ESE 5700 Project 1, Part 3: The MAC Unit

Panos Dimtsoudis (panosdim) Dhruva Jindal (djindal)

10/27/26

Team and contributions

Task	panosdim	djindal
3.1 Register `diffr`: design, setup/hold, power benches	X	
3.2 Datapath: `fa_m`, `and2_u`, `mult4`, `add12` and their unit tests	X	
3.3 Sequencer `seq`: logic design and schematic	X	
3.3 Sequencer and mux unit tests (`tb_seq`, `tb_mux`), `mux2_u`	X	
3.3 Top level `mac` and full bench `tb_mac`		X
3.4 $T_{CLK,min}$ sweep, clock driver, hold analysis		X
3.5 Tests A, B, C		X
Energy and area	X	
Report	X	X

AI Usage Disclosure

Claude was used to help organize the report and to generate tables, figure placeholders, and figure captions. It was also used to write python scripts that produced plots from CSV data obtained from **our** simulations. All circuit designs, simulations, extracted data, and reported results were produced and verified by the authors.

Summary of results

Unless stated otherwise, values are rounded to two significant figures at $V_{DD} = 0.8\,\text{V}$. Full-MAC results come from `tb_mac` with the extracted SRAM arrays. Area is reported to three significant figures.

Quantity	Value	Section
Supply V_{DD} / density factor D	0.8 V / 79	–
Register: t_{pcq} / t_{ccq} (ps)	94 / 71	2
Register: t_{setup} / t_{hold} (ps)	87 / −49	2
Register: clock-input capacitance (fF)	1.5 (D held) / 1.9 (D toggling)	2
Register: energy per clock (fJ)	1.8 (D held) / 6.2 (D toggling)	2
Full adder carry delay per bit (ps)	67	3
Multiplier critical path: LE estimate / measured (ps)	760 / 890	3
Accumulator ripple: estimate / measured (ps)	870 / 885	3
Cycles from START to DONE (limit 20)	19	4, 6
$T_{CLK,min}$: hand estimate / measured (ns)	1.1 / 0.90	5
Hold margin, counter to SRAM address latch (ps)	90	5
Tests A, B, C at $T_{CLK,min}$	all pass (845, 3600, 800)	6
$t_{dot} = 19 \times T_{CLK,min}$ (ns)	17	Metrics
E_{dot}, Test A: MAC supplies / with clock (pJ)	6.2 / 8.1	Metrics
E_{dot}, MAC supplies, Test B / C (pJ)	5.5 / 6.2	Metrics
Energy per MAC, Test A, with / without memory (pJ)	0.39 / 0.19	Metrics
Total area (μm^2)	505	Metrics
FOM_{MAC}, with clock ($\mu m^2 \cdot pJ \cdot ns$)	7.0×10^4	Metrics
FOM_{MAC}, MAC supplies only	5.3×10^4	Metrics

1 Architecture

The unit computes $\mathrm{ACC} = \sum_{i=0}^{15} W[i]\,X[i]$ for 4-bit unsigned $W[i]$ and $X[i]$. The top-level cell `mac` (Figures 1 and 2) contains:

- **Memories:** two copies of the Part 2 macro, `MEMW` and `MEMX`. They share the address `MA<0:3>` and the data input `DIN<3:0>`; their outputs are `DW` and `DX`.
- **Address mux:** four `mux2_u` (`MX<0:3>`) select the external `ADDR` while the unit is idle and the counter `CNT` while it computes (`MA` $= S\,?\,$`CNT` $:$ `ADDR`).
- **Write gating:** two `and2_u` (`GW`, `GX`) pass `WE_W` and `WE_X` only while the unit is idle (`WE_W` $\cdot \bar{S}$), so a stray write enable cannot corrupt a memory during a dot product.
- **Datapath:** operand registers `RW<0:3>`, `RX<0:3>` (`W`, `X`); the multiplier `MUL` (`mult4`, `PM<0:7>`); the product register `RP<0:7>` (`PR`); the adder `ADD` (`add12`, `SUM` $=$ `ACC` $+$ `PR`); and the accumulator register `RA<0:11>` (`ACC`).
- **Sequencer:** `SEQ` (`seq`, Section 4), with the address counter, the state flop and `DONE`.

