slack at which Q is still captured and t_{cq} does not grow more than 10% above the unconstrained value: 1.10×68.51 ps = 75.36 ps.

Setup margin	CLK / Q at 0.4 V	t_{cq}	vs. 75.36 ps wall
40 ps	106.01 ns / 106.08143 ns	71.43 ps (+4.3%)	pass
30 ps	120.01 ns / 120.08817 ns	78.17 ps (+14%)	fail
20 ps	134.01 ns / 134.09757 ns	87.57 ps (+28%)	fail
10 ps	148.01 ns / 148.1109 ns	100.90 ps (+47%)	fail

Q still rises on every trial (the bit is captured); the wall is delay, not function. The smallest passing slack is 40 ps, so $t_{su,\text{rise}} = 40$ ps.

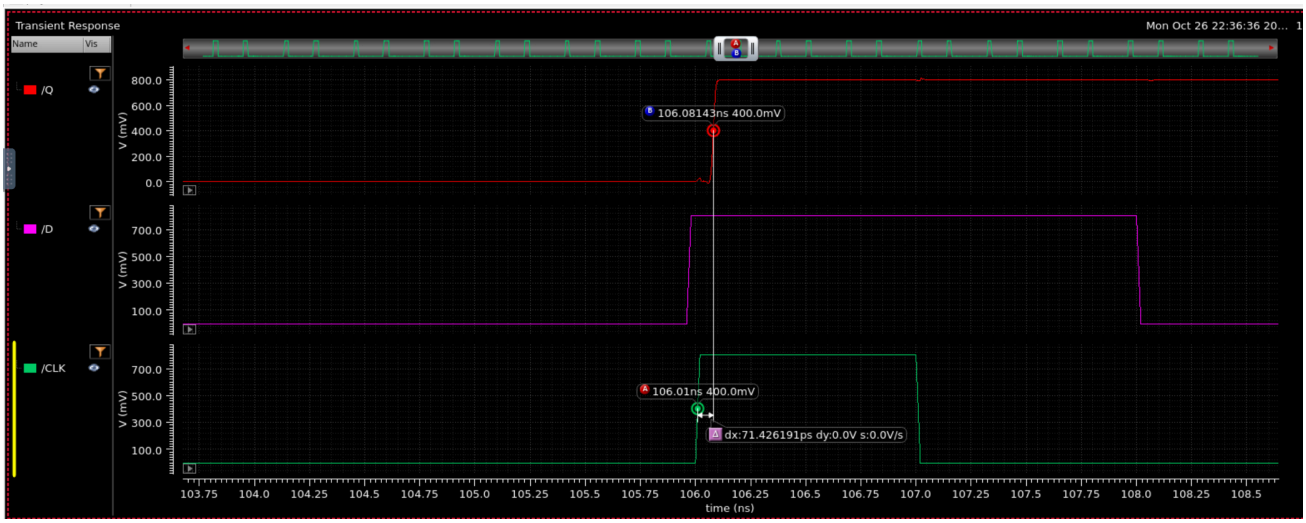


Figure 10: `dff_setup_rise.vec`, 40 ps setup: CLK (green, A) at 106.01 ns, 400 mV; Q (red, B) at 106.08143 ns, 400 mV; d$x = 71.43$ ps. 4.3% slower than 68.51 ps; passes the 75.36 ps wall.

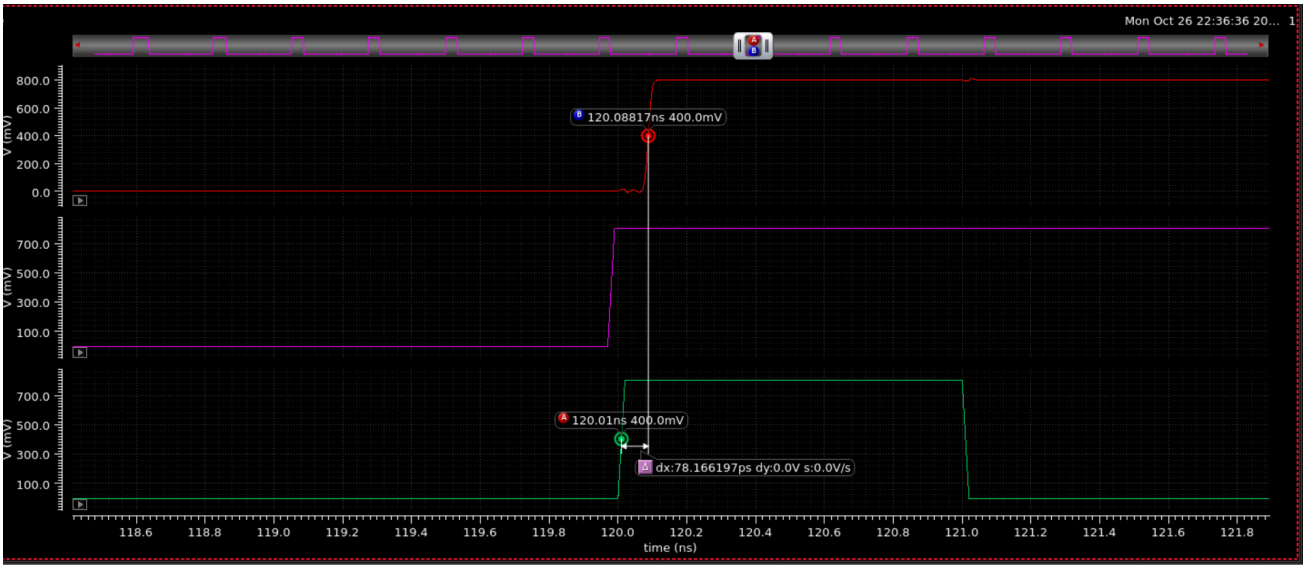


Figure 11: `dff_setup_rise.vec`, 30 ps setup: CLK (A) at 120.01 ns, 400 mV; Q (B) at 120.08817 ns, 400 mV; d$x = 78.17$ ps. Fails the 10% wall.

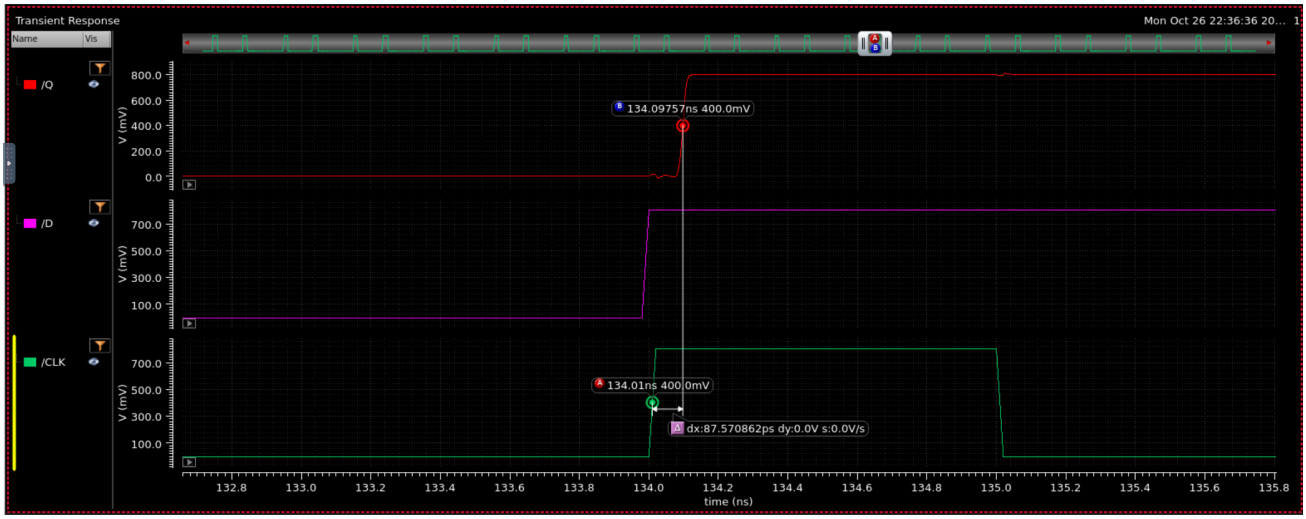


Figure 12: `dff_setup_rise.vec`, 20 ps setup: CLK (green, A) at 134.01 ns, 400 mV; Q (red, B) at 134.09757 ns, 400 mV; d$x = 87.57$ ps. Fails the 10% wall.

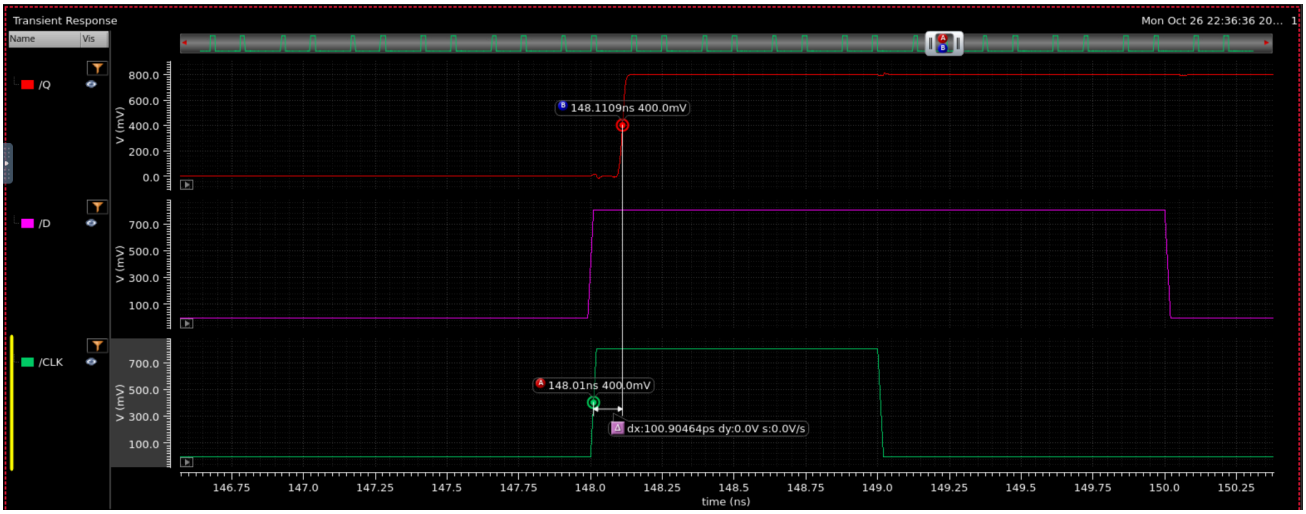


Figure 13: `dff_setup_rise.vec`, 10 ps setup: CLK (green, A) at 148.01 ns, 400 mV; Q (red, B) at 148.1109 ns, 400 mV; d$x = 100.90$ ps. Captured, but t_{cq} fails the 75.36 ps wall.

Setup time, D falling (`dff_setup_fall.vec`). Same 10% wall on the falling t_{cq}: $1.10 \times 62.18\,\text{ps} = 68.39$ ps.

Setup margin	CLK / Q at 0.4 V	t_{cq}	vs. 68.39 ps wall
40 ps	106.01 ns / 106.0758 ns	65.80 ps (+5.8%)	pass
30 ps	120.01 ns / 120.08197 ns	71.97 ps (+16%)	fail
20 ps	134.01 ns / 134.09081 ns	80.81 ps (+30%)	fail
10 ps	148.01 ns / 148.11657 ns	106.57 ps (+71%)	fail

Every trial captures the falling D; $t_{su,\text{fall}} = 40$ ps, the smallest passing slack. Combined with the rising sweep, both edges sit in $(30, 40]$ ps, so the number used for timing is $t_{su} = 40$ ps.

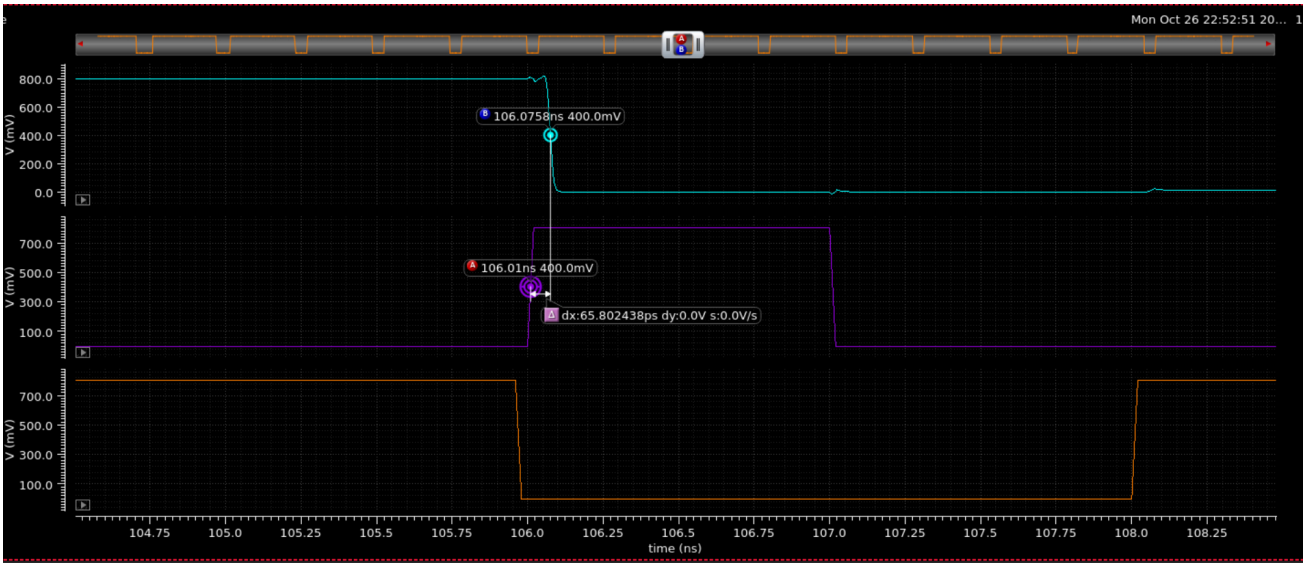


Figure 14: `dff_setup_fall.vec`, 40 ps setup: CLK (A) at 106.01 ns, 400 mV; Q (B) at 106.0758 ns, 400 mV; d$x = 65.80$ ps. Passes the 68.39 ps wall.

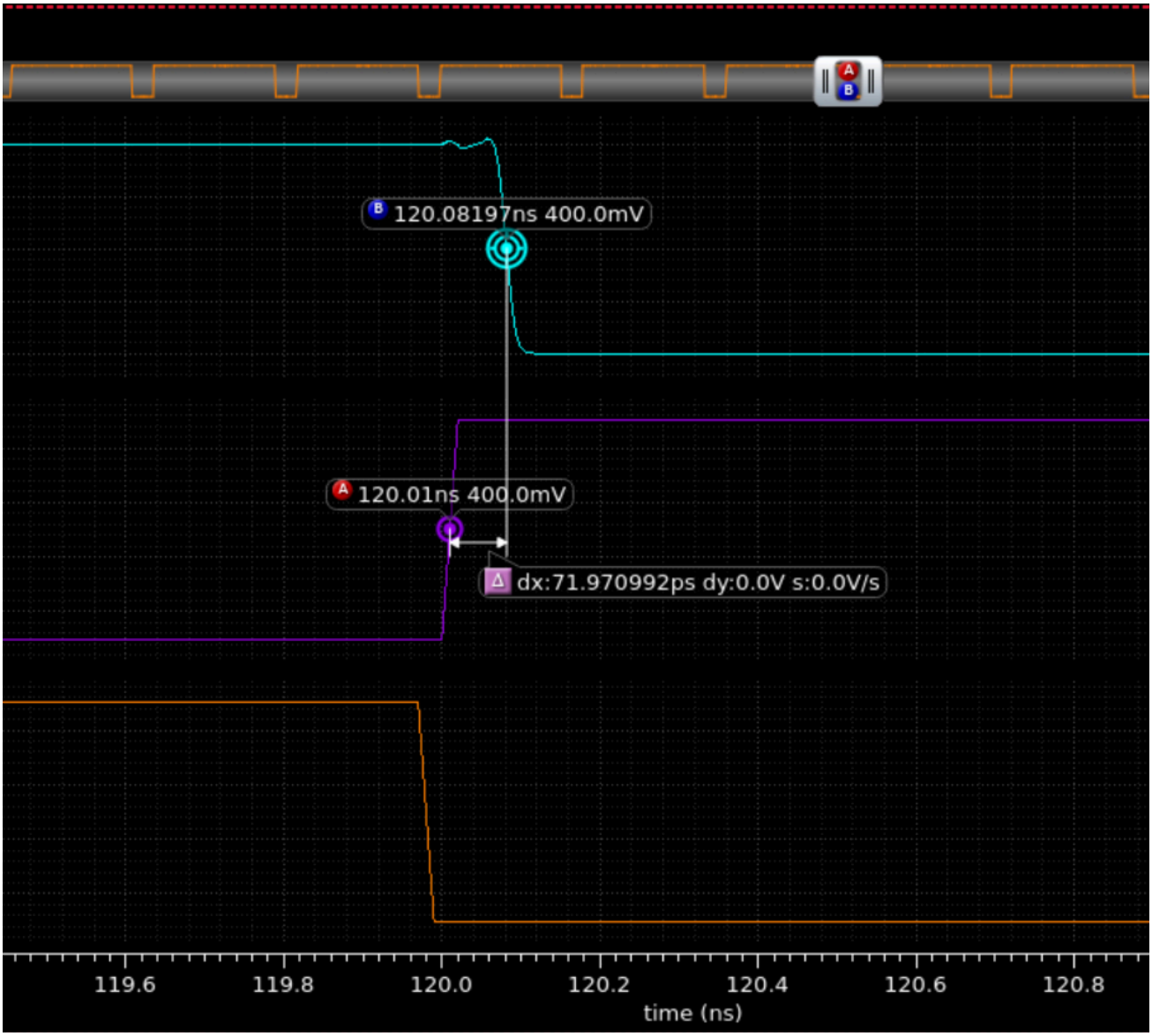


Figure 15: `dff_setup_fall.vec`, 30 ps setup: CLK (A) at 120.01 ns, 400 mV; Q (B) at 120.08197 ns, 400 mV; d$x = 71.97$ ps. Fails the 10% wall.

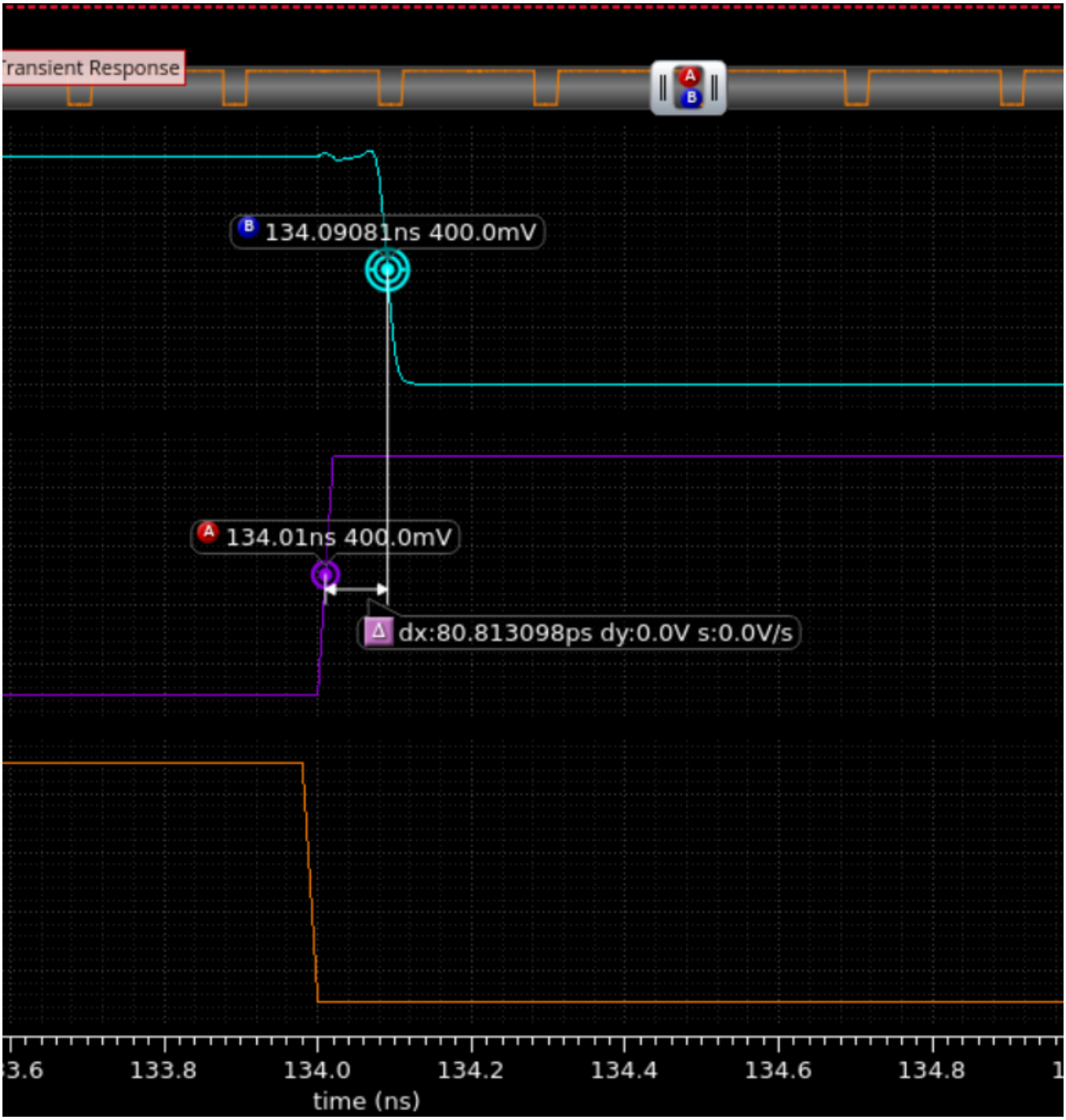


Figure 16: `dff_setup_fall.vec`, 20 ps setup: CLK (A) at 134.01 ns, 400 mV; Q (B) at 134.09081 ns, 400 mV; d$x = 80.81$ ps. Fails the 10% wall.

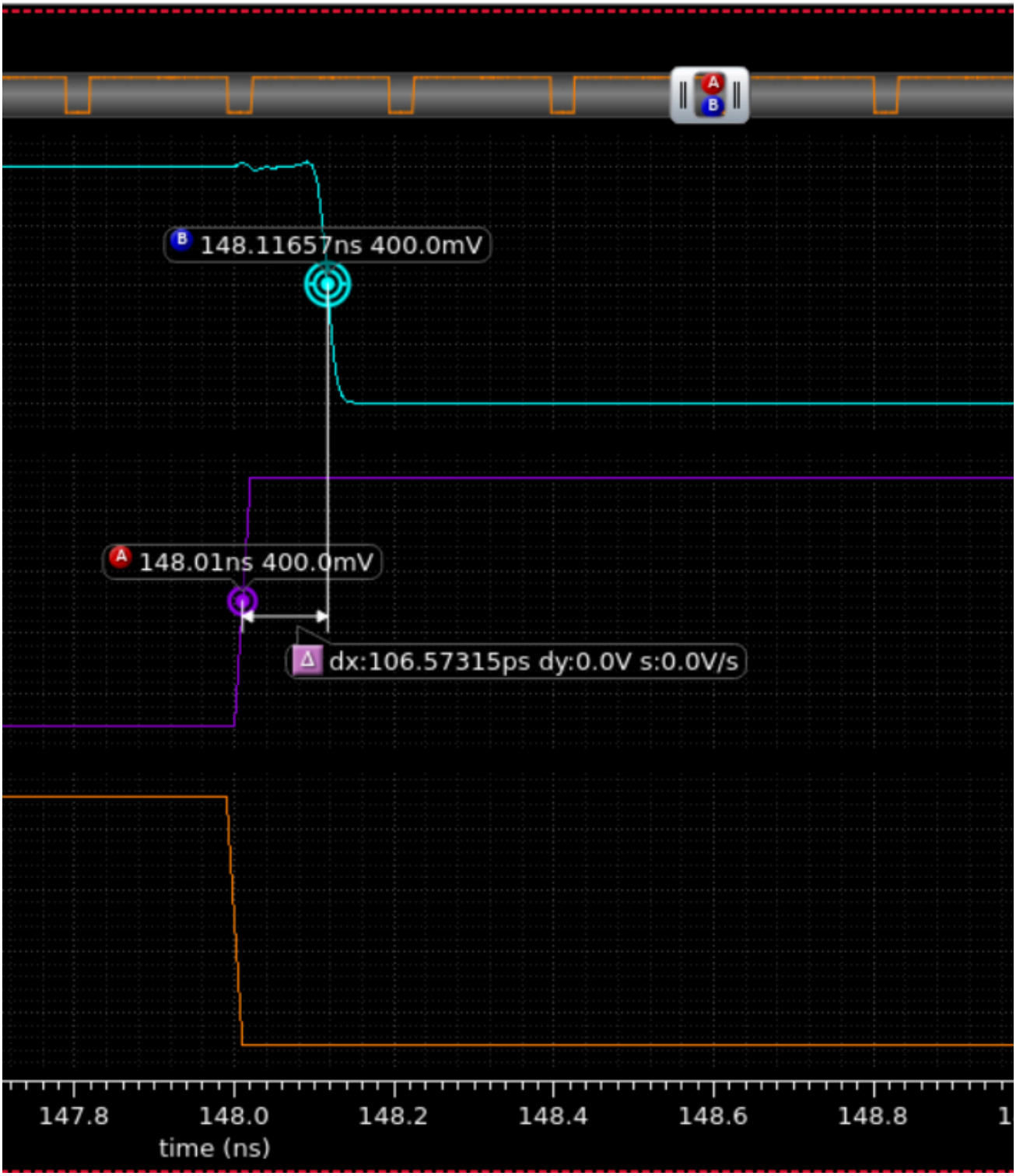


Figure 17: `dff_setup_fall.vec`, 10 ps setup: CLK (A) at 148.01 ns, 400 mV; Q (B) at 148.11657 ns, 400 mV; dx = 106.57 ps. Fails the 10% wall.

Setup margin	D rising t_{cq}	D falling t_{cq}
baseline	68.51 ps	62.18 ps
40 ps	71.43 ps (pass)	65.80 ps (pass)
30 ps	78.17 ps (fail)	71.97 ps (fail)
20 ps	87.57 ps (fail)	80.81 ps (fail)
10 ps	100.90 ps (fail)	106.57 ps (fail)

2.4 Hold time

D rising (`dff_hold_rise.vec`). D is LOW at the capturing CLK edge and rises after it; Q should stay LOW (the value that existed at the edge). Criterion is functional: Q must remain the captured value, not a 10% t_{cq} wall.

Hold margin	CLK edge	Observed Q	Result
0 ps	8 ns	stays HIGH; dip to ≈ 0.48 V	fail
10 ps	22 ns	falls to 0 V and stays LOW	pass
20–1000 ps	later edges	captures LOW	pass

The rising-data hold wall is in $(0, 10]$ ps. The number used for timing is $t_{h,\text{rise}} = 10$ ps, the smallest passing slack.

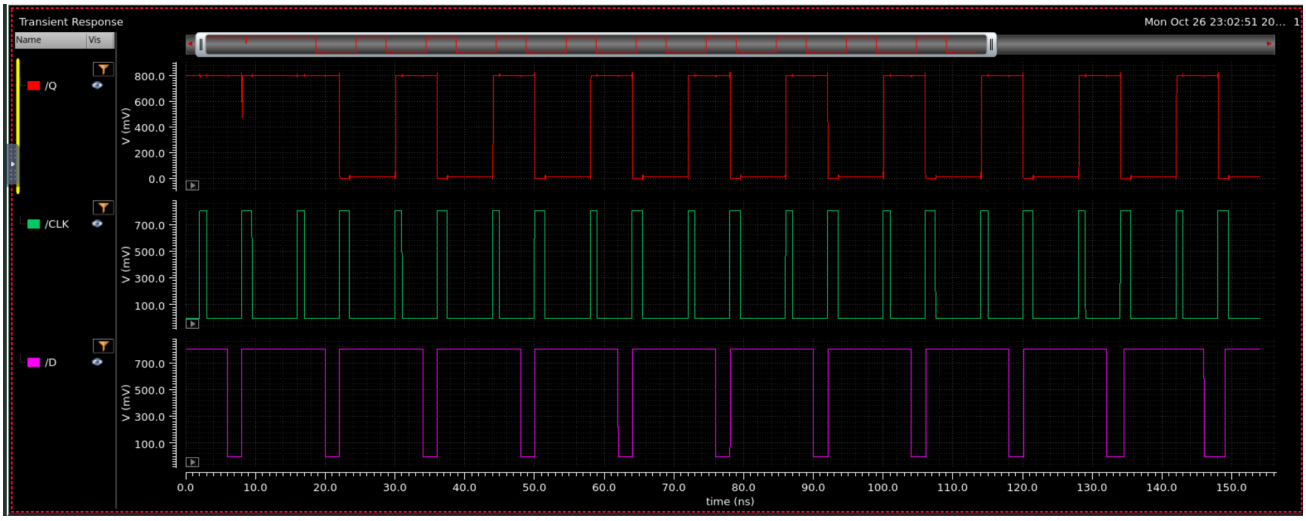


Figure 18: `dff_hold_rise.vec`, 0–154 ns: Q (red), CLK (green), D (magenta). D rises after each capturing CLK. First edge (8 ns, 0 ps hold) Q stays high; from 22 ns (10 ps) onward Q captures LOW.

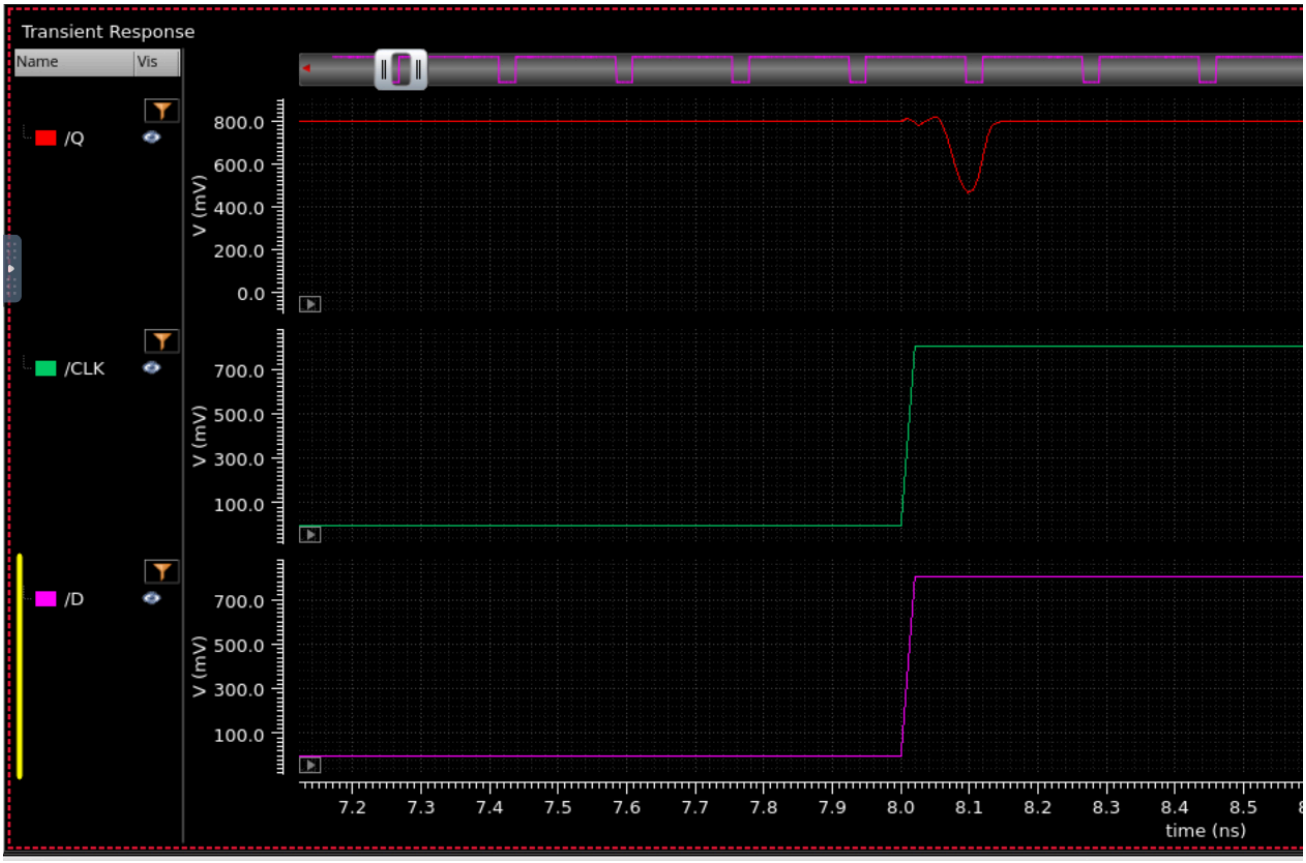


Figure 19: 0 ps hold at 8 ns: D and CLK rise together. Q dips to ≈ 0.48 V then returns HIGH (expected LOW). Fail.



Figure 20: 10 ps hold at 22 ns: D rises just after CLK. Q falls to 0 V and stays LOW. Pass.

Hold time, D falling (`dff_hold_fall.vec`). D is HIGH at the capturing CLK edge and falls after it; Q should stay HIGH. At 0 ps (8 ns) Q rises to 0.8 V after t_{cq} and remains HIGH: pass. The 10 ps edge (22 ns) and later trials (20–1000 ps) also capture HIGH. So $t_{h,\text{fall}} \leq 0$ ps; finding a failing (negative) hold would need D to fall before CLK. Combined hold used for timing is the worse edge: $t_h = 10$ ps.



Figure 21: `dff_hold_fall.vec`, 0 ps hold at 8 ns: CLK (purple) rises as D (orange) falls. Q (cyan) rises to 0.8 V near 8.1 ns and stays HIGH (expected HIGH). Pass.

2.5 Energy per clock

$V_{DD} = 0.8$ V, $C_Q = 1$ fF, 20 clocks from 10 ns to 90 ns. Energy drawn from the VDD source (external CLK-driver energy excluded):

$$E = \frac{-V_{DD}}{20} \int_{10\,\text{ns}}^{90\,\text{ns}} I_{V0+}\, \mathrm{d}t.$$

Idle (`dff_energy_idle.vec`, D constant): 3.895 fJ/clock. Toggling (`dff_energy_toggle.vec`): 9.309 fJ/clock. Extra energy due to data switching: $9.309 - 3.895 = 5.414$ fJ/clock.

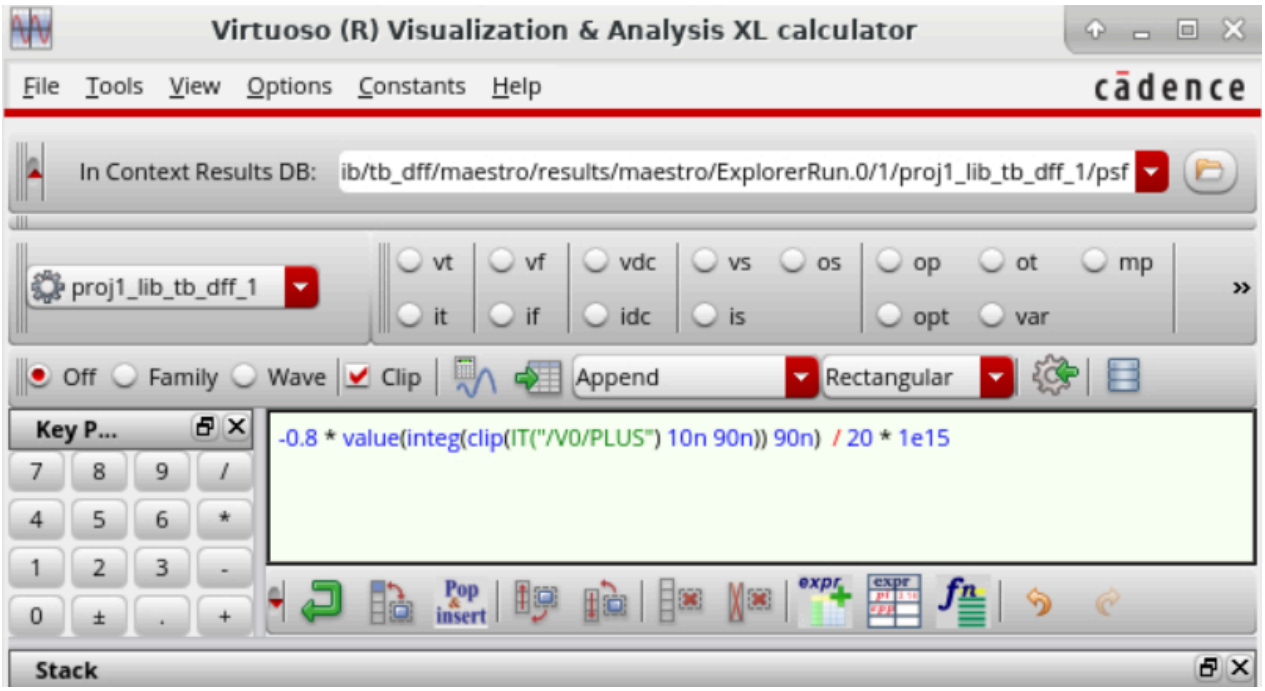


Figure 22: ViVA calculator: $-0.8 \times \text{value}(\text{integ}(\text{clip}(I_{V0+}, 10\,\text{ns}, 90\,\text{ns})), 90\,\text{ns})/20 \times 10^{15}$. Same expression for idle and toggle.

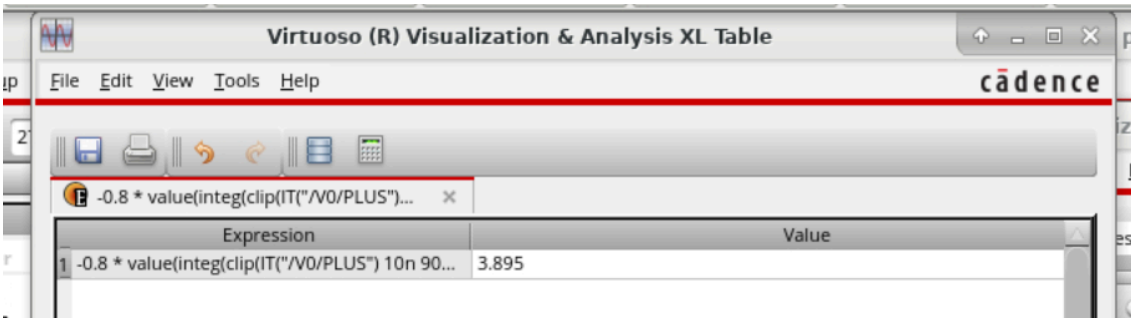


Figure 23: Idle: 3.895 fJ/clock. D constant, $V_{DD} = 0.8$ V, $C_Q = 1$ fF, 20 clocks.

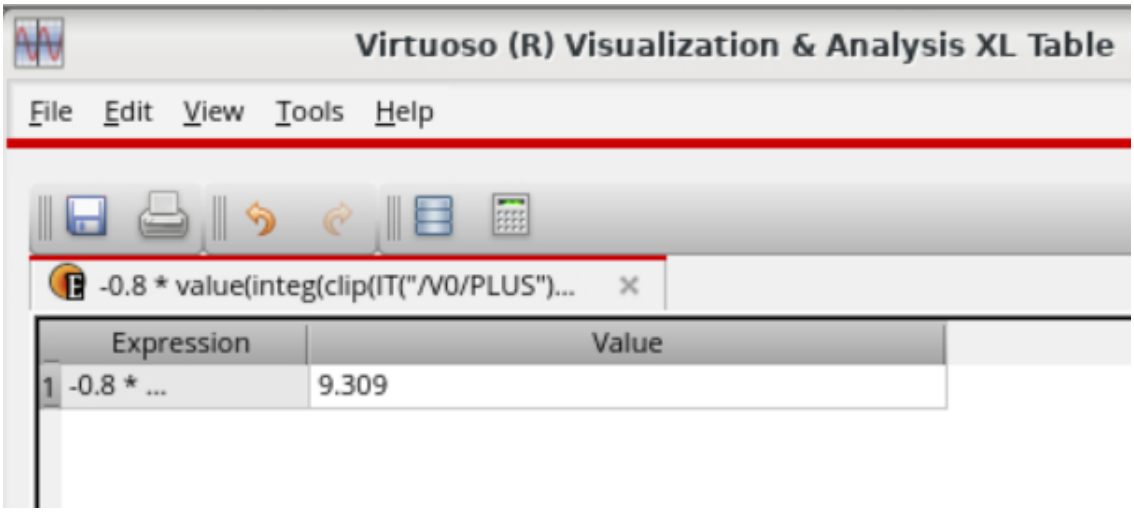


Figure 24: Toggle: 9.309 fJ/clock. Same window and load. Extra switching energy 5.414 fJ/clock.

2.6 Clock-input capacitance

On `tb_dff_cclk`, CLK is driven by `vpulse` V1 ($0 \to 0.8$ V, $t_r = 20$ ps); VDD is 0.8 V; $C_Q = 1$ fF. Effective rising-edge capacitance from the charge delivered by the CLK source:

$$C_{CLK} = -\int I_{V1+}\,\mathrm{d}t \;/\; V_{DD} = 2.822\,\mathrm{fF}.$$

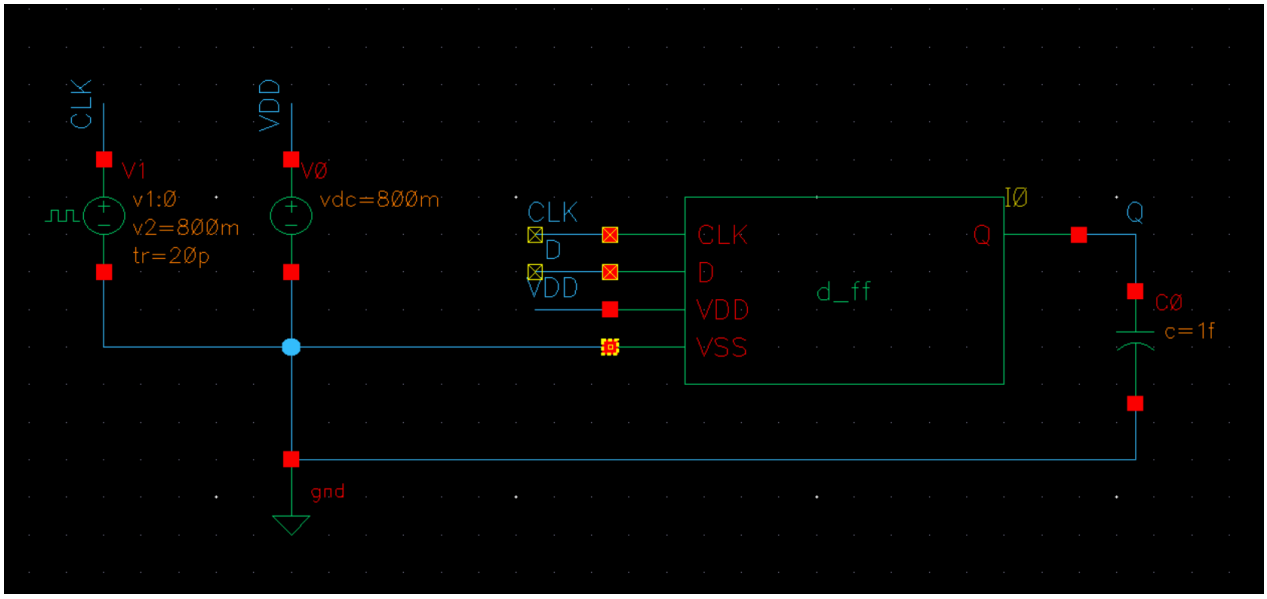


Figure 25: `tb_dff_cclk`: `d_ff` with CLK from V1 (`vpulse`, $0 \to 800$ mV, $t_r = 20$ ps), $V_{DD} = 0.8$ V (V0), $C_Q = 1$ fF.

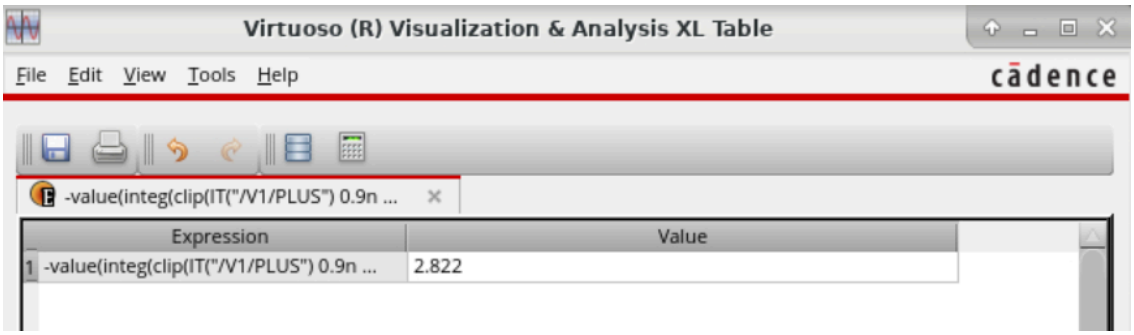


Figure 26: ViVA table: $-\text{integ}(\text{clip}(I_{V1+}, 0.9\,\text{ns}, \ldots)) = 2.822$ fF. Rising-edge C_{CLK} at $V_{DD} = 0.8$ V.

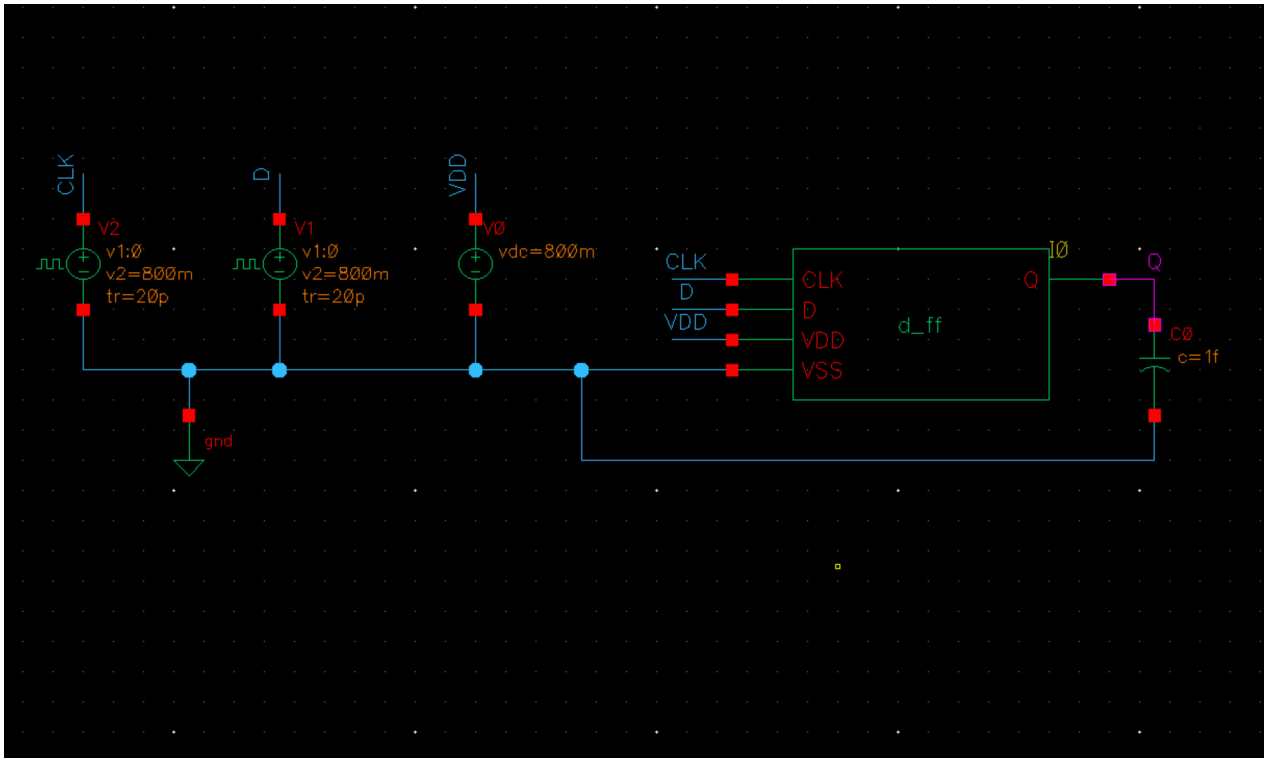


Figure 27: `tb_dff` (pulse-source view): `d_ff` driven by CLK (V2) and D (V1), $V_{DD} = 0.8$ V, $C_Q = 1$ fF, 20 ps edges.

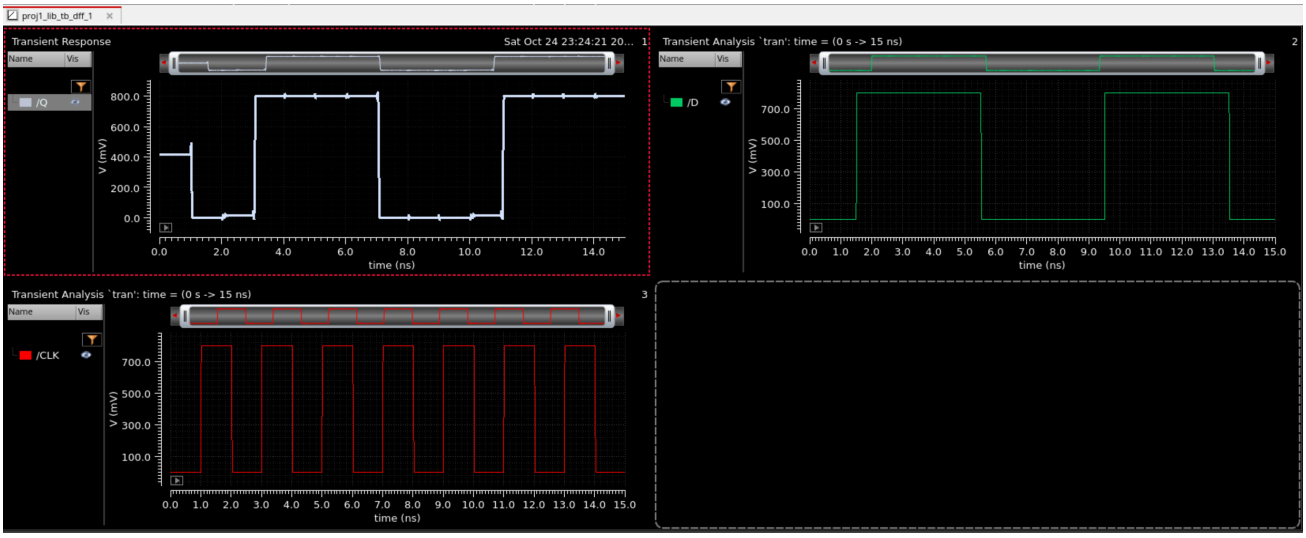


Figure 28: `tb_dff` transient, 0–15 ns: Q (left), D (right), CLK (bottom). Positive-edge capture; $T_{CLK} = 2$ ns.