Every state element is the register `diffr` of Section 2: 28 in the datapath and 9 in the sequencer, 37 in all.

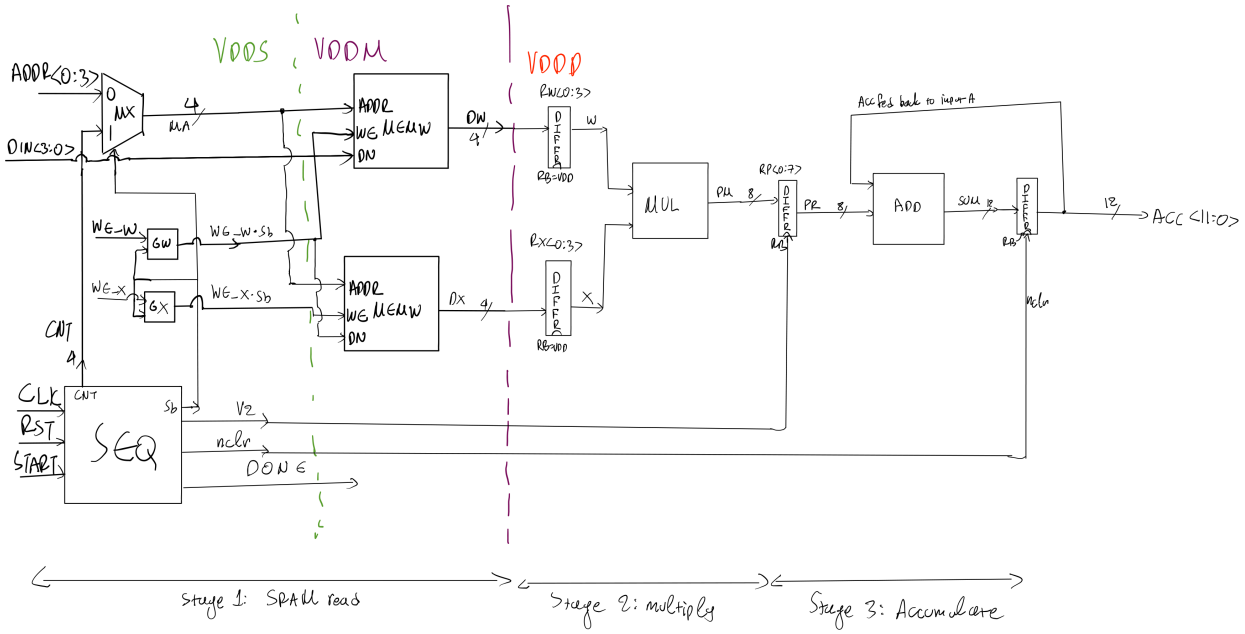


Figure 1: Block diagram of `mac`, with the three pipeline stages marked along the bottom. The dashed vertical lines divide the three supply domains (VDDS, VDDM, VDDD). Each register is a bank of `diffr`; the number on each bus is its width. RW and RX have RB tied to VDD, RP is reset by `V2` and RA by `nclr`. All registers and both macros share one CLK node (not drawn).

Pipeline. A read in the macro takes $t_{access} = 442\,\text{ps}$ (Part 2), and the multiplier and the accumulator adder each take close to 0.9 ns (Section 3). Putting the read, the multiply and the add in one cycle would need about $442 + 890 + 885 + 87 \approx 2.3\,\text{ns}$ (the read is timed from the CLK edge at the macro, so only the capturing register's t_{setup} is added), so the unit is pipelined in three stages (Figure 1):

1. **SRAM read:** the macros' input latches sample address k at a rising edge, and $W[k]$, $X[k]$ appear on `DOUT` within the same cycle; the operand registers `RW`, `RX` capture them at the next edge.
2. **Multiply:** `MUL` forms the product of `W` and `X`, captured by `RP`.
3. **Accumulate:** `ADD` adds `PR` to `ACC`, captured by `RA`.

With E_0 the edge that samples `START`, the macros sample address k at E_{k+1}, the operand registers capture word k at E_{k+2}, the product register captures $W[k]X[k]$ at E_{k+3} and `ACC` adds it at E_{k+4}. The last product ($k = 15$) is added at E_{19}, and `DONE` rises right after E_{19}: **19 cycles** from `START` to `DONE`, within the limit of 20. Splitting multiply and accumulate into separate stages costs one cycle of latency and lets the clock run at the slower of the two instead of their sum.

Supply domains. The `mac` cell has three supply pins so that the energy of each part can be measured separately: **VDDM** for both macros, **VDDD** for the datapath (`RW`, `RX`, `MUL`, `RP`, `ADD`, `RA`) and **VDDS** for the sequencer, address muxes and write gating (`SEQ`, `MX`, `GW`, `GX`). In the bench, the input drivers, output loads and clock driver run from a fourth supply, VDDTB. All are at 0.8 V.

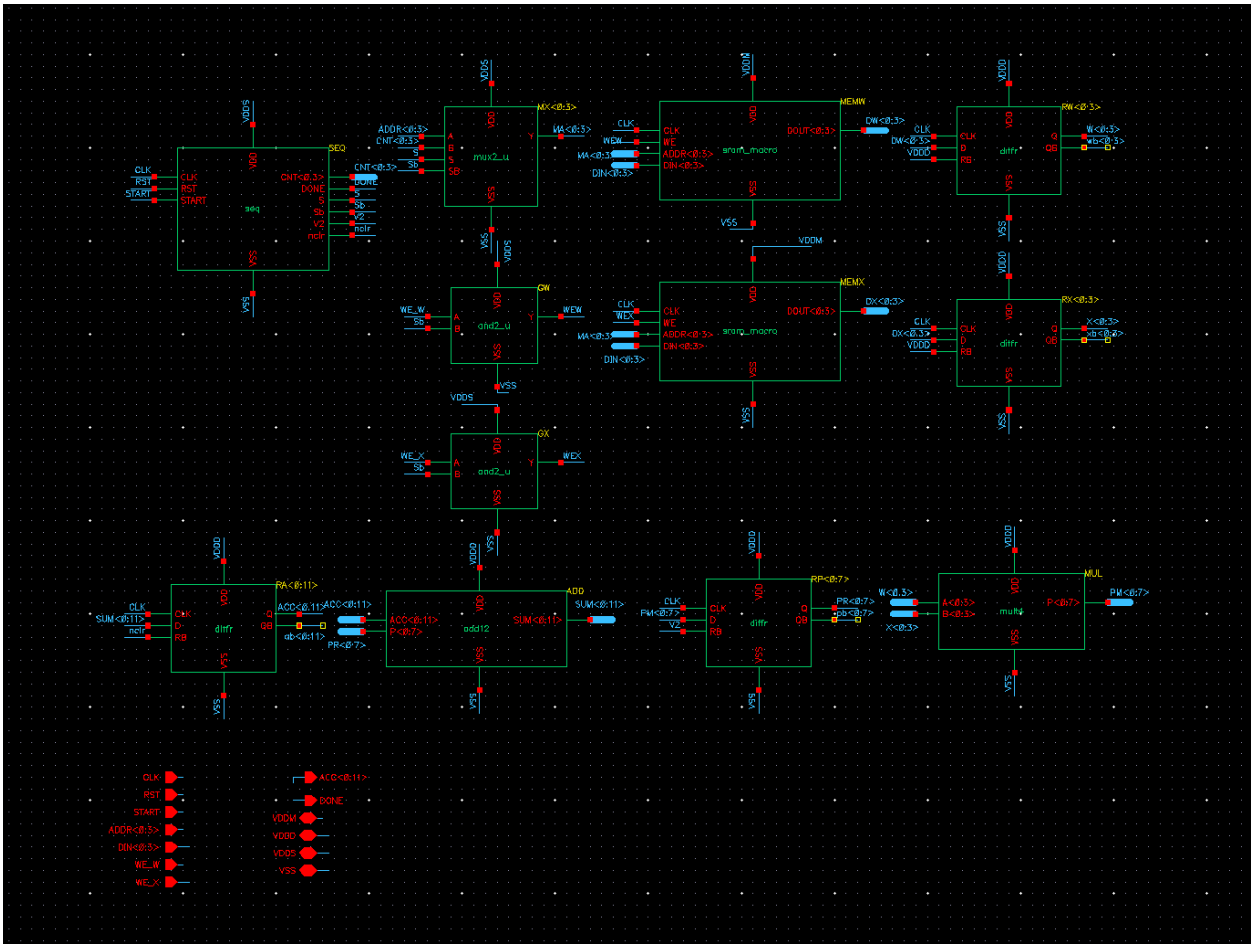


Figure 2: Top-level schematic `mac`. Top left: the sequencer `SEQ`; top middle: the address muxes `MX<0:3>` and the write-enable gates `GW`, `GX`; top right: the macros `MEMW` and `MEMX` (on VDDM) and the operand registers `RW<0:3>`, `RX<0:3>`. Bottom, right to left: the multiplier `MUL`, the product register `RP<0:7>` (RB = `V2`), the adder `ADD` and the accumulator `RA<0:11>` (RB = `nclr`), all on VDDD.

2 Register (3.1)

2.1 Design

The register `diffr` is a static transmission-gate leader-follower flip-flop (Lecture 7, slide 36) with a synchronous active-low reset (Figure 3). The usual input inverter is replaced by a NAND2 of `D` and `RB` (`nand2_u`), so with `RB` = 1 the flop stores `D`, and with `RB` = 0 it loads a 0 at the next rising edge. The clock inverter makes `CKB` locally.