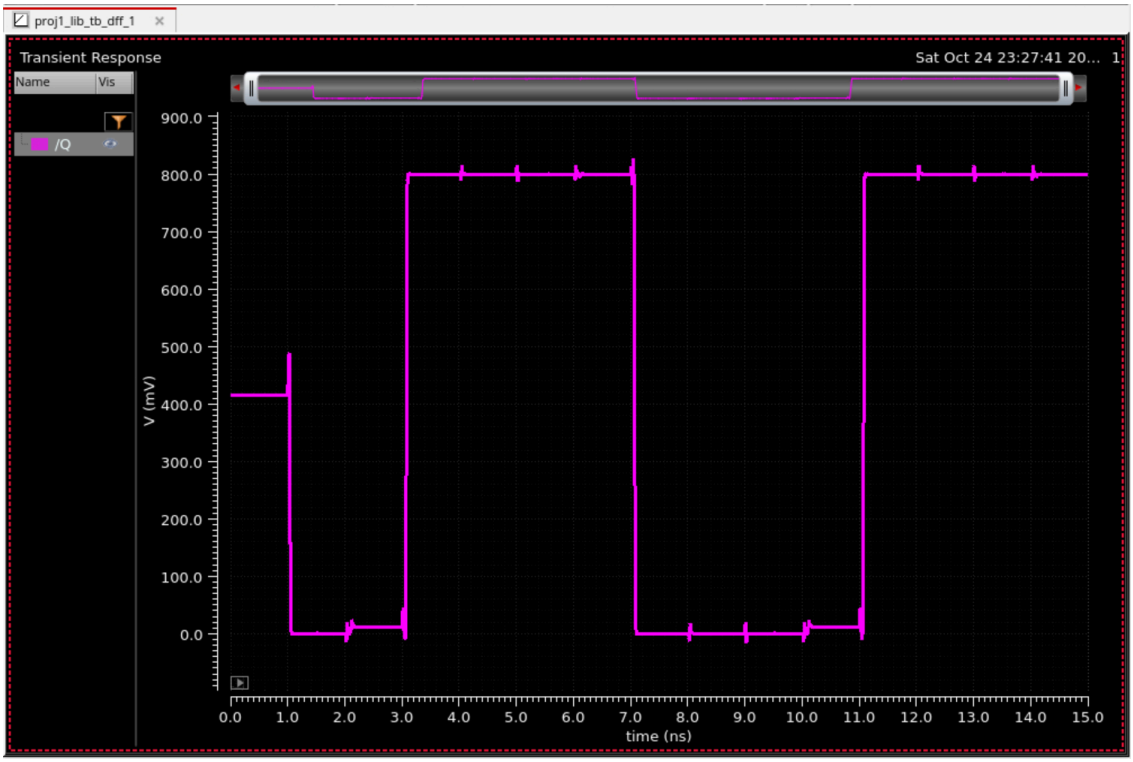


Figure 29: Earlier Q close-up at the 3.1 ns (rise) and 7.1 ns (fall) edges. Coarse $t_{cq} \approx 100$ ps; replaced by the 68.51 ps marker measurement above. Glitches at integer nanoseconds are clock feedthrough on the slave.

2.7 Enabled registers

`register_1b` wraps the flip-flop with a `mux2` load enable and a synchronous reset. `tb_register_1b` drives CLK, D, EN and RST from four `vpulse` sources (0.8 V, 20 ps edges).

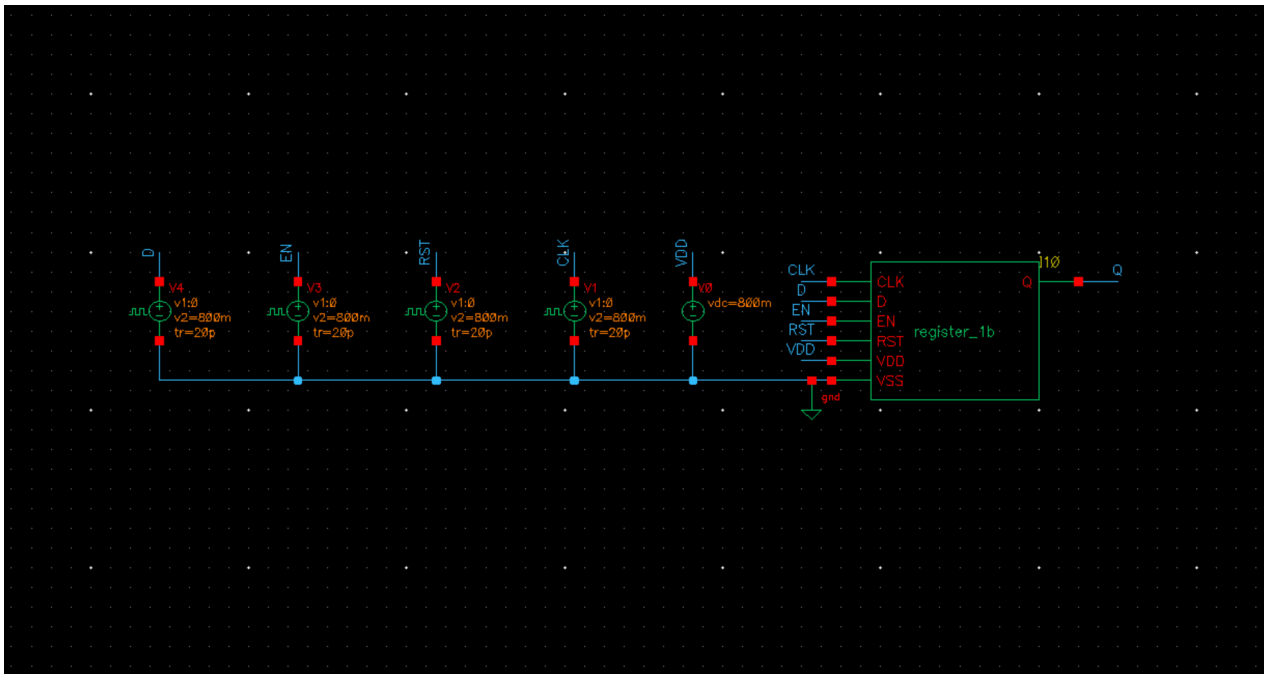


Figure 30: `tb_register_1b`: `register_1b` with CLK (V1), RST (V2), EN (V3), D (V4), $V_{DD} = 0.8$ V.

RST is active-low: it sits high except for a low pulse near 10 ns. EN is high 0–6.5 ns (load), low 6.5–12 ns (hold / recirculate), high 12–16 ns (load). Q follows D while EN = 1, holds 1 through the EN = 0 window until the RST

pulse, then clears on the next rising CLK (≈ 11 ns). After RST releases, EN returns high and Q loads again (13 ns rise, 15 ns fall).



Figure 31: `tb_register_1b`, 0–20 ns: VDD, RST, mux output, Q, internal AND, EN. Hold while EN = 0; Q clears on the CLK after RST goes low.

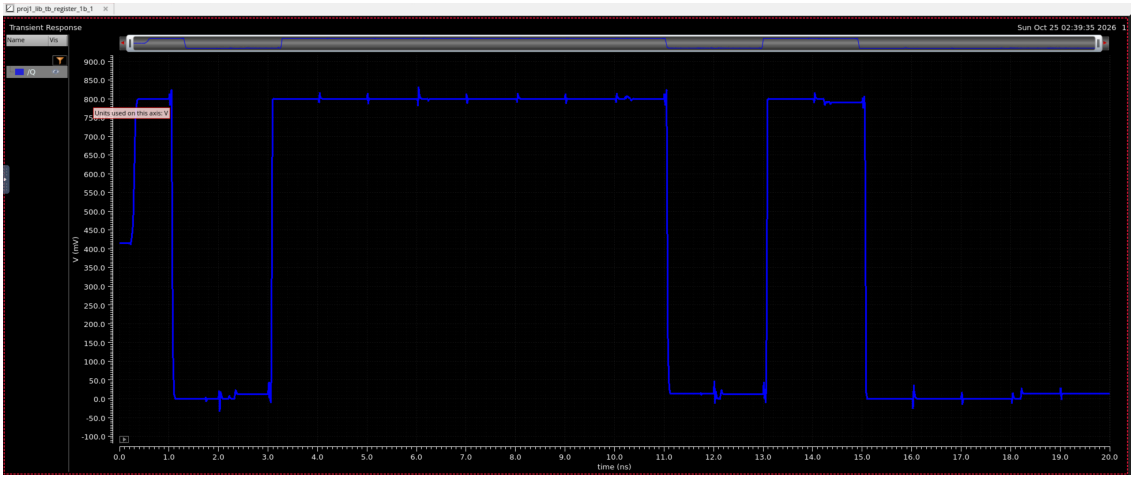


Figure 32: Q close-up: load (3.1 ns), hold through EN = 0, reset at 11.1 ns, reload at 13.1 ns, D = 0 captured at 15.1 ns.

`register_12b` is twelve `register_1b` in parallel (accumulator word). `tb_register_12b` drives the 12-bit D bus, CLK, EN and RST at $V_{DD} = 0.8$ V.

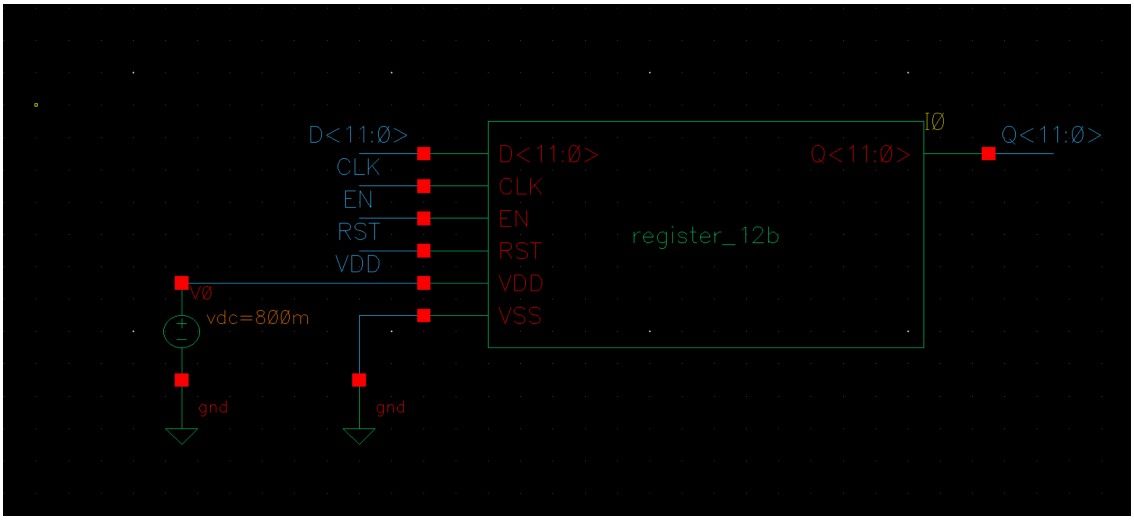


Figure 33: `tb_register_12b`: `register_12b` with D⟨11:0⟩, CLK, EN, RST; Q⟨11:0⟩ out.

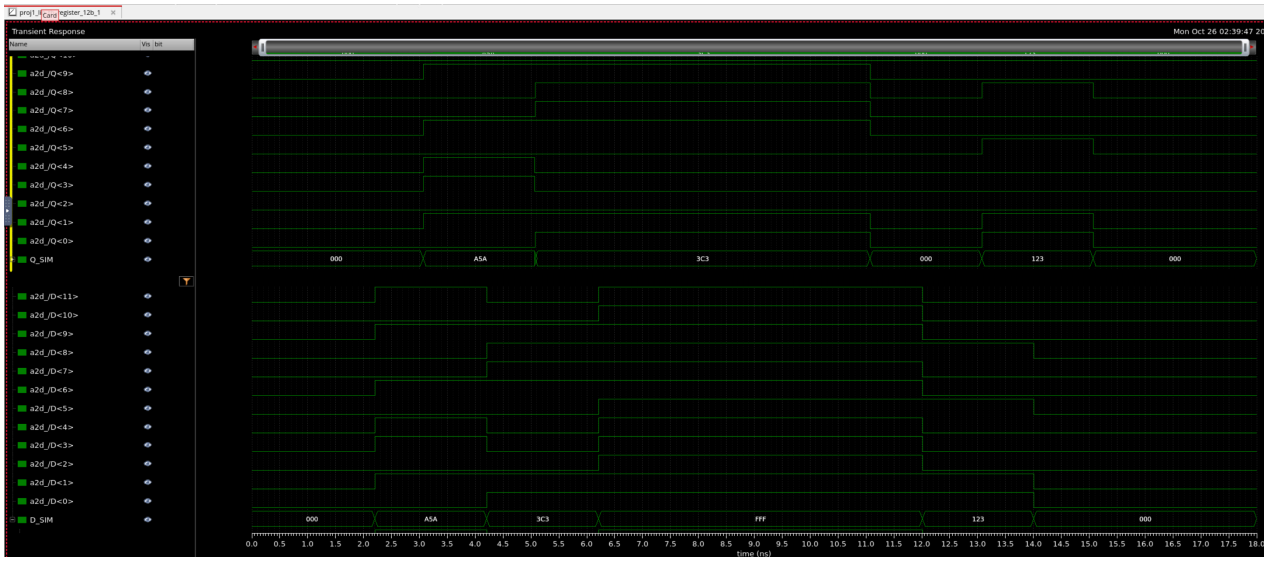


Figure 34: `tb_register_12b`: Q follows D on the rising CLK (000 →A5A→3C3→ 123). D =`FFF` while EN/RST holds Q, then Q returns to 000.

Quantity	Value	Determined by
Setup time t_{su} (D 0→1)	40 ps	10% t_{cq} wall at 75.36 ps; 40 ps: 71.43 ps (pass); 30/20/10 ps fail
Setup time t_{su} (D 1→0)	40 ps	10% t_{cq} wall at 68.39 ps; 40 ps: 65.80 ps (pass); 30/20/10 ps fail
Hold time t_h (D 0→1 after CLK)	10 ps	functional: 0 ps fail (Q stays HIGH), 10 ps pass (Q stays LOW)
Hold time t_h (D 1→0 after CLK)	≤ 0 ps	0 ps at 8 ns: Q rises to 0.8 V and stays HIGH (pass)
Clock-to-Q t_{cq} (rise)	68.51 ps	CLK 0.4 V at 6.010 ns to Q 0.4 V at 6.078507 ns, $C_Q = 1$ fF, $V_{DD} = 0.8$ V
Clock-to-Q t_{cq} (fall)	62.18 ps	CLK 0.4 V at 8.01 ns to Q 0.4 V at 8.07218 ns, $C_Q = 1$ fF, $V_{DD} = 0.8$ V
Clock-input capacitance C_{CLK}	2.822 fF	rising-edge $\int I_{\mathrm{CLK}}\,\mathrm{d}t/V_{DD}$ on `tb_dff_cclk`, $t_r = 20$ ps
Energy per clock (idle, D constant)	3.895 fJ	$-V_{DD}\int I_{V0+}\,\mathrm{d}t$ over 20 clocks (10–90 ns), $C_Q = 1$ fF
Energy per clock (D toggling)	9.309 fJ	same integral on `dff_energy_toggle.vec`
Extra energy from data switching	5.414 fJ	9.309 − 3.895; CLK-driver energy excluded

3. Datapath

3.1 Leaf cells

Static CMOS throughout, $L = 18$ nm: `tg` (2+2 fins), `mux2` (two `tg` and an `inv_2x`, $Y = S\,?\,B : A$), `xor2` (complementary static gate with input inverters, 44 fins), `and2` (`nand2_u`+`inv_2x`), `or2` (NOR2+`inv_2x`). The full adder is $S = (A \oplus B) \oplus C_{in}$, $C_{out} = C_{in}(A \oplus B) + AB$ from two `xor2`, two `and2` and an `or2` (148 fins).

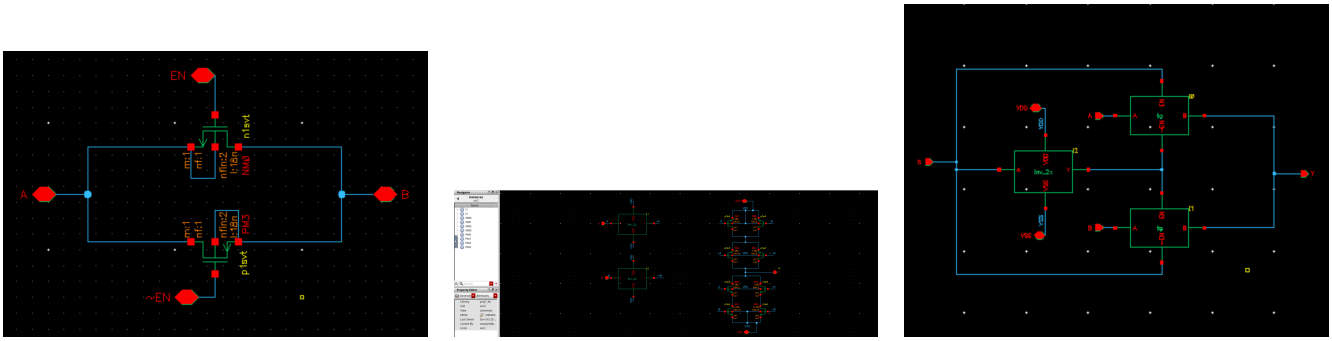


Figure 35: Leaf cells: `tg` (nfin 2 each), `xor2`, `mux2`. The `xor2` view is the corrected one (two series pull-down stacks).

3.2 Multiplier

Array multiplier: 16 AND gates form the partial products $a_i b_j$ (four `and2x4`, one per row j), and three 4-bit ripple-carry adders (`fulladder_4b`, 12 full adders) sum the rows, each row shifted by one bit; $P_0 = a_0 b_0$, $P_{1..3}$ from the low bit of each adder row, $P_{4..7}$ from the last row and its carry. It reuses one verified full-adder cell in a regular array, which made it quick to build and debug; the cost is speed, because the longest path ripples through all three adder rows into the last row's carry (P_6, P_7). A carry-save (Wallace or Dadda) tree would shorten the path to two full-adder levels plus one final carry-propagate adder, for the same 12 full adders but irregular wiring. 2096 fins.

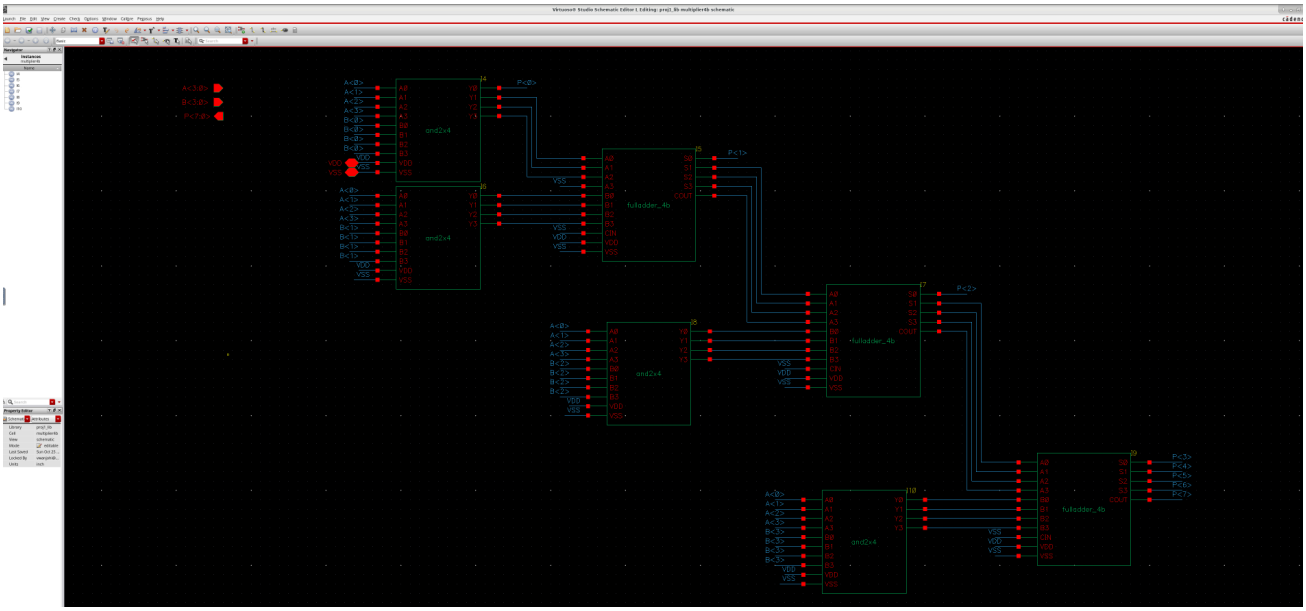


Figure 36: `multiplier4b` (corrected): partial-product rows I4, I6, I8, I10 (`and2x4`, row j gets $A\langle 0..3\rangle$ and B_j on all four B pins) and ripple rows I5, I7, I9 (`fulladder_4b`, $C_{in} = 0$); each row adds the previous row's upper sum bits and carry to the next partial product. Sizes ($L = 18$ nm): `and2` = `nand2_u` (PMOS 2‖2, NMOS 2+2 series fins) + `inv_2x` (8/4); full adder: two `xor2` (NMOS 2/4, PMOS 2 fins, input `inv_2x`), two `and2`, one `or2`.

3.3 Accumulator adder

12-bit ripple-carry adder of the same full adder (`fulladder_12b`): $A = \mathrm{ACC}\langle 11{:}0\rangle$, $B = \{0000, P\langle 7{:}0\rangle\}$, $C_{in} = 0$; the sum goes to `register_12b` (12 `register_1b`, EN = ACC_EN, synchronous reset on CLEAR). Twelve bits are enough: the largest result is $16 \times 15 \times 15 = 3600 < 4096$. Only the low 8 bits receive the product, so the upper 4 stages only propagate the carry. 3600 fins for the accumulator (1776 adder, 1824 register).

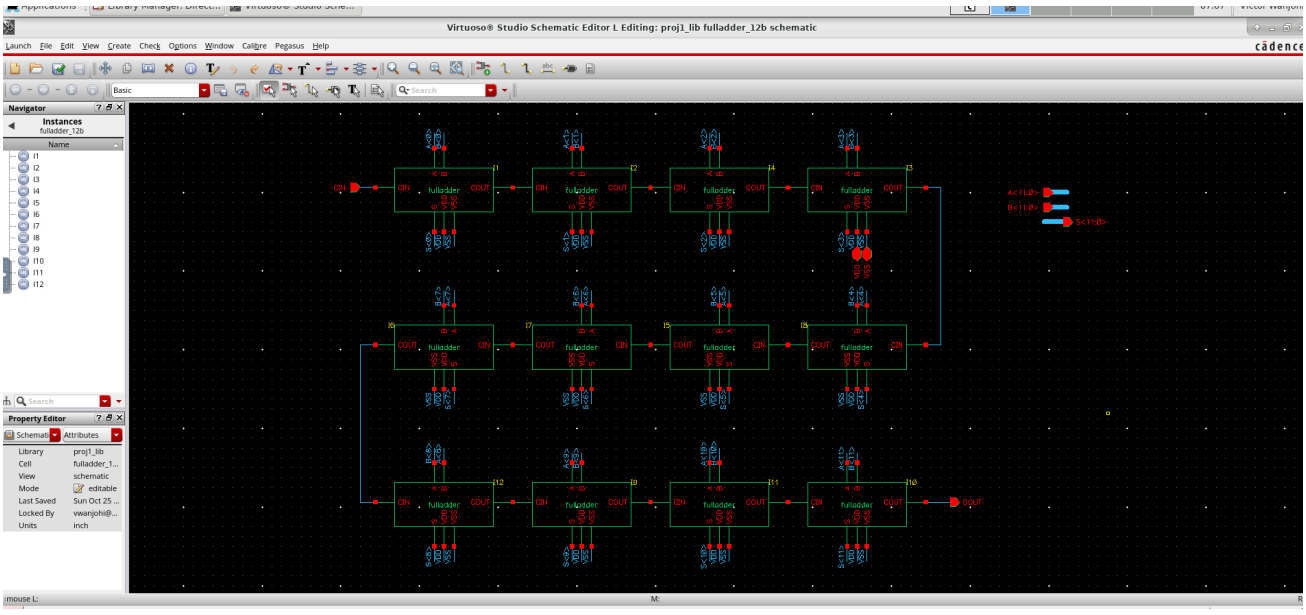


Figure 37: `fulladder_12b`: twelve `fulladder` cells in a ripple chain (I1 bit 0 with C_{in} to I10 bit 11 with C_{out}), drawn as three rows of four; each cell is 148 fins (two `xor2`, two `and2`, one `or2`).

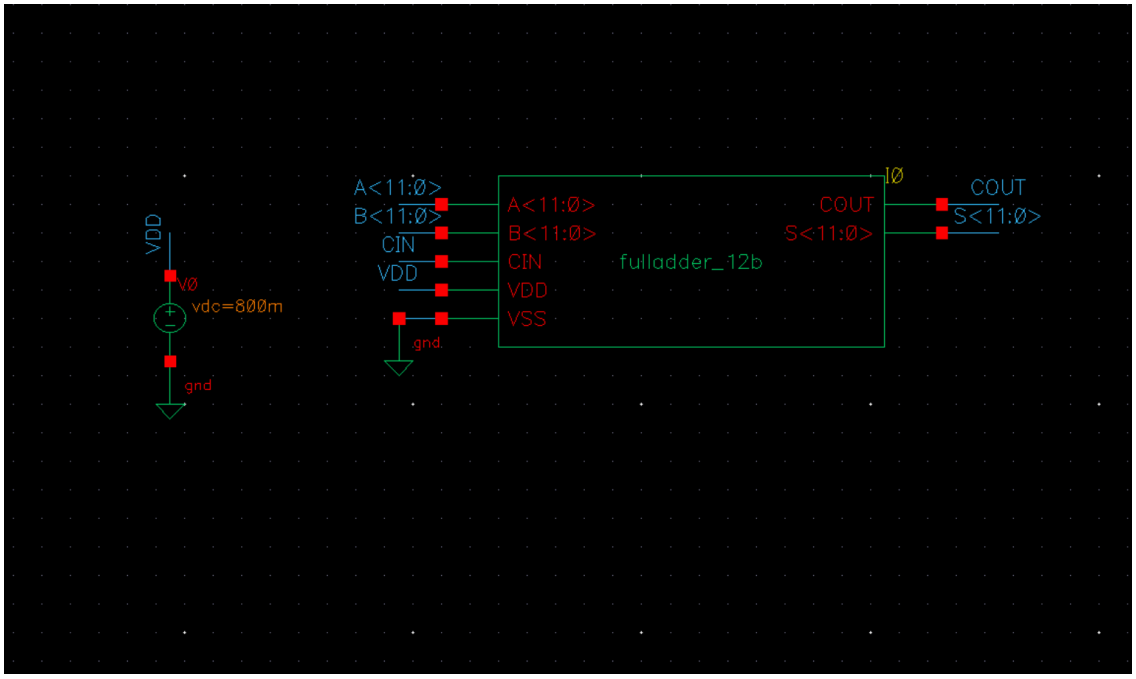


Figure 38: `tb_fulladder_12b`: `fulladder_12b` at $V_{DD} = 0.8$ V. Ports A⟨11:0⟩, B⟨11:0⟩, CIN in; SUM and COUT out.