- **Leader** (transparent while CLK = 0): TG1 passes `DN` = $\overline{D \cdot RB}$ to `X1`; `I1` drives `QM` = $\overline{\texttt{X1}}$. When CLK rises, TG1 opens and TG2 closes the loop `QM` → `I2` → `X1`.
- **Follower** (transparent while CLK = 1): `I3` drives `QMB`, TG3 passes it to `X2`, and `I4` drives Q. `I5` makes $\overline{Q}$, which TG4 feeds back to `X2` while CLK = 0.

All inverters are `inv_u` (NMOS 2 / PMOS 3 fins), all transmission gates are 2/2 fins, and `nand2_u`

has 4-fin NMOS in series and 3-fin PMOS in parallel (its logical effort is 7/5). The flop has 24 transistors, 60 fins and $\sum WL = 0.0372\,\mu\text{m}^2$, so its estimated area is $79 \times 0.0372 = 2.9\,\mu\text{m}^2$ per bit.

Instance	Cell	Connection	Fins (N/P)
I0	nand2_u	D, RB → DN	4+4 series / 3, 3
I6	inv_u	CLK → CKB	2/3
TG1	TG	DN ↔ X1, on while CLK = 0	2/2
I1	inv_u	X1 → QM	2/3
I2, TG2	inv_u, TG	QM → X1, on while CLK = 1	2/3, 2/2
I3	inv_u	QM → QMB	2/3
TG3	TG	QMB ↔ X2, on while CLK = 1	2/2
I4	inv_u	X2 → Q	2/3
I5, TG4	inv_u, TG	$Q \to \overline{Q} \to$ X2, on while CLK = 0	2/3, 2/2

Table 1: Devices of diffr. TG1–TG4 are the four transmission gates drawn flat in the schematic.

Why this style. We used a static flip-flop because the accumulator must hold its result after DONE, and several other registers can remain idle for many cycles. A dynamic C^2MOS or TSPC register would eventually lose its stored value. The transmission-gate design is static and non-ratioed, and it uses the same unit devices as the rest of the design (Lecture 7, slide 42). Replacing the input inverter with a NAND2 adds the required synchronous reset without adding another logic stage.