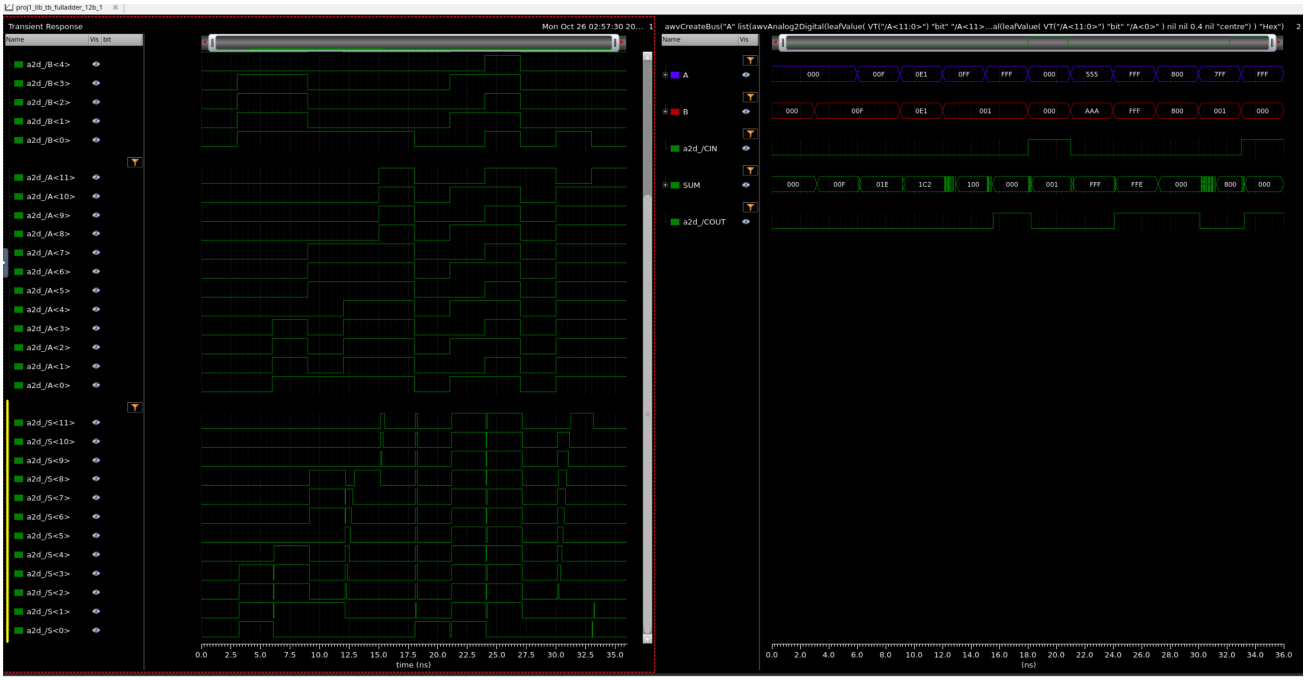


Figure 39: `tb_fulladder_12b`, 0–35 ns: A, B, CIN and SUM as bits (left) and hex buses (right).

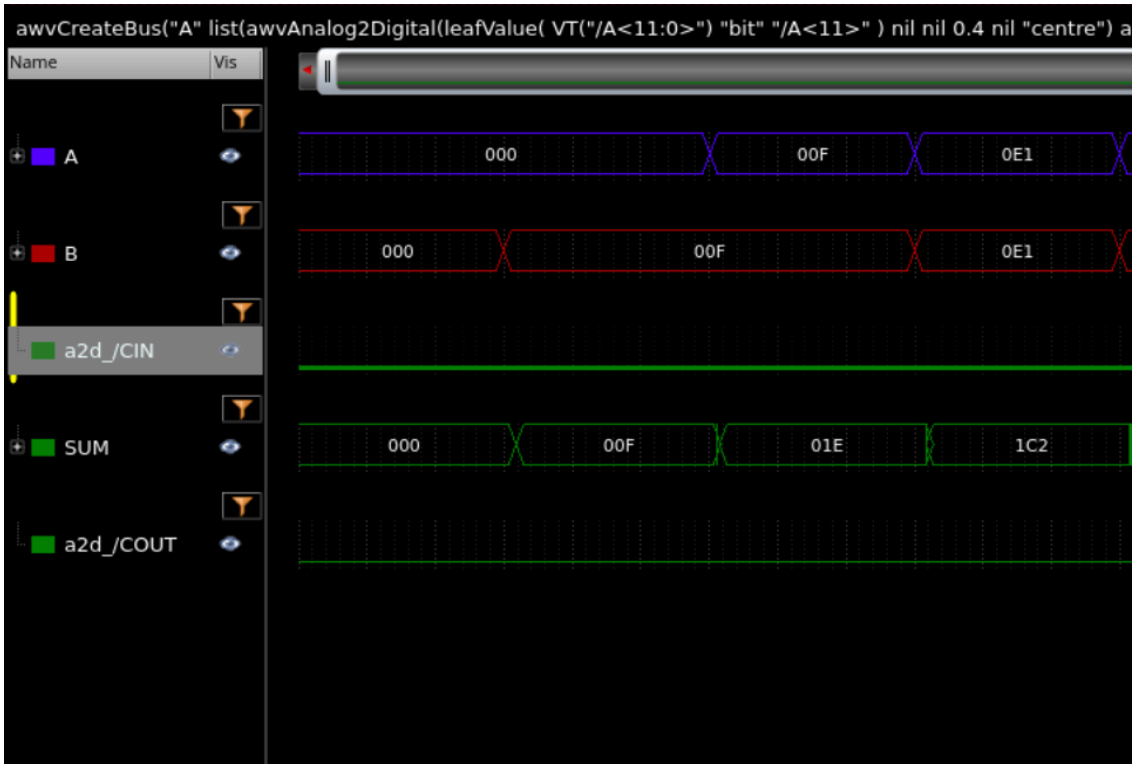


Figure 40: Close-up: 0x00F + 0x00F = 0x01E; 0x0E1 + 0x0E1 = 0x1C2. CIN = 0, COUT = 0.

3.4 Logical-effort analysis

Delay of each stage in units of $\tau = 10.5$ ps (Part 2: FO4 = 53 ps = 5τ, chain of `inv_u`) as $d = f + p$, with the effort $f = g\,h$ computed directly from fin counts: $f = R \cdot C_{\text{load}}/C_{\text{inv_u}}$, where R is the gate's worst-edge pull-up or pull-down resistance relative to `inv_u` (PMOS 4 / NMOS 2 fins, $C_{\text{inv_u}} = 6$ fins) and C_{load} the gate fins it drives. A full-adder input or carry-in (one `xor2` input: `inv_2x` 12 fins + two 2-fin devices; plus one `nand2_u` input, 4 fins) is 20 fins.

Stage	Load	R (worst)	f	p	d [τ]
`inv_2x` (xor2 input inverter)	4 fins	0.5	0.3	1	1.3
`xor2` core (2-series: PMOS 2+2, NMOS 2+4 fins)	20 fins	2.75	9.2	3	12.2
`nand2_u` (2+2 series, 2 ‖ 2)	12 fins	2	4	2	6
`inv_2x` (and2 output)	4 fins	0.5	0.3	1	1.3
`nor2` in `or2` (PMOS 2+2 series)	12 fins	2.5	5	2	7
`inv_2x` (or2 output = C_{out})	20 fins	0.5	1.7	1	2.7

Full-adder arcs: $C_{in} \to C_{out}$ (and2 + or2) $= 6 + 1.3 + 7 + 2.7 \approx 17\tau$; $C_{in} \to S$ (one xor2) $\approx 13\tau$; $A/B \to S$ (two xor2) $\approx 26\tau$; $A/B \to C_{out}$ (xor2, and2, or2) $\approx 30\tau$; a partial-product `and2` into a full-adder input $\approx 6 + 2.7 \approx 9\tau$.

Multiplier critical path. With ripple rows, the longest path alternates carry and sum through all three rows and then ripples along the last one: $a_i b_j$ (and2) $\to$ I5 bit 0 ($A/B \to C_{out}$) $\to$ I5 bit 1 ($C_{in} \to S$) $\to$ I7 bit 0 ($A/B \to C_{out}$) $\to$ I7 bit 1 ($C_{in} \to S$) $\to$ I9 bit 0 ($A/B \to C_{out}$) $\to$ I9 bits 1–3 ($C_{in} \to C_{out}$) $\to P_7$:

$$D_{\text{mult}} \approx 9 + 30 + 13 + 30 + 13 + 30 + 3 \times 17 = 176\,\tau \approx 1.85 \text{ ns}.$$

Measured worst case: 1.14–1.24 ns. The estimate is about 50% high, as expected: it charges every stage its slower edge and the full 20-fin load, and the slowest simulated vector (244) need not excite the whole alternating path.

Both agree on what dominates: the `xor2` core, whose 2-fin series PMOS stack makes it about 4× weaker than `inv_u` on the rising edge (12 of the 26 τ in a sum arc). **Sizing from this analysis:** widening the `xor2` stacks to 4-fin PMOS and NMOS ($R \approx 1$) cuts each xor2 stage from $\approx 12\tau$ to $\approx 5\tau$ and the multiplier path to $\approx 120\tau \approx 1.3$ ns (LE); the cells were simulated at the drawn sizes.

Accumulator adder. The product bits arrive at different times; the late ones are the upper bits. Worst case: P_4 settles last, then $A/B \to C_{out}$ at bit 4, carries through bits 5–10 and $C_{in} \to S$ at bit 11:

$$D_{\text{add}} = 30 + 6 \times 17 + 13 = 145\,\tau \approx 1.5\,\text{ns}.$$

A carry from ACC bit 0 through all 12 bits,

$$30 + 11 \times 17 \approx 217\,\tau \approx 2.3\,\text{ns},$$

starts at t_{cq} after the edge and so ends earlier than the path through the multiplier. The register input adds the `mux2` and `and2` before the flip-flop, $\approx 10\tau \approx 0.1$ ns.

3.5 Unit tests

Block	Bench	Test	Predicted	Measured
Multiplier	`tb_multiplier`	Exhaustive: all 256 pairs vs. $a \cdot b$; 0 errors for $t_v \geq 1.3$ ns; worst vector 244 (A 3 → 4, $B = 15$)	1.85 ns (LE, 176τ)	1.14–1.24 ns (≈ 1.2 ns)
12-bit acc.	`tb_accumulator_12b`	accumulate, RST, hold; 16×0xE1 → 0xE10	—	$16 \times 225 = 3600$

Multiplier bench (`tb_multiplier4b`, Fig. below). Eight `vpulse` sources count through all 256 (A, B) pairs, one per t_v (A the low nibble: bit k has delay $2^k t_v$, width $2^k t_v$, period $2^{k+1} t_v$; B bit k uses $2^{k+4} t_v$), 20 ps edges, tran stop $256\,t_v$; each product bit drives one `inv_2x` (`LD<7:0>`). Inputs are ideal voltage sources, not buffered by `inv_u`.

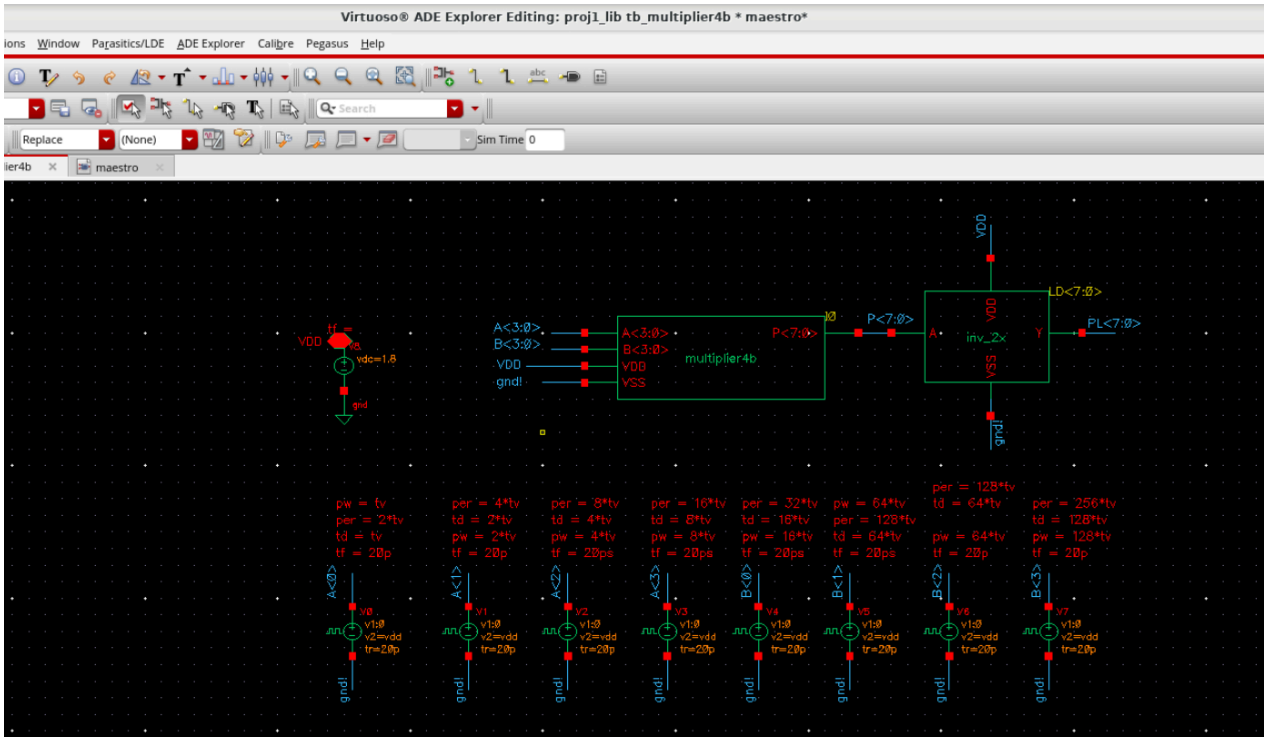


Figure 41: `tb_multiplier4b` (schematic capture; VDD in this view is the bench source).

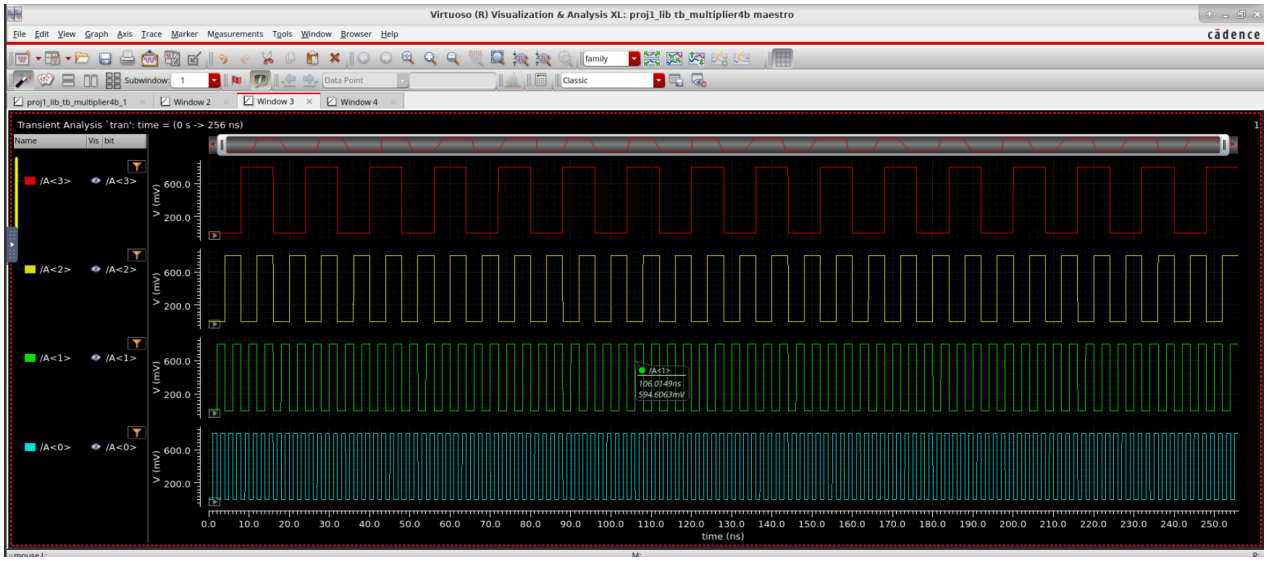


Figure 42: Stimulus $A\langle 3{:}0\rangle$ at $t_v = 1$ ns: A_0 toggles every vector, A_3 every 8.

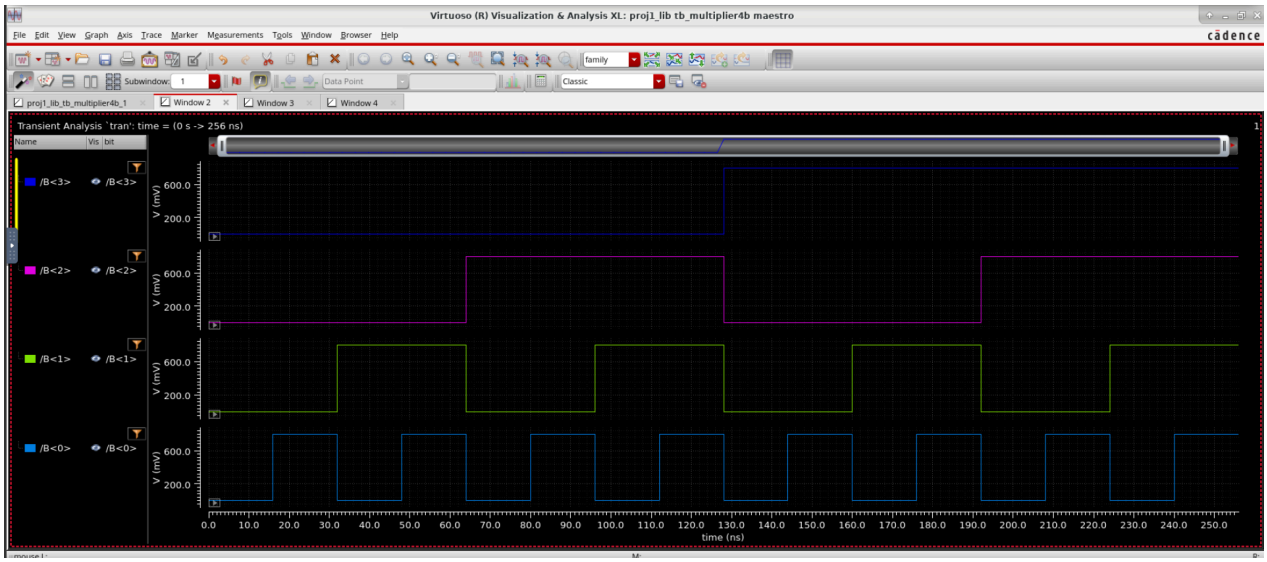


Figure 43: Stimulus $B\langle 3{:}0\rangle$: B steps once every 16 vectors, so all 256 pairs occur in 256 ns.

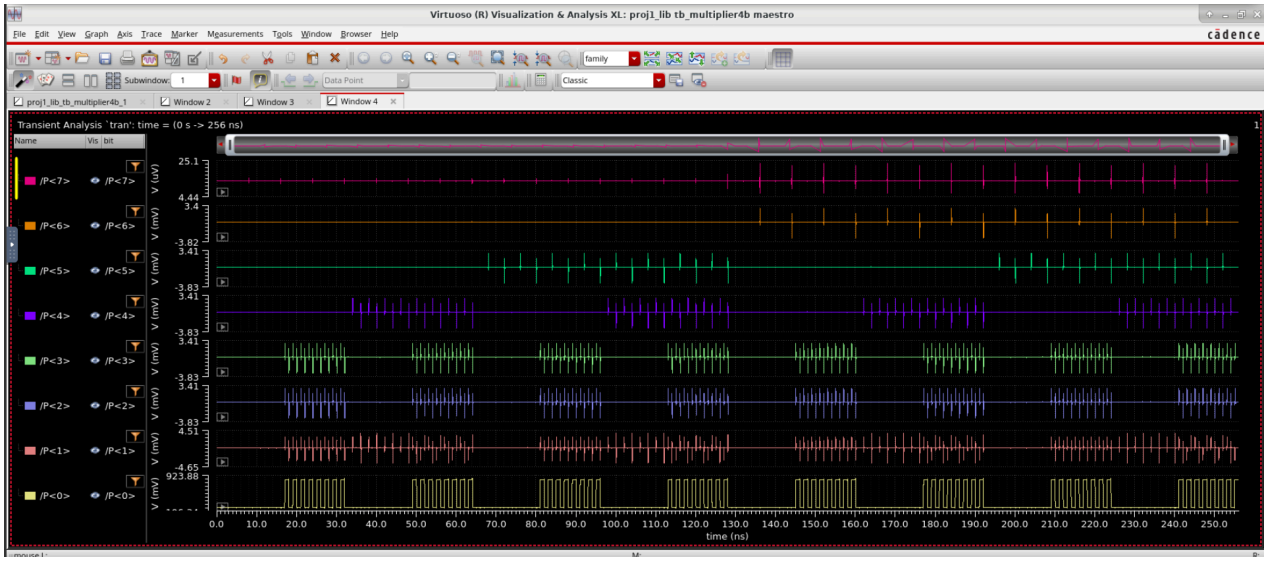


Figure 44: First run (before the fixes): only $P_0 = a_0 b_0$ switches; $P_{1..7}$ stay near 0 V (mV-scale glitches). This located two design errors, below.

Errors found by the first run. (1) `xor2`: the pull-down had the $A/\bar{A}$ devices in parallel, in series with the $B/\bar{B}$ pair in parallel, so it always conducted and fought the pull-up when $A \neq B$; corrected to two series stacks $A \cdot B$ and $\bar{A} \cdot \bar{B}$. (2) `multiplier4b`: all four `and2x4` received $(A\langle 0..3\rangle, B\langle 0..3\rangle)$ and formed $a_i b_i$; each row j now receives B_j on all four B inputs. A logic-level check of the corrected netlist gives 0 wrong products out of 256. The t_v sweep below is that rerun.

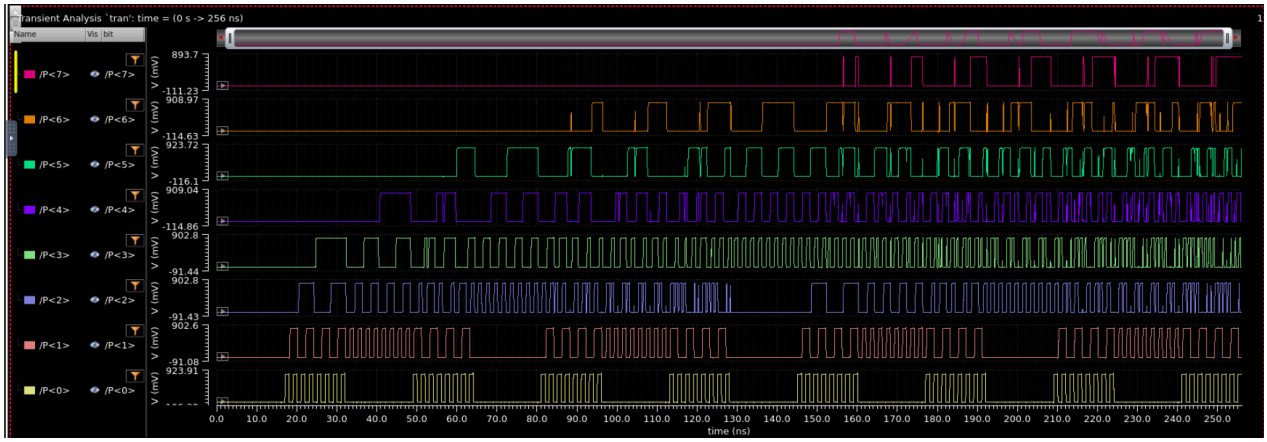


Figure 45: After the fixes ($t_v = 1$ ns): every product bit switches. First rising edges match the hand values: P_1 at vector 18 (2×1), P_3 at 24 (8×1), P_4 at 40 (8×2), P_5 at 59 (11×3), P_7 at 159 ($15 \times 9 = 135$); short spikes are ripple glitches inside a vector.

Exhaustive check. Three ADE expressions turn the bits into integers: $P_{int} = \sum_k 2^k V(P_k)/V_{DD}$, $AB_{int} = (\sum_k 2^k V(A_k)/V_{DD})(\sum_k 2^k V(B_k)/V_{DD})$, and $\text{err} = |P_{int} - AB_{int}|$. `errS` samples err once per vector at $0.95\,t_v$ (`sample(err, 0.95 tv, 255.95 tv, "linear", tv)`) and `errmax` = `ymax(errS)` with spec < 0.5.