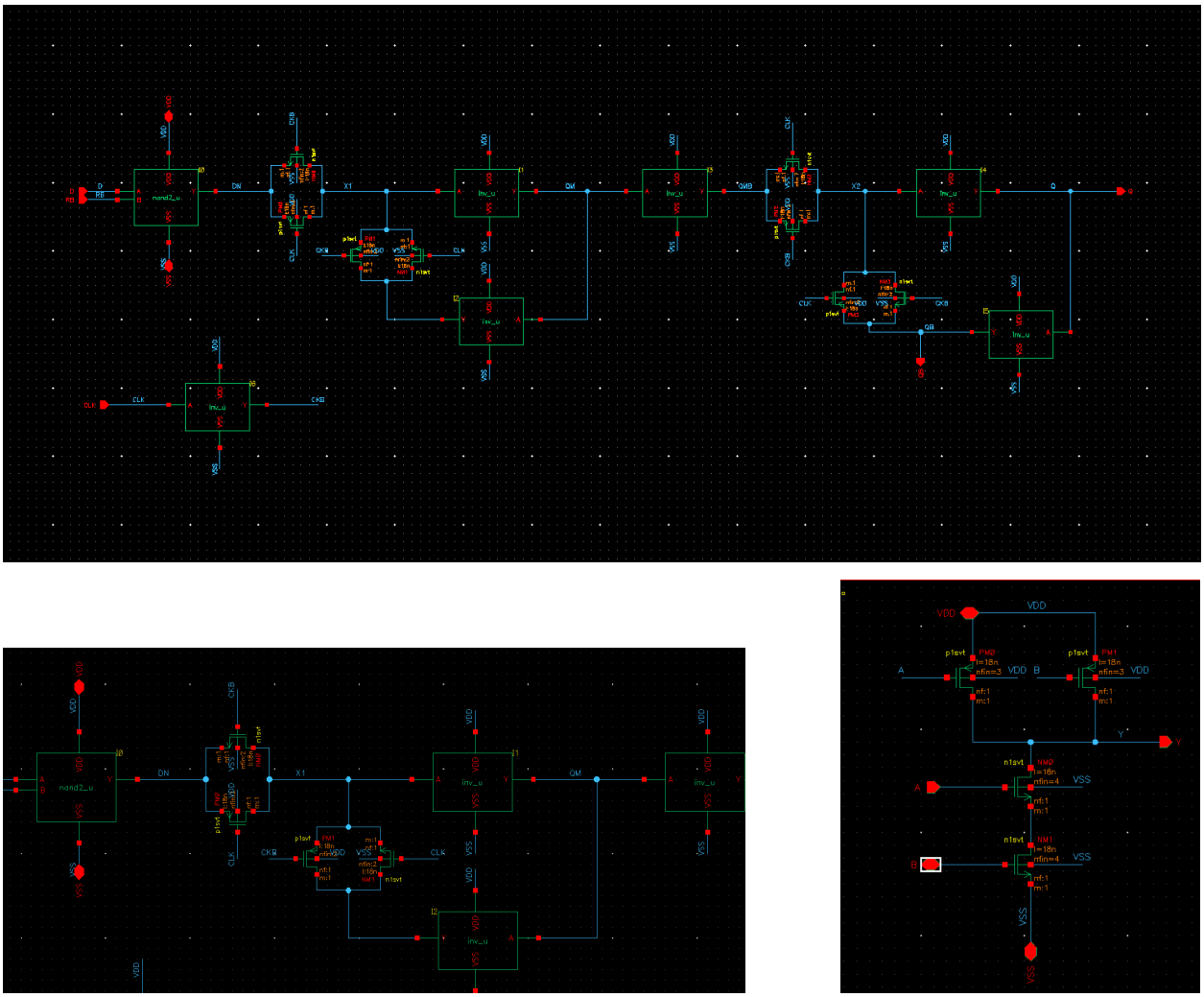


Figure 3: Top: `diffr`. Left to right: the reset NAND `I0`, TG1 with the feedback pair `I2`/TG2, the leader inverter `I1` (`QM`), `I3` (`QMB`), TG3, the output inverter `I4` (Q) and the feedback pair `I5`/TG4 ($\overline{Q}$); bottom left, the clock inverter `I6`. The lower images show the enlarged leader on the left and `nand2_u` on the right. The NAND2 uses 4-fin series NMOS and 3-fin parallel PMOS devices.