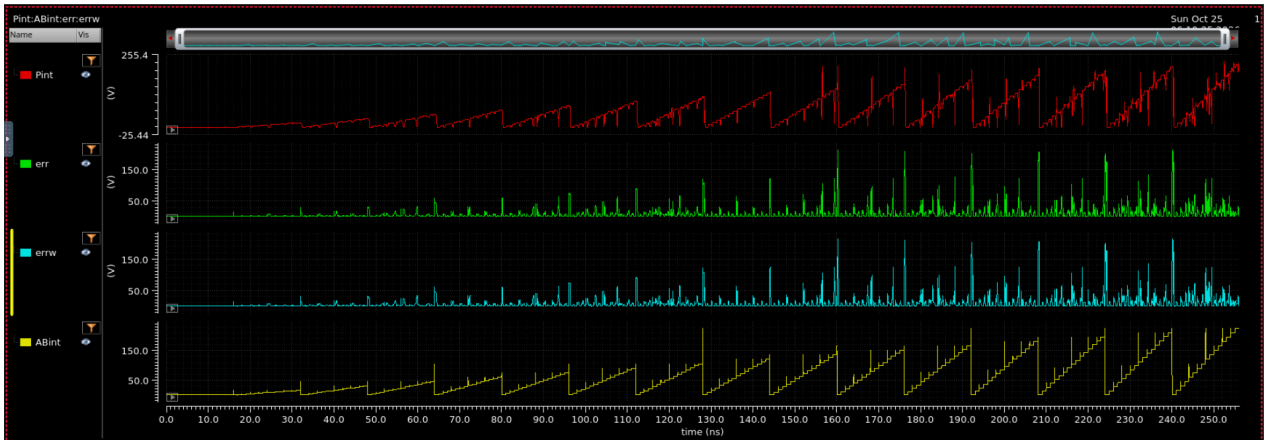


Figure 46: P_{int} follows the AB_{int} staircase in every block of 16 vectors; the err spikes sit at the vector boundaries, where AB_{int} itself spikes because the input bits do not cross at the same instant.

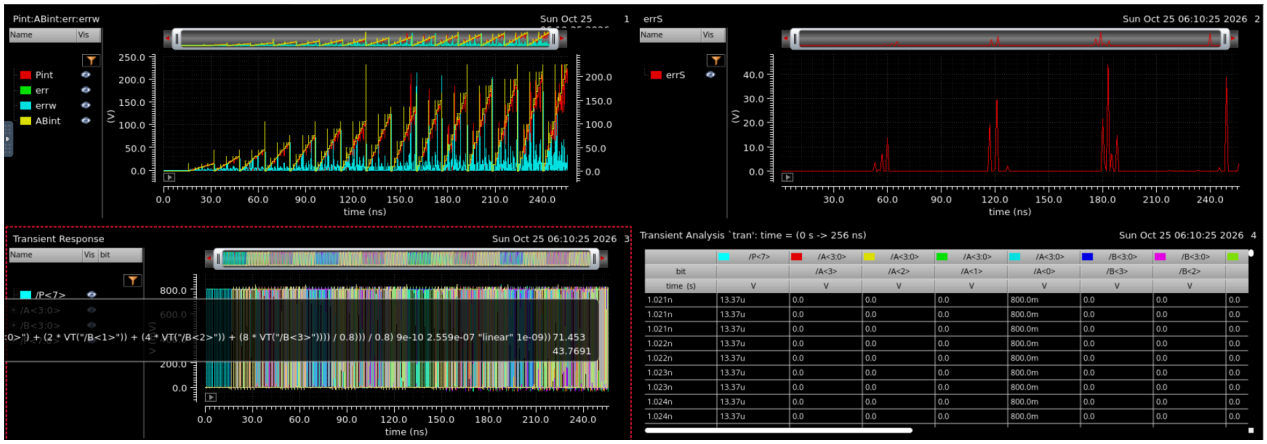


Figure 47: Sampled check at $t_v = 1$ ns (top right, `errS`): 0 for almost all 256 vectors; about ten vectors in four clusters (56–60, 118–122, 180–185, 242–246 ns) are non-zero, peak `errmax` = 43.77.

At $t_v = 1$ ns `errmax` = 43.77 (71.45 with the earlier 0.9 t_v sample point). The non-zero values are fractional, not the power-of-two error a settled wrong bit would give, so those vectors are still switching about 0.95 ns after the inputs: a settling limit, not a logic error.

t_v **sweep.** Tran stop 256 t_v, so every point simulates all 256 vectors; `t_worst` gives the worst vector, $i = \text{round}(t/t_v - 0.95)$.

t_v	sample (0.95 t_v)	`errmax`	`t_worst`	result (< 0.5)
1.0 ns	0.950 ns	43.77	183.0 ns	fail (vector 182)
1.1 ns	1.045 ns	13.66	269.4 ns	fail (vector 244)
1.2 ns	1.140 ns	0.621	293.9 ns	fail (vector 244)
1.3 ns	**1.235 ns**	**0.170**	323.6 ns	**pass**
1.4 ns	1.330 ns	0.136	348.5 ns	pass
1.5 ns	1.425 ns	0.118	373.4 ns	pass

All 256 products are correct once $t_v \geq 1.3$ ns. The slowest vector is 244 ($B = 15$, A: 0011 → 0100, product 45 → 60): with $B = 15$ all four partial-product rows are active and three A bits switch, so the carry ripples through the whole

array into P_4. It is still settling at 1.14 ns and settled by 1.235 ns, so the **measured worst-case multiplier delay is 1.14–1.24 ns ($\approx$ 1.2 ns)**. A second data point: vector 182 (A 5 $\rightarrow$ 6, $B = 11$) has P_6 crossing $V_{DD}/2$ 962 ps after the input edge (`d_mult` at $t_v = 1$ ns). Typical vectors settle in 0.3–0.6 ns (zoom below). The residual 0.12–0.17 at passing t_v is the bit voltages sitting a few mV off the rails, weighted up to 128.

78 rows

Point	Test	Output	Nominal	Spec	Weight	Pass/Fail
Parameters: tv=1n						
1	proj1_lib_tb_mul...	/P<7>				
1	proj1_lib_tb_mul...	/A<3:0>				
1	proj1_lib_tb_mul...	/B<3:0>				
1	proj1_lib_tb_mul...	/P<7:0>				
1	proj1_lib_tb_mul...	errmax_old	71.45			
1	proj1_lib_tb_mul...	Pint				
1	proj1_lib_tb_mul...	ABint				
1	proj1_lib_tb_mul...	err				
1	proj1_lib_tb_mul...	errw				
1	proj1_lib_tb_mul...	errS				
1	proj1_lib_tb_mul...	errmax	43.77	< 0.5		fail
1	proj1_lib_tb_mul...	tworst	**183n**	< (256 * tv)		error
1	proj1_lib_tb_mul...	d_mult	962.1p			
Parameters: tv=1.1n						
2	proj1_lib_tb_mul...	/P<7>				
2	proj1_lib_tb_mul...	/A<3:0>				
2	proj1_lib_tb_mul...	/B<3:0>				
2	proj1_lib_tb_mul...	/P<7:0>				
2	proj1_lib_tb_mul...	errmax_old	206			
2	proj1_lib_tb_mul...	Pint				
2	proj1_lib_tb_mul...	ABint				
2	proj1_lib_tb_mul...	err				
2	proj1_lib_tb_mul...	errw				
2	proj1_lib_tb_mul...	errS				
2	proj1_lib_tb_mul...	errmax	13.66	< 0.5		fail
2	proj1_lib_tb_mul...	tworst	**269.4n**	< (256 * tv)		error
2	proj1_lib_tb_mul...	d_mult	eval err			
Parameters: tv=1.2n						
3	proj1_lib_tb_mul...	/P<7>				
3	proj1_lib_tb_mul...	/A<3:0>				
3	proj1_lib_tb_mul...	/B<3:0>				
3	proj1_lib_tb_mul...	/P<7:0>				
3	proj1_lib_tb_mul...	errmax_old	158.9			
3	proj1_lib_tb_mul...	Pint				
3	proj1_lib_tb_mul...	ABint				
3	proj1_lib_tb_mul...	err				
3	proj1_lib_tb_mul...	errw				
3	proj1_lib_tb_mul...	errS				
3	proj1_lib_tb_mul...	errmax	621m	< 0.5		fail
3	proj1_lib_tb_mul...	tworst	**293.9n**	< (256 * tv)		error
3	proj1_lib_tb_mul...	d_mult	669.2p			
Parameters: tv=1.3n						
4	proj1_lib_tb_mul...	/P<7>				
4	proj1_lib_tb_mul...	/A<3:0>				
4	proj1_lib_tb_mul...	/B<3:0>				
4	proj1_lib_tb_mul...	/P<7:0>				
4	proj1_lib_tb_mul...	errmax_old	164.2			
4	proj1_lib_tb_mul...	Pint				
4	proj1_lib_tb_mul...	ABint				
4	proj1_lib_tb_mul...	err				
4	proj1_lib_tb_mul...	errw				
4	proj1_lib_tb_mul...	errS				
4	proj1_lib_tb_mul...	errmax	170.3m	< 0.5		pass
4	proj1_lib_tb_mul...	tworst	**323.6n**	< (256 * tv)		error
4	proj1_lib_tb_mul...	d_mult	eval err			
Parameters: tv=1.4n						
5	proj1_lib_tb_mul...	/P<7>				
5	proj1_lib_tb_mul...	/A<3:0>				
5	proj1_lib_tb_mul...	/B<3:0>				
5	proj1_lib_tb_mul...	/P<7:0>				
5	proj1_lib_tb_mul...	errmax_old	89.95			
5	proj1_lib_tb_mul...	Pint				
5	proj1_lib_tb_mul...	ABint				
5	proj1_lib_tb_mul...	err				
5	proj1_lib_tb_mul...	errw				
5	proj1_lib_tb_mul...	errS				
5	proj1_lib_tb_mul...	errmax	136.2m	< 0.5		pass
5	proj1_lib_tb_mul...	tworst	**348.5n**	< (256 * tv)		error
5	proj1_lib_tb_mul...	d_mult	eval err			
Parameters: tv=1.5n						
6	proj1_lib_tb_mul...	/P<7>				
6	proj1_lib_tb_mul...	/A<3:0>				
6	proj1_lib_tb_mul...	/B<3:0>				
6	proj1_lib_tb_mul...	/P<7:0>				
6	proj1_lib_tb_mul...	errmax_old	63.12			
6	proj1_lib_tb_mul...	Pint				
6	proj1_lib_tb_mul...	ABint				
6	proj1_lib_tb_mul...	err				
6	proj1_lib_tb_mul...	errw				
6	proj1_lib_tb_mul...	errS				
6	proj1_lib_tb_mul...	errmax	118.4m	< 0.5		pass
6	proj1_lib_tb_mul...	tworst	**373.4n**	< (256 * tv)		error
6	proj1_lib_tb_mul...	d_mult	eval err			

Figure 48: ADE Results for the sweep: `errmax` (spec < 0.5) fails at 1.0–1.2 ns and passes from 1.3 ns. (The "error" on `t_worst` is only its spec, which cannot use a design variable; `d_mult` uses a fixed 182–183 ns window and is valid only at $t_v = 1$ ns; `errmax_old` uses fixed sample times.)

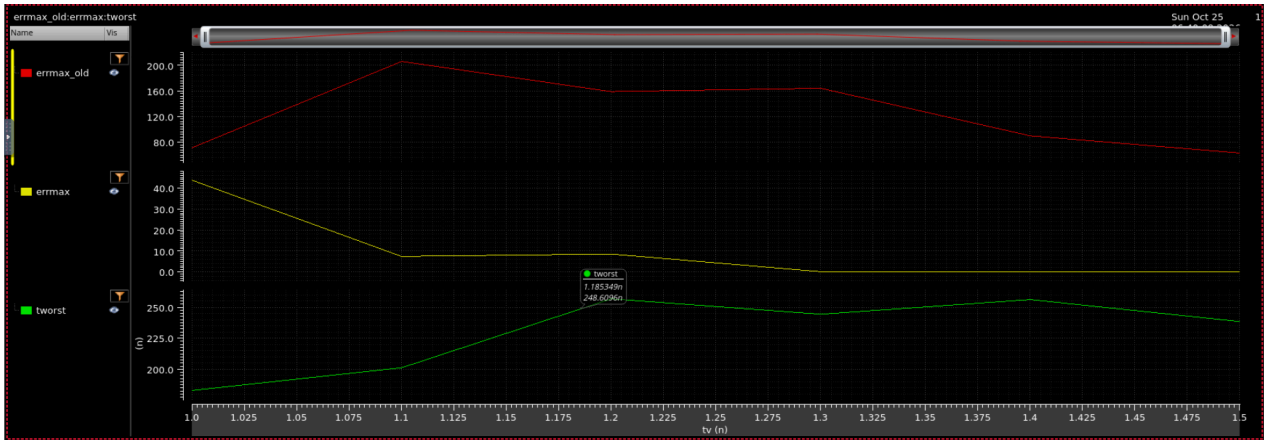


Figure 49: `errmax` and `t_worst` against t_v from an earlier run with a fixed 256 ns stop (covers fewer vectors above 1 ns); kept for the shape of the curve, the table above is the result.

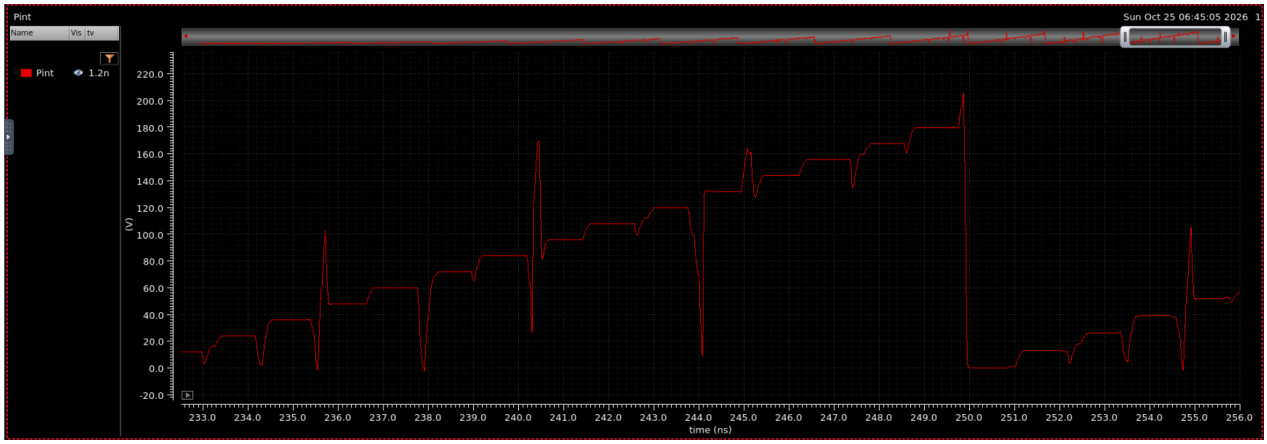


Figure 50: P_{int} at $t_v = 1.2$ ns, 233–256 ns ($B = 12$, then $B = 13$): each product settles 0.3–0.6 ns after its vector edge; the spike near 249.9 ns is the 180 → 0 step where all four A bits and B_0 flip.

12-bit adder vectors. Hand checks on the bus plot: 0x00F + 0x00F = 0x01E; 0x0E1 + 0x0E1 = 0x1C2; 0x0FF+0x001 = 0x100; 0x555+0xAAA = 0xFFF; 0xFFF+0xFFF sets SUM =0xFFE and COUT; 0x7FF+0x001 = 0x800. The ripple adder is logically correct.

Accumulator (`tb_accumulator_12b`). DUT is `accumulator_12b`: CLK, EN, PROD⟨7:0⟩, RST in; ACC⟨11:0⟩ out; $V_{DD} = 0.8$ V.

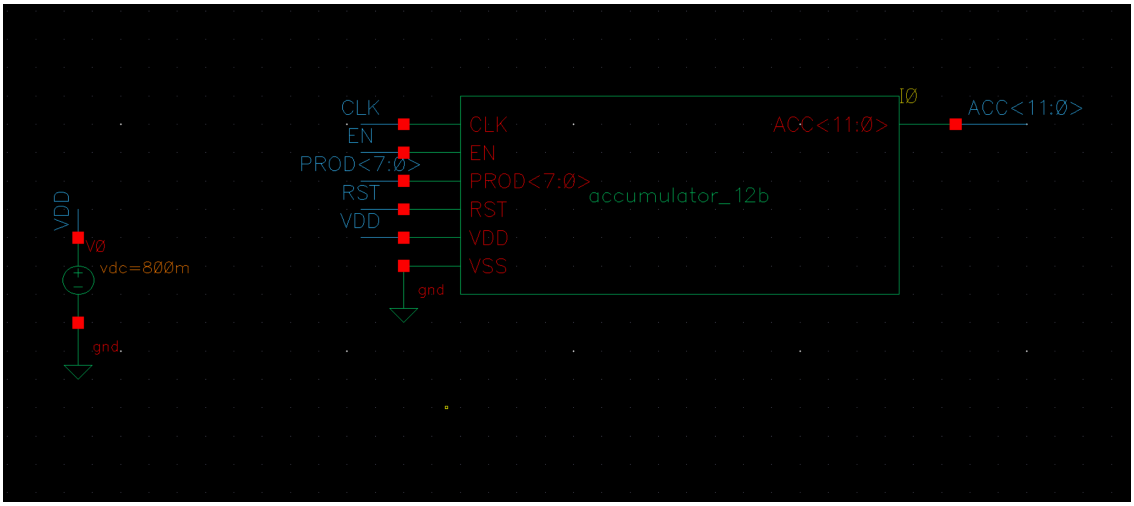


Figure 51: `tb_accumulator_12b`: `accumulator_12b` with product byte PROD⟨7:0⟩ and 12-bit ACC.

After RST, PROD = 0x0F (15) is added while EN is high: ACC = 0x00F, 0x01E, 0x02D, 0x03C. EN falls (hold), RST clears to 0. Then PROD = 0xE1 ($15 \times 15 = 225$) for sixteen clocks: ACC steps 0x0E1, 0x1C2, …, 0xD2F, 0xE10. $16 \times 225 = 3600 = 0$xE10, the Test B full-scale sum, with carry into the upper nibble.



Figure 52: `tb_accumulator_12b`, 0–100 ns: ACC and PROD bits (left); CLK, EN, RST and hex ACC/PROD (right).

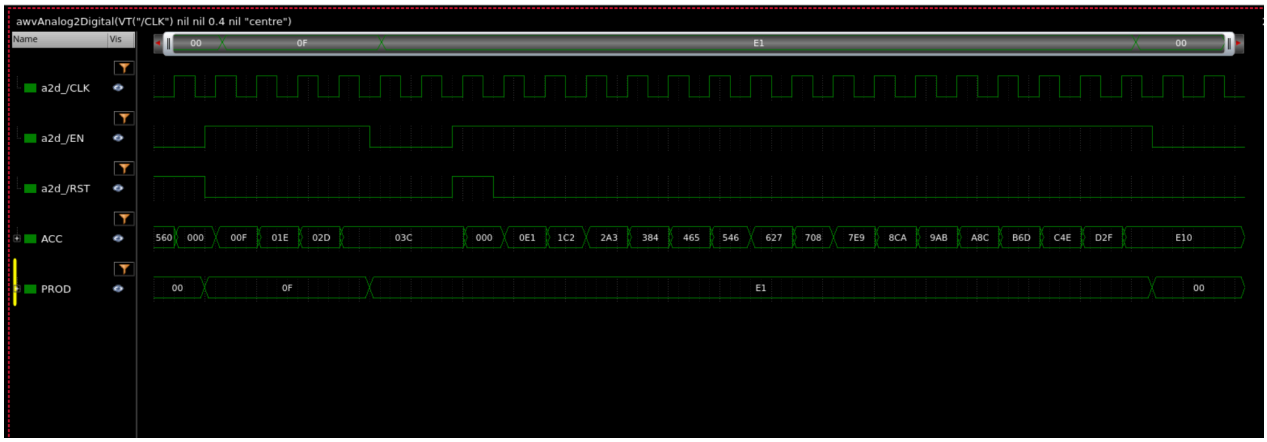


Figure 53: Hex close-up: four adds of 0x0F, RST, then sixteen adds of 0xE1 to 0xE10.

4. Sequencer

4.1 State diagram

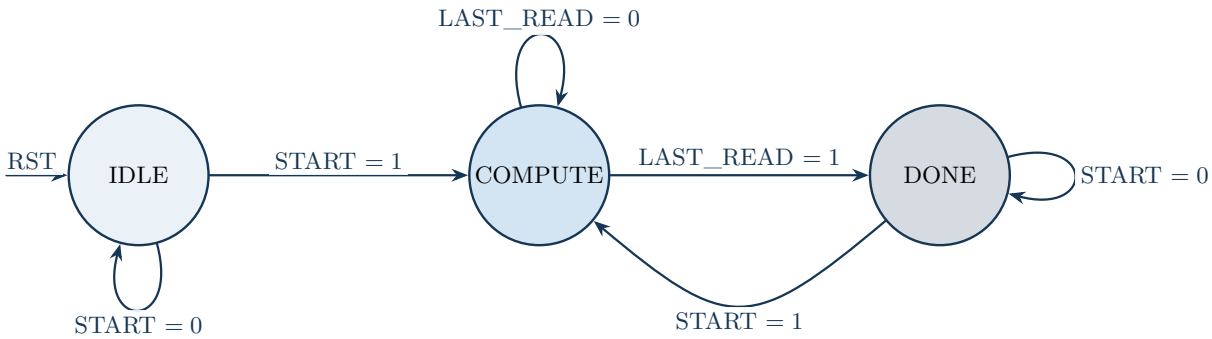


Figure 54: `controller_mac` FSM. Equal-size states; self-loops are holds. START from IDLE or DONE enters COMPUTE (START is ignored while RUN). LAST_READ ends the run one cycle after the counter reaches 15. RST from any state returns to IDLE.

Encoding (from `controller_mac`): two state registers, S_0 and DONE; IDLE = $(S_0, \text{DONE}) = (0, 0)$, COMPUTE: RUN = $S_0\overline{\text{DONE}} = 1$, DONE = $(0, 1)$. Two more registers carry the pipeline: VALID (= RUN one cycle late, so the first product is accumulated one cycle after its read) and LAST_READ (set the cycle after the counter reaches 15). RST clears S_0 and DONE from any state. Per state:

- **IDLE / DONE, START high:** $G_0 = \text{START} \cdot \overline{\text{RUN}}$ raises CLEAR (clears the accumulator, the counter, VALID and LAST_READ) and sets S_0; DONE clears. START is ignored while RUN.
- **COMPUTE:** COUNT_EN = RUN $\cdot \overline{\text{LAST}}$ steps the counter 0 to 15 and holds it at 15 (LAST = count = 15); ACC_EN = RUN $\cdot$ VALID; S_0 stays set while LAST_READ = 0.
- **Leaving COMPUTE:** FINISH = RUN $\cdot$ LAST_READ sets DONE and clears S_0, so the exit is one cycle after the count reaches 15, which leaves time to accumulate the last product.
- **DONE:** held by DONE $\cdot \overline{\text{START}}$; ACC holds because ACC_EN = 0.

4.2 Counter and controller

`counter_4b` is a synchronous toggle-chain counter: $T_1 = EN \cdot Q_0$, $T_2 = T_1 Q_1$, $T_3 = T_2 Q_2$ (`and2`), $D_k = (Q_k \oplus T_k) \cdot \overline{RST}$ with $T_0 = EN$ (`xor2`, `and2`), four `d_ff`; RST is driven by CLEAR, so the count restarts at 0 on START (728 fins). `controller_mac` is the logic above: 4 `register_1b`, 11 `and2`, 3 `or2`, 5 `inv_2x` (948 fins). In `top`, four `mux2` send COUNT to the memory address while RUN = 1 and the external ADDR otherwise, and LOAD_EN = $\overline{\text{RUN} + \text{DONE} + \text{START}}$ gates WE_W and WE_X. The counter passes all 38 vector checks (below). The controller, closed-loop with the counter, follows the expected sequence edge for edge, with DONE 17 edges after START (below).

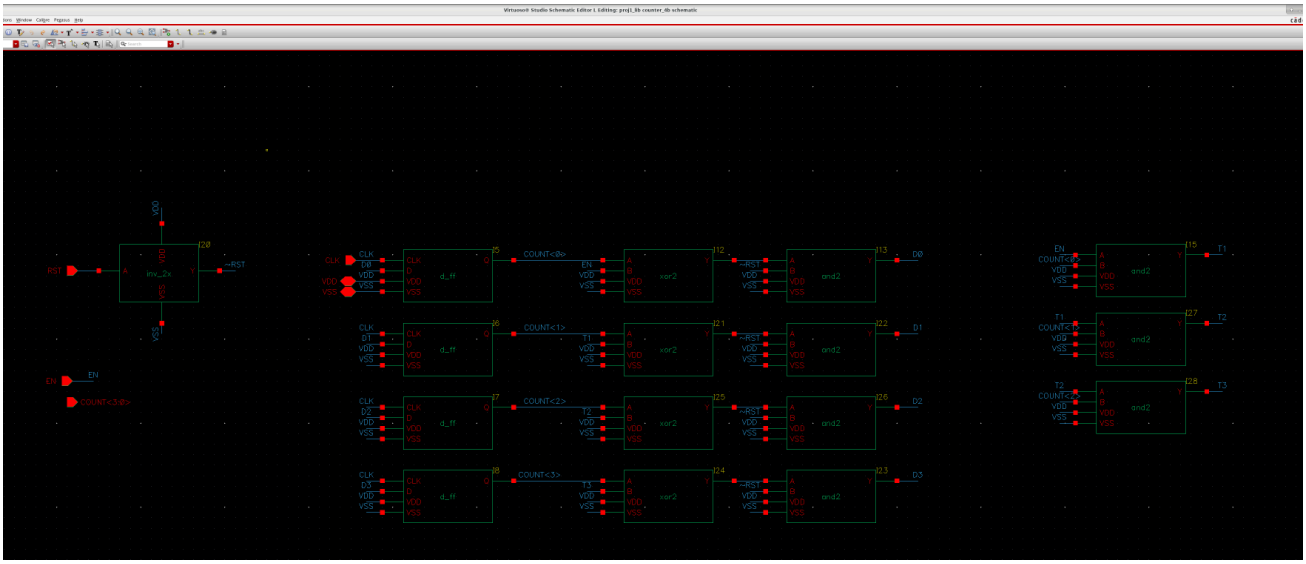


Figure 55: `counter_4b`: four `d_ff` (I5–I8); $D_k = (Q_k \oplus T_k) \cdot \overline{RST}$ through `xor2` and `and2` (I12/I13, I21/I22, I25/I26, I24/I23); toggle chain $T_1 = EN \cdot Q_0$, $T_2 = T_1 Q_1$, $T_3 = T_2 Q_2$ (I15, I27, I28); RST inverted by I20. 728 fins.

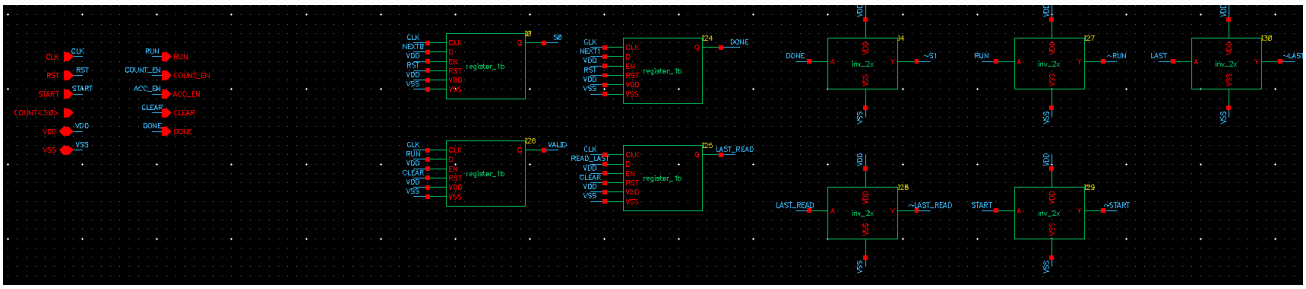


Figure 56: `controller_mac`, state registers: four `register_1b` (S_0 from NEXT0, DONE from NEXT1, both reset by RST; VALID from RUN and LAST_READ from READ_LAST, both reset by CLEAR; EN tied to VDD) and the inverters $\overline{S_1} = \overline{\text{DONE}}$, $\overline{\text{RUN}}$, $\overline{\text{LAST}}$, $\overline{\text{LAST_READ}}$, $\overline{\text{START}}$.

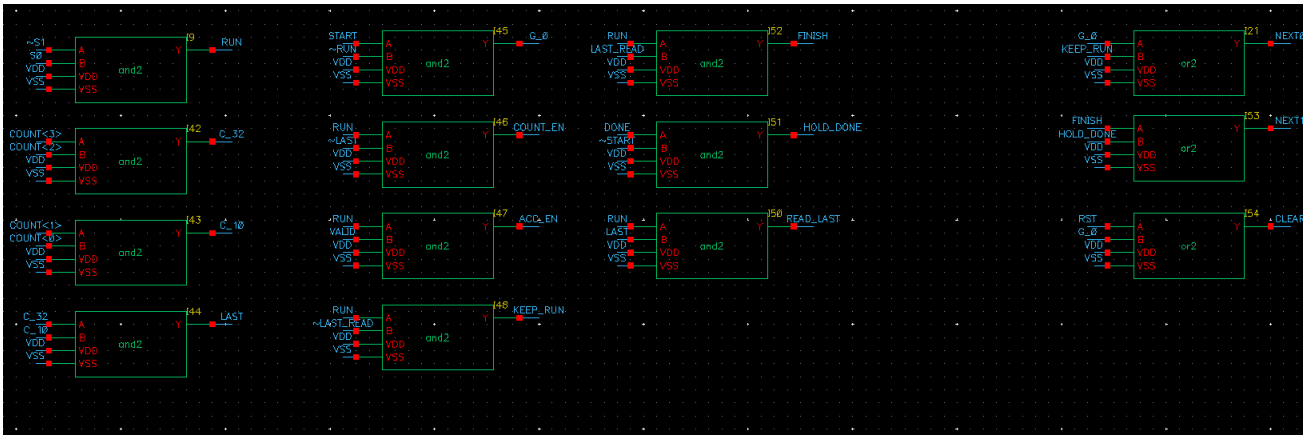


Figure 57: `controller_mac`, next-state and output logic: $\text{RUN} = \overline{S_1} S_0$; $\text{LAST} = C_{32} C_{10}$ (count = 15); $G_0 = \text{START} \cdot \overline{\text{RUN}}$; $\text{COUNT_EN} = \text{RUN} \cdot \overline{\text{LAST}}$; $\text{ACC_EN} = \text{RUN} \cdot \text{VALID}$; KEEP_RUN, FINISH, HOLD_DONE, READ_LAST; NEXT0 $= G_0 + \text{KEEP_RUN}$, NEXT1 $= \text{FINISH} + \text{HOLD_DONE}$, CLEAR $= \text{RST} + G_0$ (`and2`, `or2`).

`tb_counter4b`. Stimulus and check from a Spectre digital vector file (`counter4b.vec`): CLK (1 ns period, rising edges at 0.5, 1.5, … ns), RST and EN change while CLK is low, and COUNT⟨3:0⟩ is compared with the expected value a quarter period after every edge (output thresholds 0.2/0.6 V). The 38-edge sequence covers reset, counting

through the $15 \to 0$ wrap, hold (EN = 0), resume, reset while counting and a final hold; expected COUNT: 0 0 1 2 … F 0 1 2 3 4 | 4 4 4 | 5 … 9 | 0 0 | 1 2 3 4 | 4 4. **Result: 38/38 checks pass** (empty `.vecerr`, Spectre 0 errors, 0 warnings); outputs swing rail to rail with $V_{DD} = 0.8$ V.

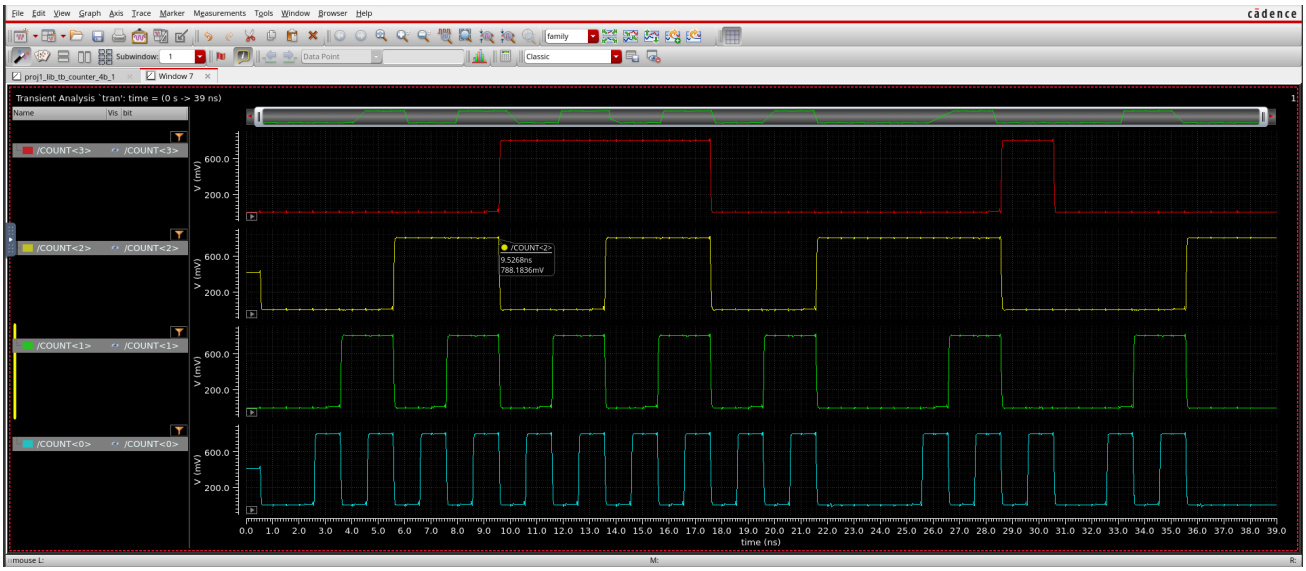


Figure 58: `tb_counter4b`, 0–39 ns: COUNT⟨3:0⟩. Bit 0 toggles every edge, bit 3 is high for counts 8–15 (9.6–17.6 ns) and 8–9 (28.6–30.6 ns); flat during the EN-low holds (21.6–25.6 ns, after 35.6 ns); reset to 0 at 30.6 ns.



Figure 59: Bit 0 next state D_0 and the master-latch node (`I5/net5`) change before each rising edge and COUNT⟨0⟩ follows after it; V_{DD} flat at 0.8 V.

```
vwanjohi@epyc-1-1:/vast/projects/radway/ese-5700-fall/tmarhguy/project_1$ R=$(ls -td ~/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.* | head -1)
grep -i -n "vec" $(find $R -name "spectre.out") | head -20
find $R -name "*.vecerr" -exec sh -c 'echo "== $1 ($(wc -l < "$1") lines)"; head "$1"' _ {} \;
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_counter_4b_1/psf/spectre.out:106:Reading file:  /vast/home/v/vwanj
ohi/simulation/stimuli/counter4b.vec
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_counter_4b_1/psf/spectre.out:108:Notice from spectre during Digita
l Vector read-in.
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_counter_4b_1/psf/spectre.out:109:    Process Vector Files.
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_counter_4b_1/psf/spectre.out:110:    File read: /vast/home/v/vwanj
ohi/simulation/stimuli/counter4b.vec
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/psf/proj1_lib_tb_counter_4b_1/psf/spectre.out:106:Reading file:  /vast/home/v/vwa
njohi/simulation/stimuli/counter4b.vec
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/psf/proj1_lib_tb_counter_4b_1/psf/spectre.out:108:Notice from spectre during Digi
tal Vector read-in.
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/psf/proj1_lib_tb_counter_4b_1/psf/spectre.out:109:    Process Vector Files.
/vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/psf/proj1_lib_tb_counter_4b_1/psf/spectre.out:110:    File read: /vast/home/v/vwa
njohi/simulation/stimuli/counter4b.vec
== /vast/home/v/vwanjohi/simulation/proj1_lib/tb_counter_4b/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_counter_4b_1/netlist/input_tran.vecerr (0 lines)
vwanjohi@epyc-1-1:/vast/projects/radway/ese-5700-fall/tmarhguy/project_1$
```

Open file in editor (cmd + click)

Figure 60: Vector check: Spectre reads `counter4b.vec`; the mismatch file `input_tran.vecerr` has 0 lines.

Debug note: the first runs left the bench's VSS net unconnected to the simulator ground, so the supply pair floated against the vector-driven inputs (nodes at 0.7–1.5 V, no counting); tying VSS to `gnd` fixed it. **`tb_controller_mac.`** `controller_mac` and `counter_4b` wired as in `top` (COUNT_EN → EN, CLEAR → counter RST), driven and checked by a 60-edge vector file (`controller.vec`, 1 ns clock): reset, START at edge 5, a full dot product, restart from DONE at edge 26, a START at edge 31 during RUN, RST at edge 35, and a second full run from edge 40. RUN, DONE, ACC_EN and COUNT are checked after every edge.

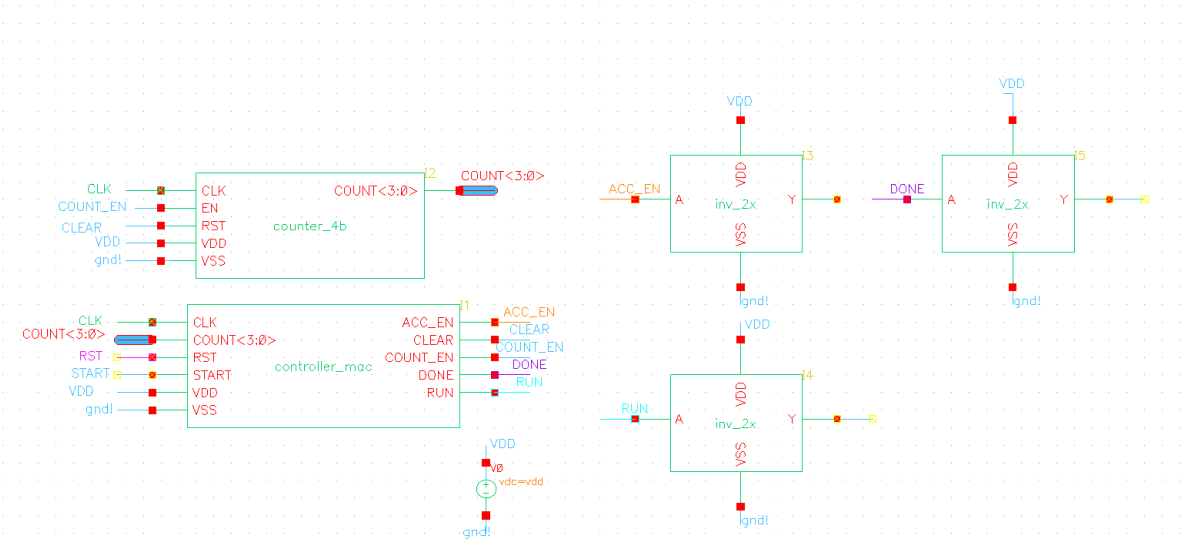


Figure 61: `tb_controller_mac`: `controller_mac` (I1) and `counter_4b` (I2) as in `top` (COUNT_EN → EN, CLEAR → counter RST, COUNT⟨3:0⟩ fed back); `inv_2x` loads on RUN, DONE and ACC_EN; $V_{DD} = 0.8$ V to `gnd!`. CLK, RST and START come from the vector file.

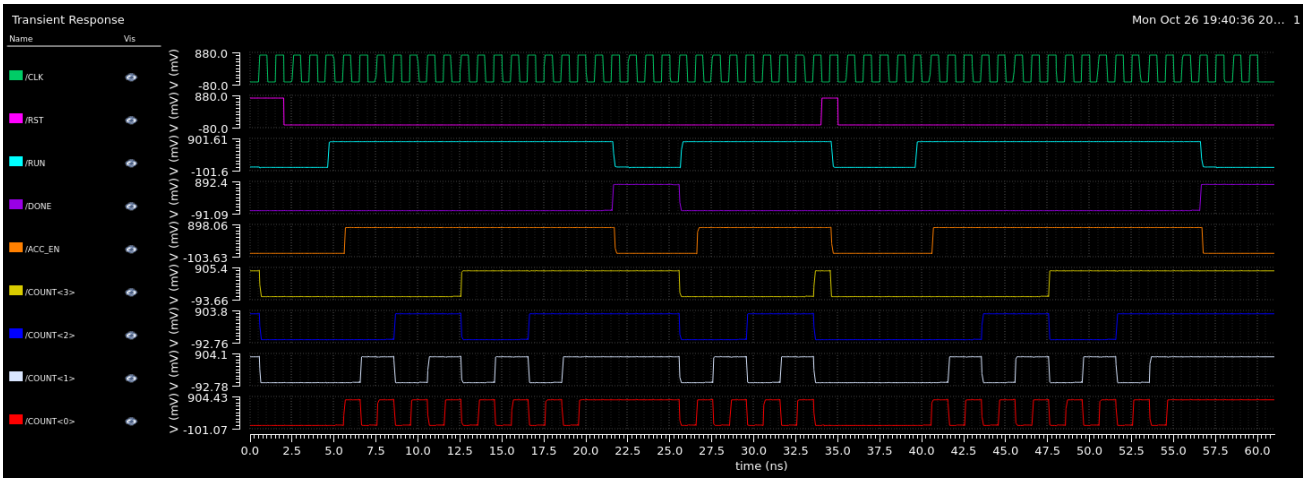


Figure 62: `tb_controller_mac`, 0–61 ns, with COUNT⟨3:0⟩: the counter steps 0→15 and holds during each run, clears on the START edge (25.5 ns) and on RST (34.5 ns). RUN rises after the START edge (4.5 ns), ACC_EN one cycle later (pipeline), and DONE after edge 22 (21.5 ns): **17 edges from START to DONE**. START at edge 26 restarts from DONE; START at edge 31 is ignored while RUN; RST at edge 35 aborts the run; the run from edge 40 ends with DONE at edge 57 and DONE holds.

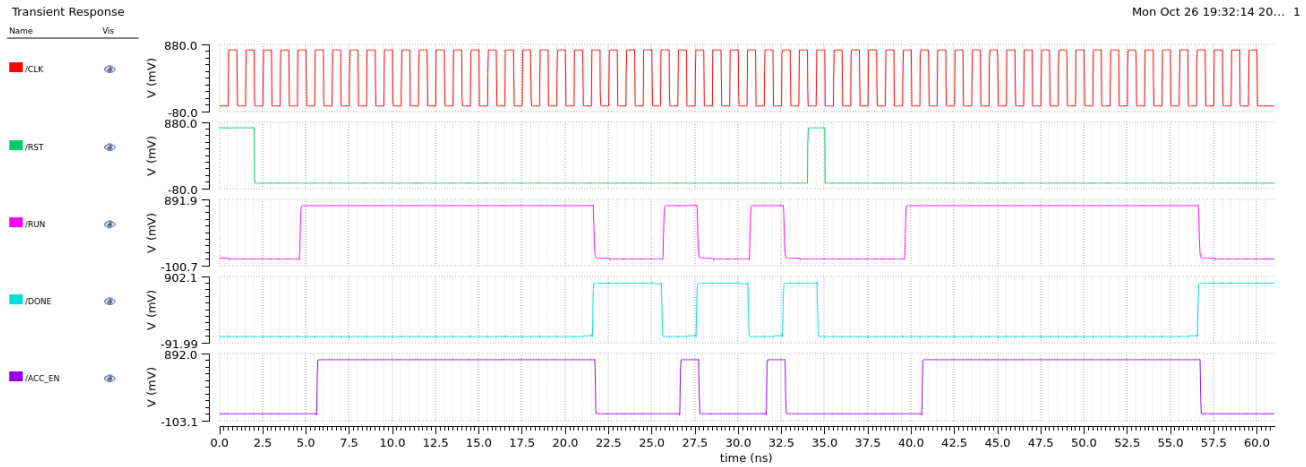


Figure 63: First run, with the counter's reset wired to RST instead of CLEAR: the restart from DONE lasts two cycles because the counter still holds 15 (LAST is already true). Wiring the counter reset to CLEAR, as in `top`, fixed it.

Result: all 60 edges match (`input_tran.vecerr` has 0 lines; Spectre read and processed `controller.vec`).

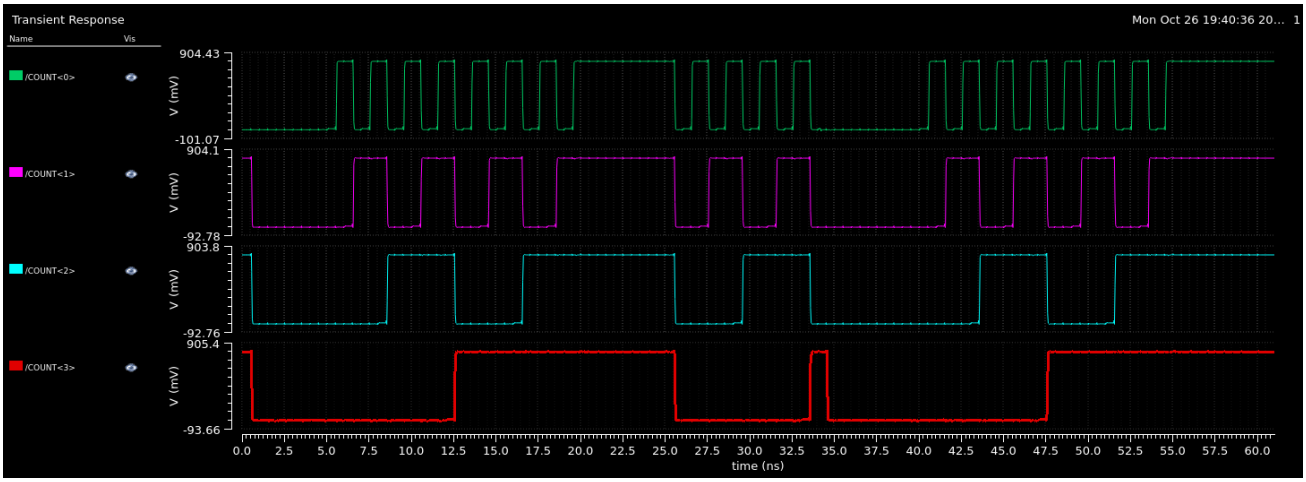


Figure 64: COUNT⟨3:0⟩ in `tb_controller_mac`: 0→15 and hold in each run, cleared at the restart (25.5 ns) and by RST (34.5 ns).

```
vwanjohi@epyc-6-9:/vast/projects/radway/ese-5700-fall/tmarhguy/project_1/High_Quality_Screenshots$ find $R -name "*.vecerr" |
 xargs wc -l
0 /vast/home/v/vwanjohi/simulation/proj1_lib/tb_controller_mac/maestro/results/maestro/ExplorerRun.0/1/proj1_lib_tb_controlle
r_mac_1/netlist/input_tran.vecerr
vwanjohi@epyc-6-9:/vast/projects/radway/ese-5700-fall/tmarhguy/project_1/High_Quality_Screenshots$
```

Figure 65: Vector check for `tb_controller_mac`: the mismatch file has 0 lines.

4.3 One-dot-product timing

After the 32 load cycles (16 writes to W, 16 to X), START is sampled on a rising CLK. Cycle 0 clears ACC and COUNT. Cycles 0–15 address the macros; VALID delays accumulate by one cycle, so $P[i]$ is added in cycle $i+1$. The last product is added in cycle 16; DONE rises on cycle 17 ($t_{dot} = 17 \times 2.5 = 42.5$ ns).

Cycle after START	COUNT	Memory	Accumulator / DONE
0	0 (clear)	read $W[0]$, $X[0]$	ACC = 0, VALID = 0
1	1	read [1]	first add: ACC ← $P[0]$
$k = 2 \ldots 15$	k	read $[k]$	ACC ← ACC +$P[k-1]$
16	15 (held)	ADDR held at 15	last add: ACC ← ACC +$P[15]$
17	15	—	DONE = 1; ACC holds

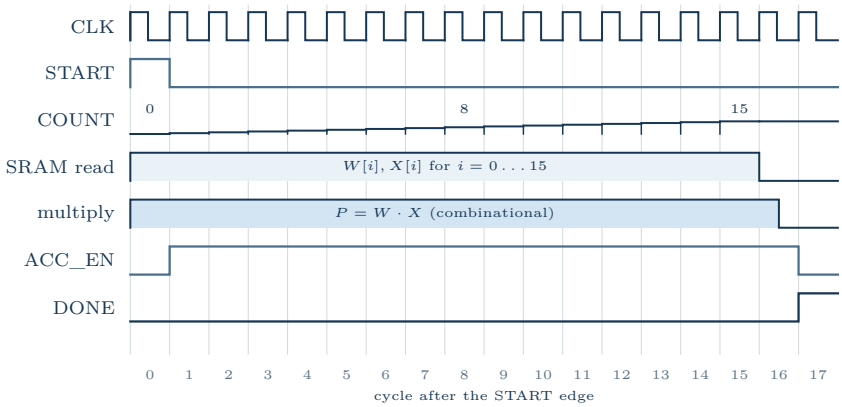


Figure 66: One dot product ($T_{CLK} = 2.5$ ns). START is sampled on the cycle-0 edge; COUNT walks 0–15 and holds; both macros are read in cycles 0–15; VALID delays ACC_EN so the accumulate occupies cycles 1–16; DONE rises on cycle 17. $t_{dot} = 42.5$ ns.

Architecture	pipelined: address and read in cycle n, MAC in n+1 (VALID)
Cycles START to DONE	17 edges after the START edge (`tb_controller_mac`, `tb_top`)
Cycles per dot product	17 (plus 32 load cycles when the memories are reloaded)
At $T_{CLK} = 2.5$ ns (selected)	$t_{dot} = 42.5$ ns (compute only; load excluded)

5. Timing closure

5.1 Critical path

The pipeline has no register between the macros and the multiplier: the macro's own address latch captures COUNT at the rising edge and its output latch holds DOUT for the rest of the cycle. The paths that must fit in one period are therefore (1) counter t_{cq} → `mux2` → macro address latch (short), and (2) CLK edge → SRAM access → multiplier → 12-bit adder → accumulator setup, the critical path: $T_{\min} \geq t_{\text{SRAM}} + t_{\text{mult}} + t_{\text{add}} + t_{su}$, with t_{SRAM} measured from CLK and so already including the macro's internal address latch.

$$T_{\text{min,non-pipe}} \geq t_{cq} + t_{\text{SRAM}} + t_{\text{mult}} + t_{\text{add}} + t_{su} + t_{\text{skew}},$$
$$T_{\text{min,pipe}} \geq \max\big(t_{cq} + t_{\text{SRAM}} + t_{su},\ t_{cq} + t_{\text{mult}} + t_{\text{add}} + t_{su}\big).$$

Term	Value	Source
t_{cq} (rise / fall)	68.51 / 62.18 ps	§2.2, 0.4 V markers, $C_Q = 1$ fF
t_{SRAM} (CLK to DOUT)	488 ps	Part 2, worst case at $T_{CLK} = 0.69$ ns
t_{mult}	1850 ps (LE); 1140–1240 ps measured	§3.4
t_{add} ($P_4 \rightarrow \text{ACC}_{11}$) + register input	$\approx 1500 + 100$ ps (LE)	§3.4
t_{su} (rise / fall)	40 ps / 40 ps	§2.3, 10% t_{cq} wall; 40 ps pass, 30 ps fail
Hand $T_{\min}$	**4.0** ns (LE); ≈ 3.3 ns with measured t_{mult}	$t_{\text{SRAM}} + t_{\text{mult}} + t_{\text{add}} + t_{su}$ (DOUT is the only pipeline stage)
Measured T_{CLK}	**2.5** ns (400 MHz)	shortest tested period where A, B and C all pass; B = 0xC10 at 2.0 ns

5.2 Period sweep

On `tb_top`, expected finals are Test A = 0x34D, Test B = 0xE10 (full-scale carries), Test C = 0x320. The period sweep is driven by Spectre digital vector files in `stimuli/timing/`, named `mac_top_<A|B|C>_<period>.vec` (Fig. 67): the same load-W, load-X, START, then 17-cycle compute sequence, scaled with the clock. The wall is Test B at 2.0 ns: A and C still pass there, B returns 0xC10, so the selected $T_{CLK} = 2.5$ ns. The interval between 2.0 and 2.5 ns was not swept. Files also exist at 3 ns and 6 ns; those periods are not in the reported table.

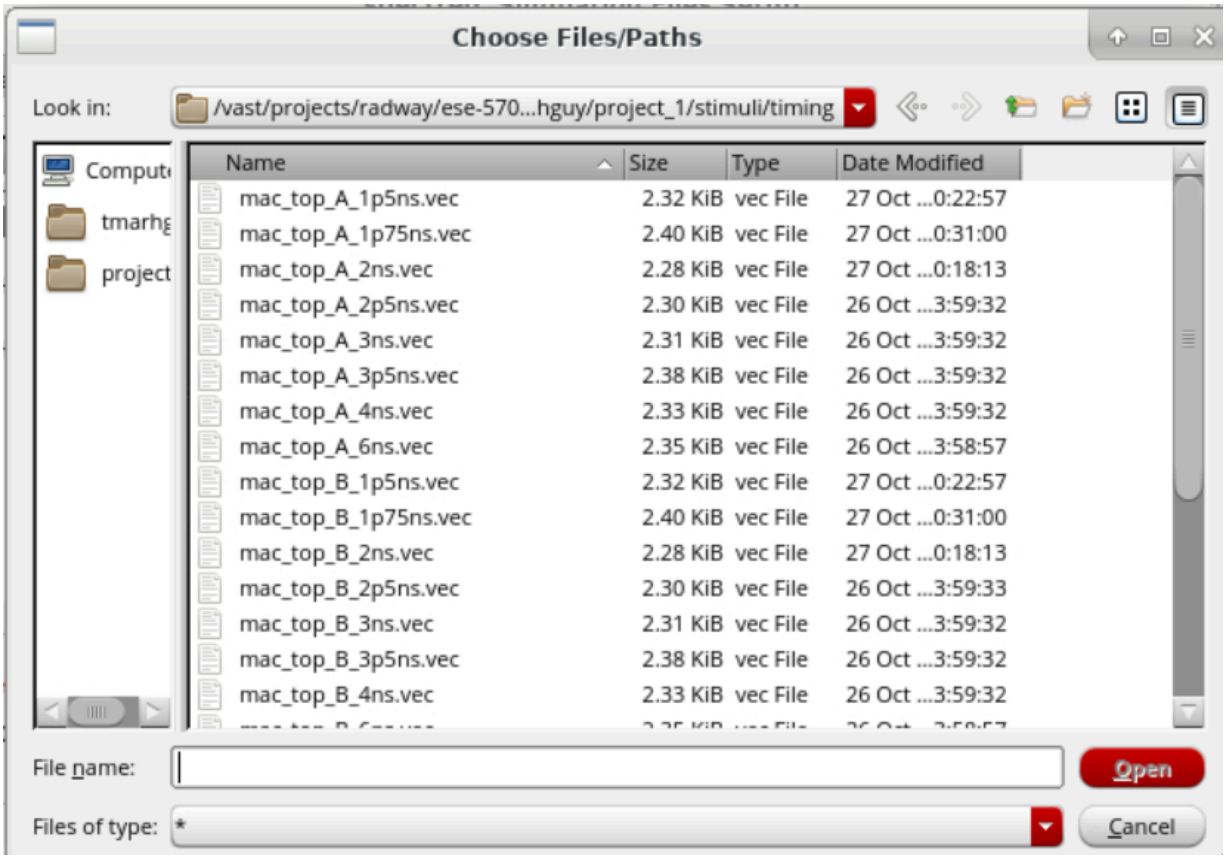


Figure 67: Vector files used on `tb_top`, `stimuli/timing/`. Visible: `mac_top_A_` and `mac_top_B_` at 1.5, 1.75, 2, 2.5, 3, 4 and 6 ns. The 10 ns functional runs used PWL sources, not these files. Energy at 2.5 ns uses `mac_top_A_2p5ns.vec`, `mac_top_B_2p5ns.vec` and the matching Test C file.

T_{CLK}	Test A	Test B	Test C	Result
10 ns	0x34D, pass	0xE10, pass	0x320, pass	A/B/C pass (§6.1)
4.0 ns	0x34D, pass	0xE10, pass	0x320, pass	A/B/C pass
2.5 ns	0x34D, pass	0xE10, pass	0x320, pass	**A/B/C pass (selected)**
2.0 ns	0x34D, pass	0xC10, fail	0x320, pass	**fail** (B)
1.75 ns	0x31D, fail	fail	fail	fail (A/B/C)
1.5 ns	0x13D, fail	fail	fail	fail (A/B/C)

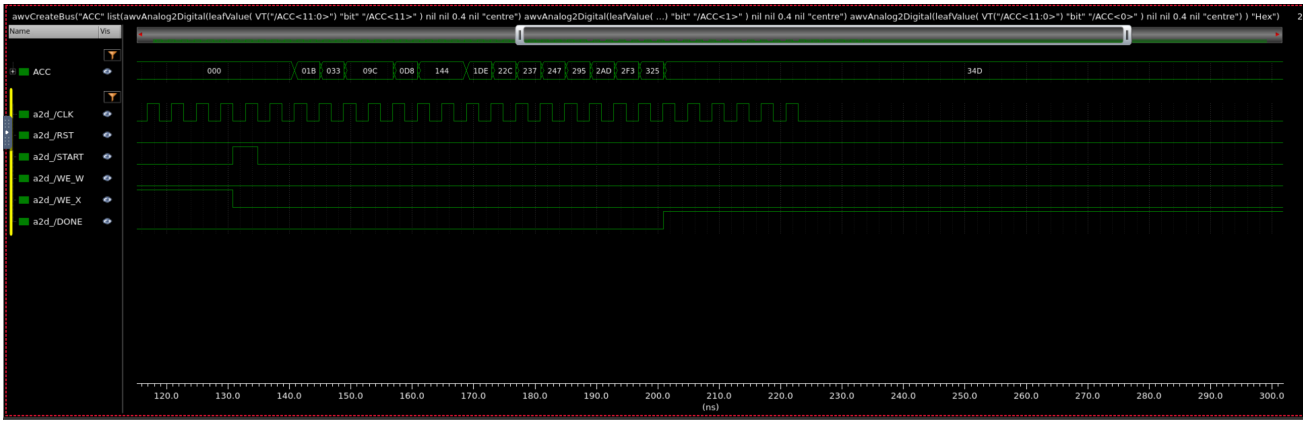


Figure 68: Test A at $T = 4.0$ ns (digital ACC bus): partial sums run 01B, 033, 09C, . . . , 325 and finish at **0x34D**. DONE rises after the compute window. Pass. Tests B and C also pass at 4.0 ns.

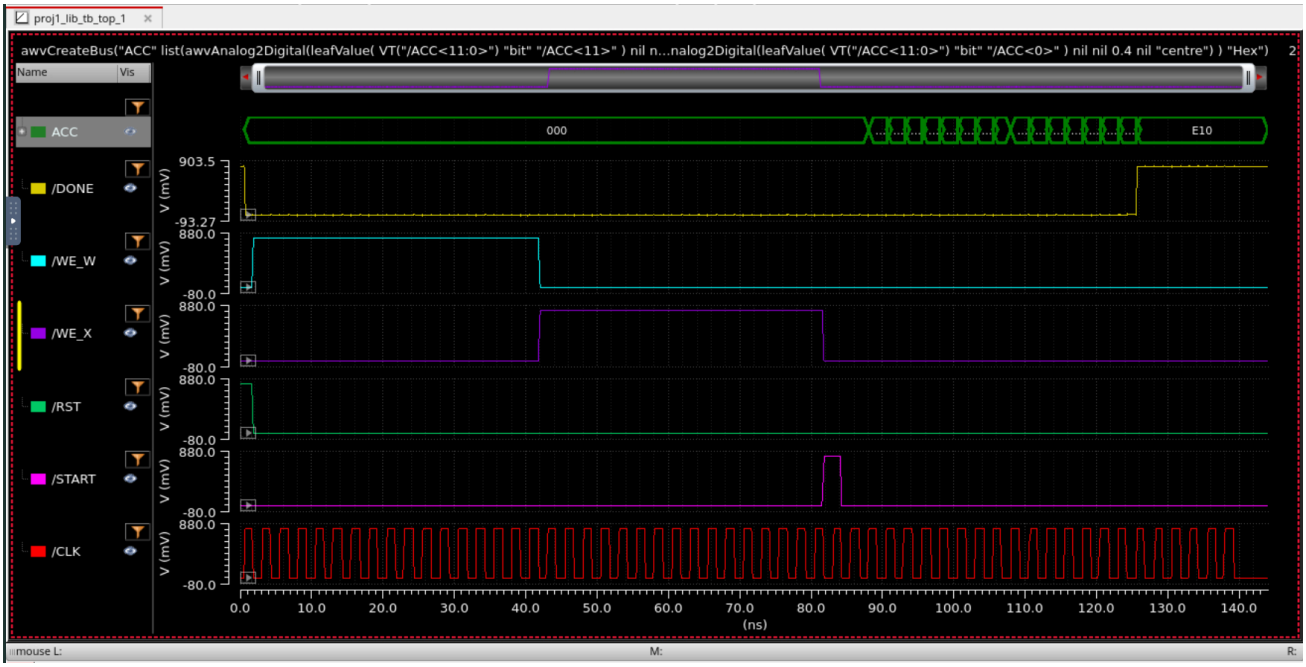


Figure 69: Test B at $T = 2.5$ ns: load W then X, START near 82 ns, ACC settles at **0xE10**, DONE high near 126 ns. Pass. Selected T_{CLK}.

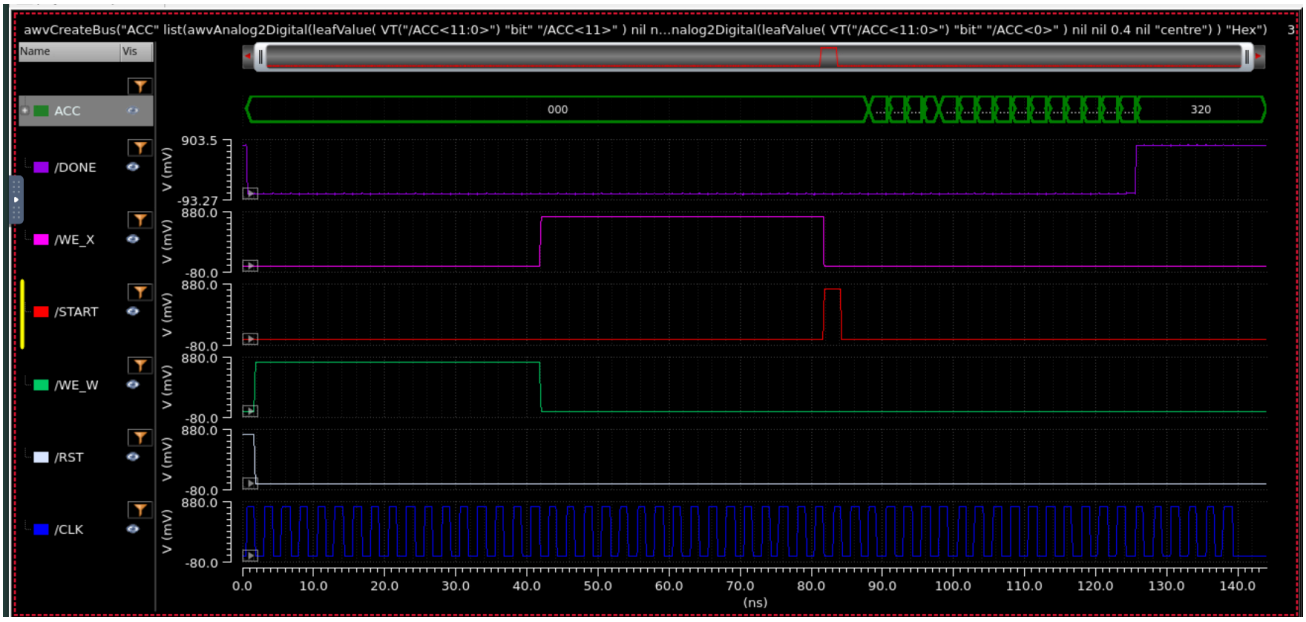


Figure 70: Test C at $T = 2.5$ ns: START near 82 ns, ACC settles at **0x320**, DONE high near 126 ns. Pass.

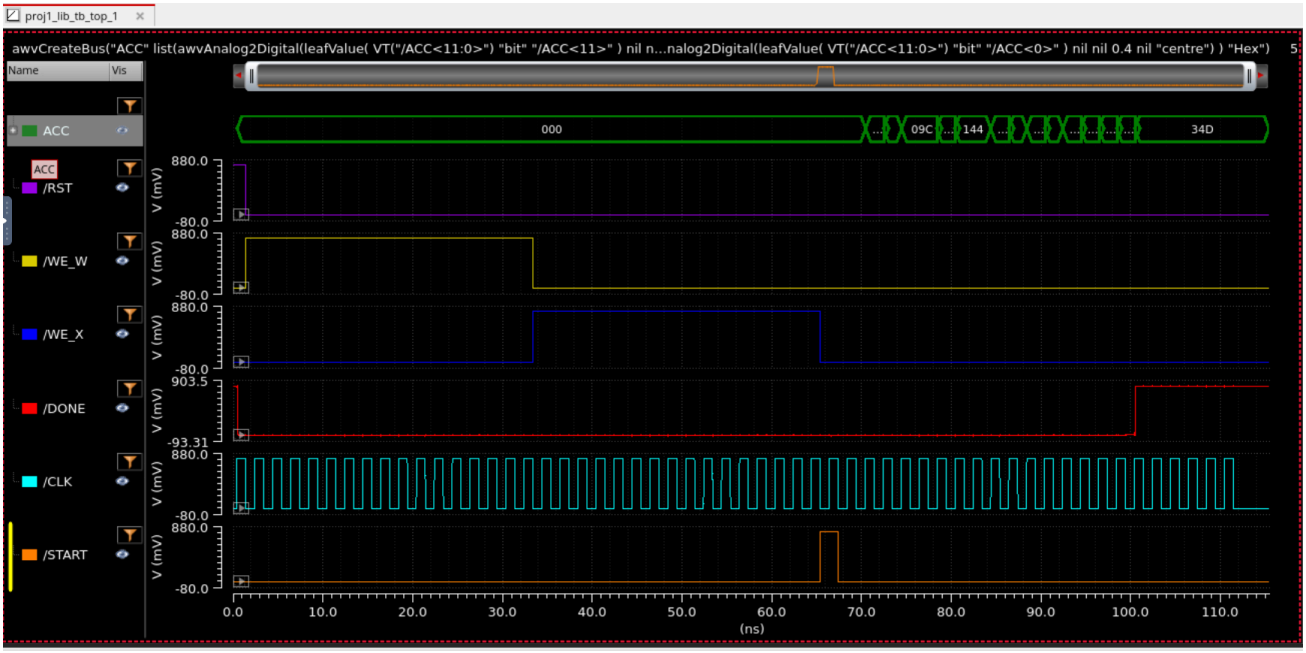


Figure 71: Test A at $T = 2.0$ ns: load W then X, START near 66 ns, ACC settles at **0x34D**, DONE high near 102 ns. Pass. Test C also passes at this period (0x320); Test B fails (0xC10).

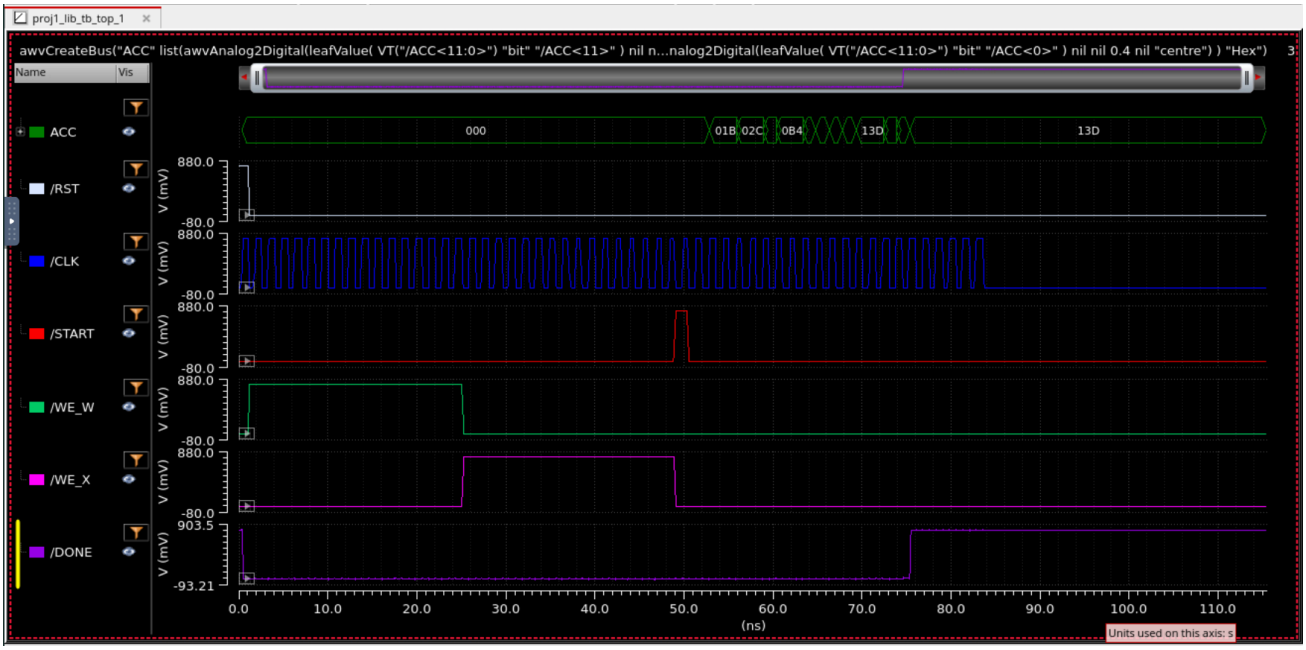


Figure 72: Test A at $T = 1.5$ ns: DONE still rises, but ACC stops at **0x13D** = 317 (expected 0x34D). Fail. Tests B and C also fail at 1.5 ns.

Hand LE (≈ 4.0 ns, or ≈ 3.3 ns with the measured multiplier) stacks SRAM, multiplier, adder and t_{su} worst cases; the measured 2.5 ns is faster because those arcs are not all worst-case in the same cycle. At 2.0 ns, Test B ($16 \times 15 \times 15$) returns 0xC10 = 3088 instead of 0xE10 = 3600 ($512 =$ 0x200 short). A (mixed products, 0x34D) and C (alternating 5×10, 0x320) still finish, so the failure tracks the longest arithmetic carry, not the sequencer: DONE still rises on the failing A runs at 1.75 and 1.5 ns. We do not isolate a single gate delay from the 2.0 ns B waveform.

5.3 Hold check

Two short paths: (1) counter → `mux2` → macro address latch, which closes at the same rising edge that changes the counter; (2) register-to-register in the controller (VALID from RUN through one `and2`). From §2.4, $t_{cq,\min} = 62.18$ ps and $t_h = 10$ ps (rise; fall ≤ 0 ps), so $t_{cq,\min} - t_h = 52$ ps of hold slack before mux/logic delay. No hold fail was seen on `tb_top` at 2.5 ns or 10 ns.

6. Verification

6.1 Test vectors

The DUT is `top`, instantiated in `tb_top`. $V_{DD} = 0.8$ V. Functional A/B/C at $T = 10$ ns use PWL files on RST, START, WE_W, WE_X, ADDR and DIN, with CLK a `vpulse`. The period sweep and the 2.5 ns energy runs use the Spectre vector files in `stimuli/timing/` (Fig. 67), generated by `make_mac_timing_vec.py`: one file per test and period, `mac_top_A_2p5ns.vec` and likewise for B/C and 1.5–6 ns. ACC⟨11:0⟩ is read as a hex bus at the 0.4 V threshold.

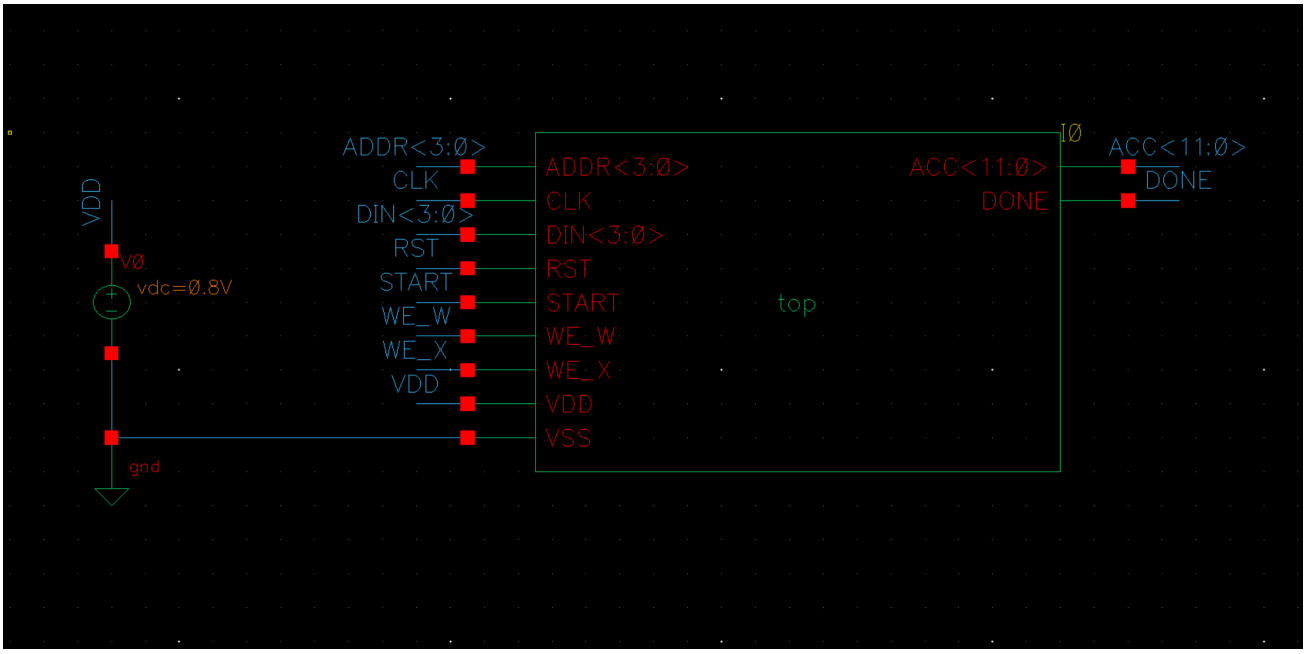


Figure 73: `tb_top`: `top` driven by CLK (V1) and V_{DD} = 0.8 V (V0); RST, START, WE_W, WE_X, ADDR⟨3:0⟩ and DIN⟨3:0⟩ from PWL files. Outputs ACC⟨11:0⟩ and DONE.

Test	$W[0..15]$	$X[0..15]$	Expected ACC
A	3, 12, 7, 0, 15, 9, 1, 14, 6, 11, 2, 13, 8, 5, 10, 4	9, 2, 15, 7, 4, 12, 0, 11, 13, 1, 8, 6, 3, 14, 5, 10	845 (0x34D)
B	all 15	all 15	3600 (0xE10)
C	5, 10, 5, 10, . . .	10, 5, 10, 5, . . .	800 (0x320)

Test	Expected	ACC	DONE	Cyc.	Result
A	845 (0x34D)	0x34D	edge 17	17	**pass** at 10/4/2.5/2.0 ns; fail 1.75 (0x31D), 1.5 (0x13D)
B	3600 (0xE10)	0xE10	edge 17	17	**pass** at 10/4/2.5 ns; fail 2.0 ns (0xC10), 1.75 ns and 1.5 ns
C	800 (0x320)	0x320	edge 17	17	**pass** at 10/4/2.5/2.0 ns; fail 1.75 ns and 1.5 ns

The 10 ns rows are the original PWL functional runs (DONE at 502 ns). The 4.0–1.5 ns rows use the scaled `.vec` files. Test B is the full-scale case (largest carry chains, no overflow); Test C alternates 5×10 and 10×5 to maximize toggling on the memory outputs and multiplier inputs. Selected operating point: all three pass at 2.5 ns.

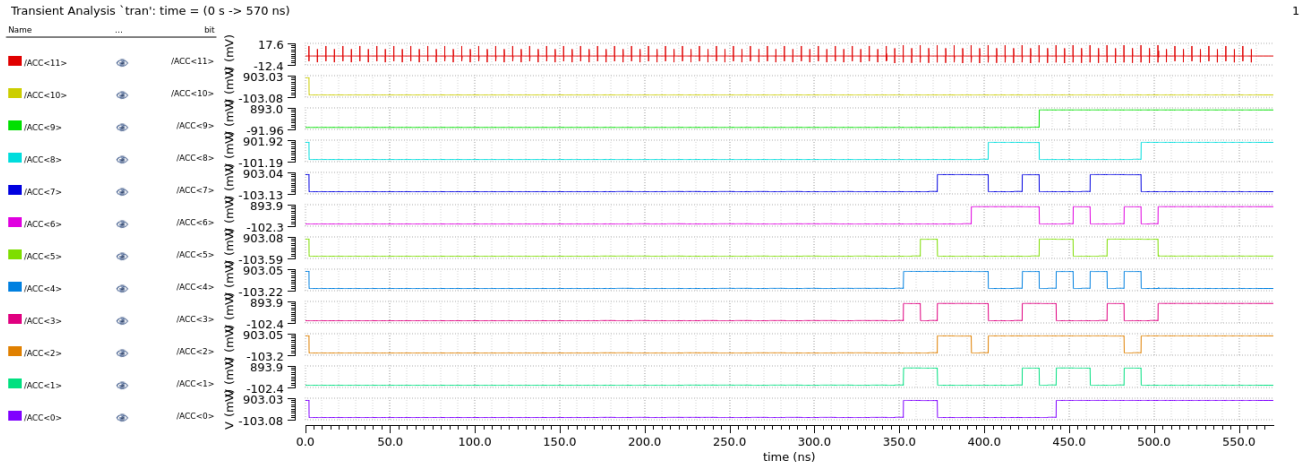


Figure 74: Test A ($T = 10$ ns), ACC⟨11:0⟩: START edge at 332 ns; first accumulate at 352 ns (0x01B = 27), then 51, 156, 156, 216, 324, 324, 478, 556, 567, 583, 661, 685, 755, 805 and **0x34D** = 845 at 502 ns, when DONE rises (17 edges after START); ACC holds until the next START or RST.

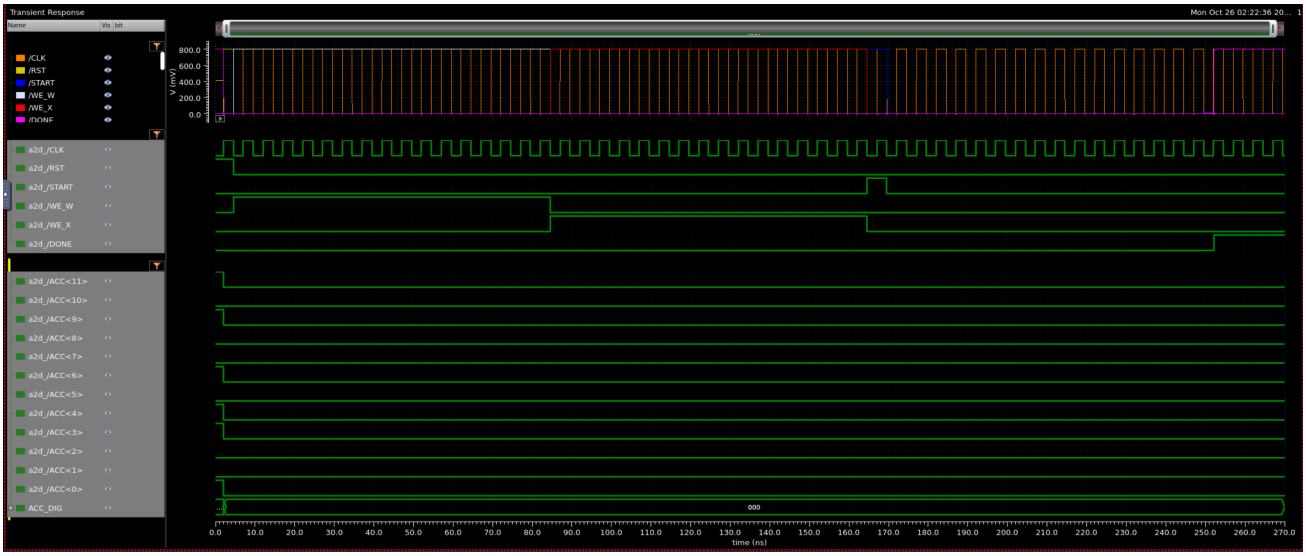


Figure 75: Earlier Test A run, before the read-latch fix: the sequencer finishes (DONE) but ACC stays 0x000.

6.2 Read-latch fix

The first runs of `tb_top` read ACC = 0 although the sequencer was correct (DONE 17 edges after START). Probing inside `top`: WE, ADDR and DIN reach both macros correctly, the wordlines fire and the read inverter output RD rises after each read, but both DOUT buses stay at 0. The macro's output latch was enabled by a pulse derived from the delayed wordline enable (RD_EN = WLD·$\overline{\text{WE_L}}$, about 150 ps wide), which closed before RD had crossed mid-rail, so the latch node peaked near 0.4 V and fell back. RD_EN is now level-sensitive on the clock-high phase (RD_EN = CLKBUF·$\overline{\text{WE_L}}$, one input of `I11` in `controller`): the latch follows RD for the whole high phase and closes about 40 ps before precharge pulls RD down after the falling edge. The Part 2 `tb_sram_macro` was not re-run with that RD_EN change; `tb_top` Tests A–C after the fix are the check.

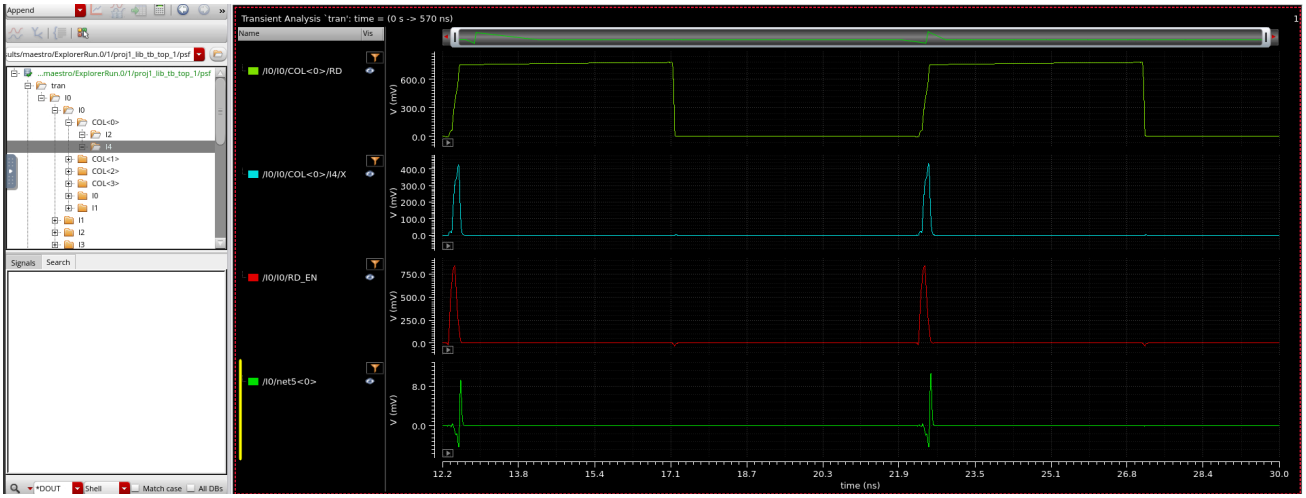


Figure 76: Before the fix: RD rises after the read, but the RD_EN pulse closes first; the latch node X peaks near 0.4 V and returns to 0, so DOUT stays 0.

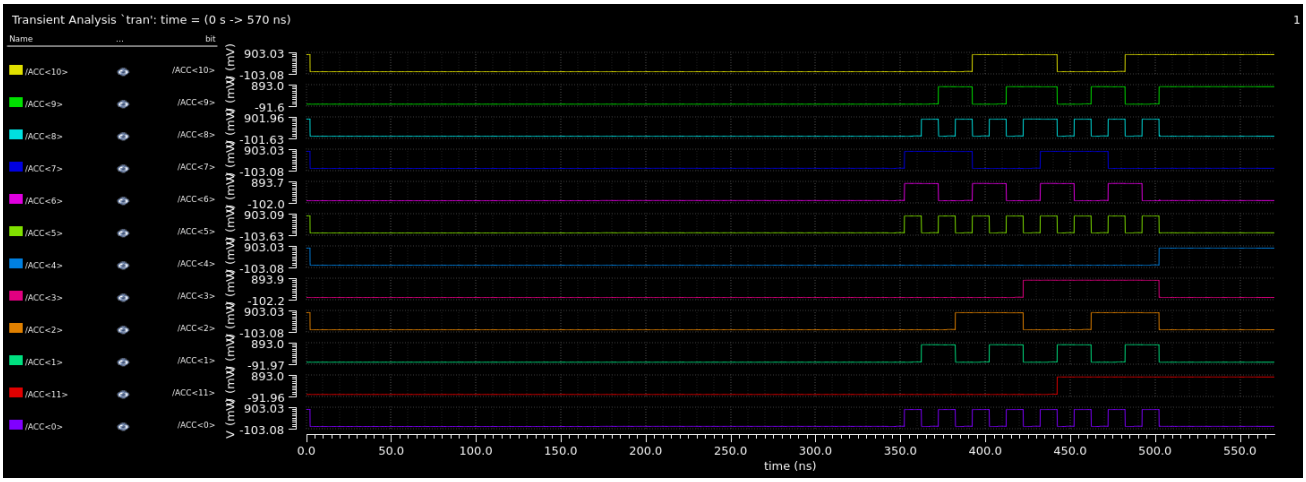


Figure 77: Test B after the fix ($T = 10$ ns): ACC$\langle 11{:}0 \rangle$ steps by 225 from 352 ns (0x0E1, 0x1C2, ...) and reaches 0xE10 = 3600 at 502 ns, when DONE rises; ACC holds afterwards.

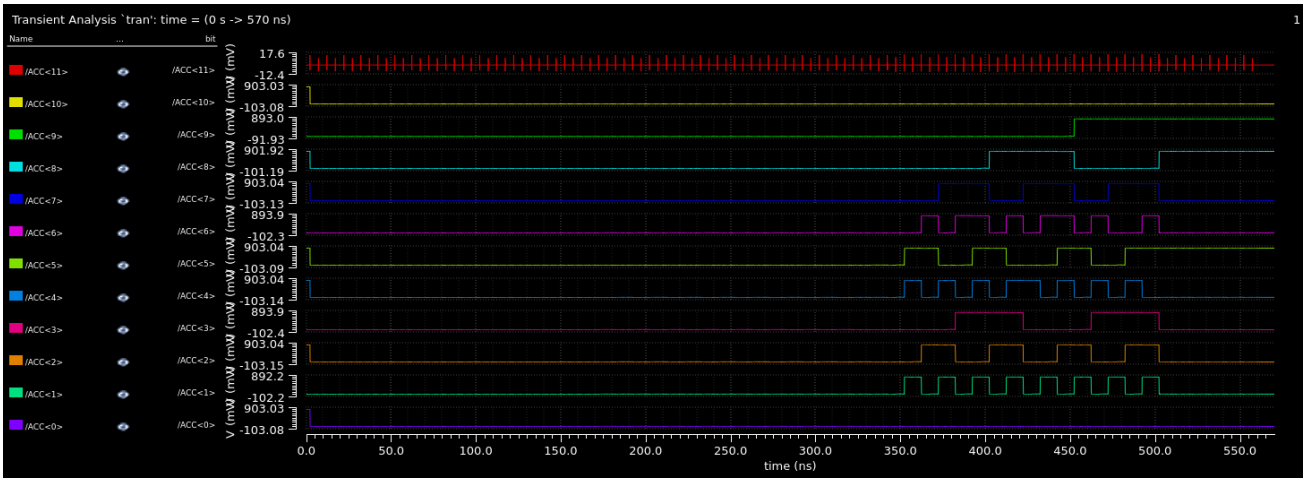


Figure 78: Test C ($T = 10$ ns): ACC steps by 50 from 352 ns (0x032, 0x064, ...) to 0x320 = 800 at 502 ns. The memory outputs alternate 0101/1010 every cycle, the maximum-toggling case.

RST mid-run, START during COMPUTE, and back-to-back dots without reload were checked on the controller bench (60/60) and are not repeated on `tb_top`.

7. Metrics and figure of merit

7.1 Reported metrics

At $V_{DD} = 0.8$ V, nominal, 27°C. Selected $T_{CLK} = 2.5$ ns (400 MHz): A/B/C all pass; B fails at 2.0 ns (0xC10). Energy and FOM use that same period. Headline figures are two significant digits; the table keeps the measured digits.

Metric	Value
V_{DD}	0.8 V (nominal, 27°C)
Density factor D	28 (one fin = 0.0242 μm^2)
T_{CLK} (selected)	**2.5** ns (400 MHz)
Hand $T_{\min}$ (LE)	$\approx$ 4.0 ns (3.3 ns with measured multiplier)
N_{cycles} (START to DONE)	17 (`tb_controller_mac`, `tb_top`)
t_{dot}	$17 \times 2.5 =$ **42.5** ns
SRAM W / X (Part 2)	$54.6 + 54.6 = 109.2$ μm^2 *(est.)*
Datapath	5696 fins $\times 0.0242 = 137.8$ μm^2 *(est.)*
Sequencer and glue	1848 fins $\times 0.0242 = 44.7$ μm^2 *(est.)*
Total area	$109.2 + 137.8 + 44.7 =$ **291.8** μm^2 *(est.)*
Test A E_{dot} (with memory)	8.870 pJ (0.5544 pJ/MAC)
Test A without memory	0.38 pJ/MAC *(est.)* (§7.3)
FOM$_{MAC}$	$\mathbf{1.1 \times 10^5}$ μm^2·pJ·ns
(Area·$E_{dot} \cdot t_{dot}$)	$291.8 \times 8.870 \times 42.5 = 1.100 \times 10^5$

7.2 Energy

From the single VDD source on `tb_top` at $T = 2.5$ ns (vectors `mac_top_A_2p5ns.vec`, `mac_top_B_2p5ns.vec`, `mac_top_C_2p5ns.vec`). External vector-driver and CLK-source energy are excluded. The sign convention is $-V_{DD} \int I_{V0+}\,\mathrm{d}t$: Spectre's device current I_{V0+} leaves the positive terminal of V0, so the chip energy is the negative of that integral.

$$E_{dot} = -V_{DD} \int_{82\,\mathrm{ns}}^{126\,\mathrm{ns}} I_{V0+}\,\mathrm{d}t = -0.8 \int_{82\,\mathrm{ns}}^{126\,\mathrm{ns}} I(\texttt{/V0/PLUS})\,\mathrm{d}t.$$

ViVA calculator (result in pJ):

```
-0.8 * value(integ(clip(IT("/V0/PLUS") 82n 126n)) 126n) * 1e12
```

The clip 82–126 ns is 44 ns. That is the recorded Cadence window, not the latency:

$$t_{dot} = N_{cycles}\, T_{CLK} = 17 \times 2.5\,\mathrm{ns} = 42.5\,\mathrm{ns}.$$

On the 2.5 ns analog plots, START is the pulse near 82 ns and DONE rises near 126 ns, so the integral covers the compute phase plus about 1.5 ns of post-DONE hold. The 0.4 V crossings were not re-marked in the calculator; FOM uses $t_{dot} = 42.5$ ns and the measured E_{dot} below.

Test	ACC	E_{dot} (meas.)	$E_{\mathrm{MAC}} = E_{dot}/16$	ViVA
A	0x34D = 845	**8.870** pJ	0.5544 pJ/MAC	8.870
B	0xE10 = 3600	15.29 pJ	0.9556 pJ/MAC	15.29
C	0x320 = 800	9.500 pJ	0.5938 pJ/MAC	9.500

Each dot product is 16 MACs. FOM uses Test A.

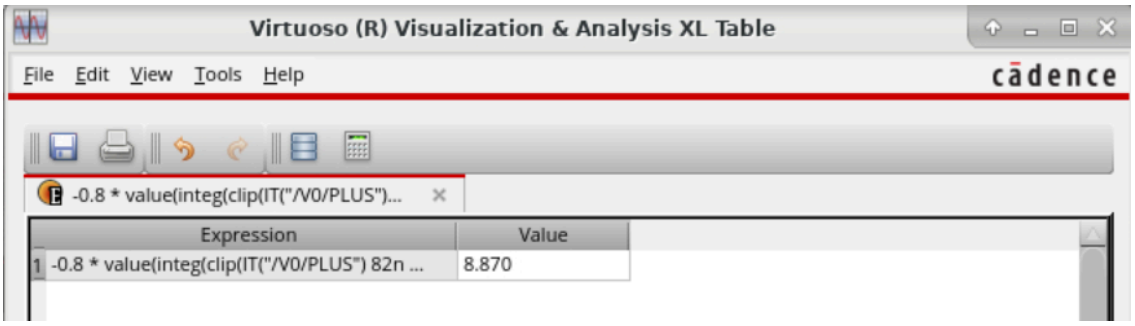


Figure 79: Test A energy at $T = 2.5$ ns: ViVA `-0.8 * value(integ(clip(IT("/V0/PLUS") 82n 126n)) 126n) * 1e12` = 8.870 pJ. VDD pin V0 only.

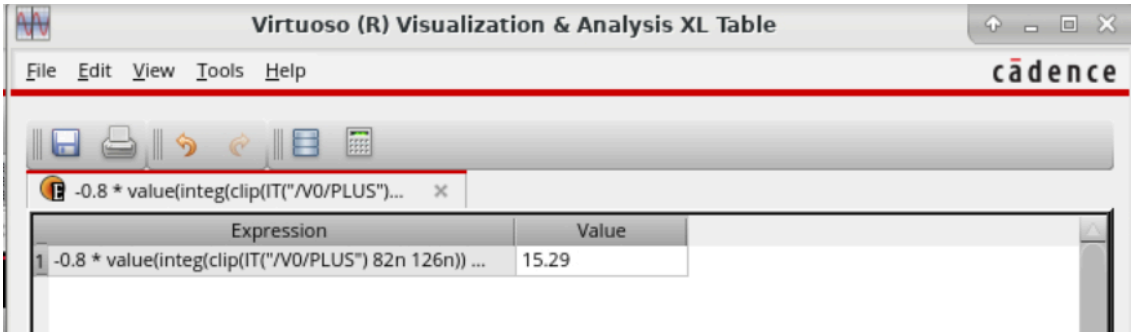


Figure 80: Test B energy at $T = 2.5$ ns: same 82–126 ns window, 15.29 pJ. VDD only.

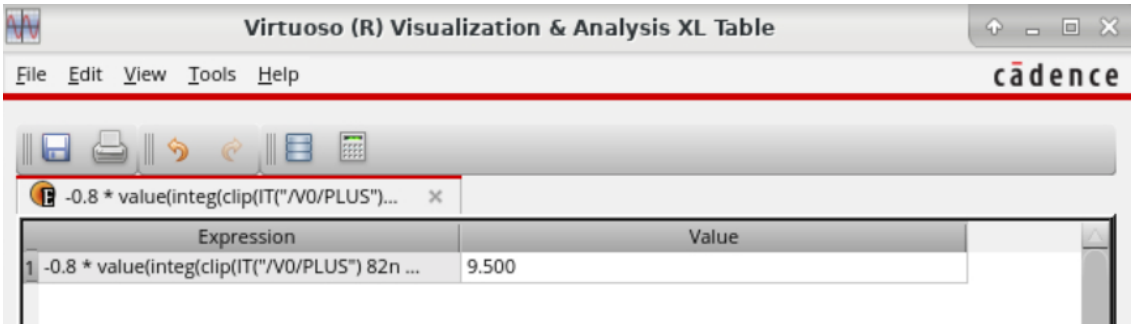


Figure 81: Test C energy at $T = 2.5$ ns: 9.500 pJ. VDD only.

B is highest because every cycle is $15 \times 15 = 225$ with the longest adder carries. C alternates 5×10 and 10×5 (more toggling than A on the memory outputs, but smaller products). A is the lowest of the three (8.870 pJ).

7.3 Memory vs. datapath energy

`tb_top` has one VDD pin (V0) feeding both SRAM macros and the MAC logic, and this environment has no Cadence session in which to probe hierarchical instance currents, so the split below is an estimate from the Part 2 macro measurements, not a second Spectre integral. It is labeled *(est.)* throughout.

Part 2 measured the macro's own VDD at $T_{CLK} = 0.69$ ns: checkerboard read $P_{\text{read}} = 115.2\ \mu$W, clock-stopped leakage $P_{\text{leak}} = 3.99\ \mu$W, clock-running idle 112.2 μW. Dynamic energy per access is then

$$E_{\text{dyn}} = (P_{\text{read}} - P_{\text{leak}})\, T_{CLK,\text{SRAM}} = (115.2 - 3.99)\,\mu\text{W} \times 0.69\,\text{ns} = 76.7\,\text{fJ}.$$

At the MAC's 2.5 ns period the same access still costs about E_{dyn}, plus leakage over the longer cycle:

$$E_{\text{leak}} = P_{\text{leak}} \times 2.5\,\text{ns} = 3.99\,\mu\text{W} \times 2.5\,\text{ns} = 10.0\,\text{fJ},$$
$$E_{\text{macro/cyc}} \approx 76.7 + 10.0 = 86.7\,\text{fJ}.$$

Both macros are read in each of the 16 compute addresses:

$$E_{\text{mem}} \approx 2 \times 16 \times 86.7\,\text{fJ} = 2.77\,\text{pJ}\ \textit{(est.)}.$$

Bitline restore alone is much smaller:

$$E_{BL} = 2 \times 16 \times 4 \times C_{BL} V_{DD}^2 = 2 \times 16 \times 4 \times 3.5\,\text{fF} \times (0.8\,\text{V})^2 = 0.29\,\text{pJ}.$$

The rest of E_{mem} is the Part 2 delay-chain controller, precharge and output latches, which fire on every clock (no chip-select). That matches Part 2: clock-running idle (112 μW) is almost the read power (115 μW), so SRAM energy is nearly pattern-independent.

Subtracting from the measured totals:

Test	E_{dot} (meas.)	E_{mem} *(est.)*	E_{logic} *(est.)*	E_{MAC} w/o mem. *(est.)*
A	8.870 pJ	2.77 pJ	6.10 pJ	0.38 pJ/MAC
B	15.29 pJ	2.77 pJ	12.52 pJ	0.78 pJ/MAC
C	9.500 pJ	2.77 pJ	6.73 pJ	0.42 pJ/MAC

Where the energy goes, Test A: about one-third the two clocked SRAM macros (2.77/8.87), two-thirds the array multiplier, 12-bit adder, registers and sequencer (6.10/8.87). The 6.4 pJ gap from A to B sits in E_{logic} (full-scale carries every cycle; W and X are constant, so memory activity is no larger than A). The 0.63 pJ gap from A to C is the extra DOUT and multiplier-input toggle of the 5/10 checkerboard, a few tenths of a picojoule on top of the clocked SRAM overhead.

FOM_{MAC} uses the measured Test A total at $T_{CLK} = 2.5$ ns:

$$\text{FOM}_{MAC} = \text{Area} \cdot E_{dot} \cdot t_{dot} = 291.8\,\mu\text{m}^2 \times 8.870\,\text{pJ} \times 42.5\,\text{ns} = 1.100 \times 10^5\,\mu\text{m}^2 \cdot \text{pJ} \cdot \text{ns},$$

reported as 1.1×10^5.

8. What limited the design

The selected point is a 17-cycle MAC around two Part 2 macros. Tests A, B and C all pass at 2.5 ns; at 2.0 ns only B fails (0xC10 instead of 0xE10).

Metric	Selected value
T_{CLK} / frequency	2.5 ns / 400 MHz
t_{dot} (START to DONE)	$17 \times 2.5 = 42.5$ ns
Test A E_{dot} (with memory)	8.870 pJ (0.55 pJ/MAC)
Test A without memory	6.10 pJ (0.38 pJ/MAC) *(est.)*
Area ($D = 28$)	292 μm^2 *(est.)*
FOM$_{MAC}$	1.1×10^5 μm^2·pJ·ns

8.1 Delay

The period is set by the multiply-accumulate path, not the memory. The array multiplier's worst case is about 1.2 ns—over twice the 488 ps SRAM read—before the 12-bit ripple adder and $t_{su} = 40$ ps. Both arithmetic blocks are ripple structures from one small full adder (2-fin `xor2` stacks with input inverters): easy to verify, long in delay.

8.2 Area and energy

The datapath (138 μm^2) is larger than the two macros (109 μm^2 *(est.)*), mostly the 24 full adders and the 12 accumulator registers. SRAM energy is dominated by the clocked Part 2 controller (no chip-select), not by bitline swing, so it barely moves from A to B; the 6.4 pJ extra on Test B is the full-scale carry path.

8.3 Choices we kept

- **Register.** Static TG master-slave, 100 fins per flip-flop (152 with enable and reset), reused everywhere. A dynamic latch would be smaller and faster but cannot hold with the clock stopped (IDLE and DONE do). Measured $t_{cq} = 68.51/62.18$ ps, $t_{su} = 40$ ps, $t_h = 10/ \leq 0$ ps, $C_{CLK} = 2.822$ fF, $E = 3.895$ fJ idle / 9.309 fJ toggling.
- **Pipeline.** Overlapping the read with the multiply-accumulate costs one cycle (17 instead of 16) and stays within the 20-cycle limit. The macro output latch is the only pipeline stage, so SRAM access and arithmetic still share one period. A register after the multiplier would split them, at a second extra cycle (18).
- **Arithmetic.** Ripple array and ripple adder: smallest design effort and regular wiring, but the slowest choice. We kept them because they reused one verified full adder and left the macros as the Part 2 black box.

A carry-save multiplier, a faster final adder, or a pipeline register after the multiplier would cut the period. A chip-select on the macros would cut the clocked memory energy.