2.2 Setup, hold and clock-to-Q

The bench `tb_diffr_sh` (Figure 5) drives `D` through an `inv_u`, so `D` has a realistic edge, and loads Q with four `inv_u` (FO4). The flop runs from its own supply VDDF. The clock has 20 ps edges and a rising edge at 1.51 ns; one transition of `D` is swept from 1.30 to 1.60 ns in 2 ps steps, once rising and once falling. For each point we record t_d (`D` crossing minus CLK crossing, both at $V_{DD}/2$) and the time at which Q crosses $V_{DD}/2$ (`scripts/dffr_sh.py`).

For early `D`, Q follows at a constant clock-to-Q delay. As `D` moves toward the edge, the delay grows (Figure 4). We define the setup time as the $-t_d$ at which clock-to-Q is 10% above its plateau. When `D` arrives later still, the leader has already closed, the flop keeps the old value, and Q moves only at the following edge. The first t_d at which this happens is the hold time: a change of `D` at or after that instant no longer disturbs the value being captured.

Quantity	D rising (Q rises)	D falling (Q falls)	Value used
Clock-to-Q, FO4 load (ps)	70.9	93.5	$t_{ccq} = 71$, $t_{pcq} = 94$
Setup, clock-to-Q +10% (ps)	87.4	75.7	$t_{setup} = 87$
Hold, first missed capture (ps)	−49.4	−63.7	$t_{hold} = -49$

Table 2: Timing of `diffr`. A negative hold time means that `D` may change before the clock edge without corrupting the value captured.

The falling transition at Q is slower because `X2` must rise through TG3. Its NMOS passes a weak high level, and its PMOS turns on only after the local clock inverter generates `CKB`. We did not observe a glitch when a capture was missed. In the sweep, the last captured point and first missed point for a rising `D` were −51 and −49 ps, one 2 ps step apart. We use the more conservative 10% clock-to-Q criterion for setup, giving $t_{setup} = 87$ ps. The resulting register overhead is $t_{pcq} + t_{setup} = 94 + 87 = 181$ ps.

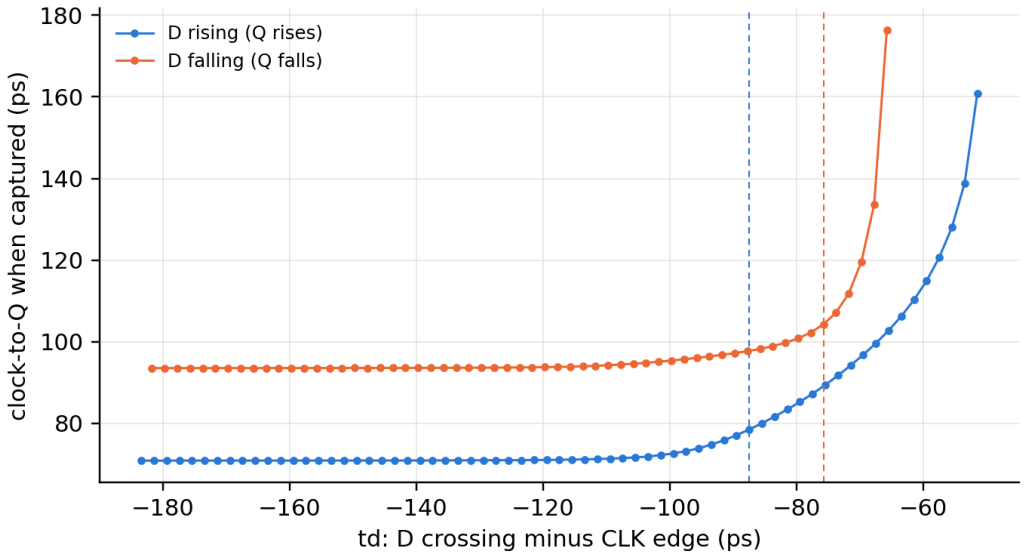


Figure 4: Clock-to-Q of `diffr` against the arrival of `D`, for every swept point at which the new value was captured. The dashed lines mark the setup times (clock-to-Q +10%): 87.4 ps for a rising `D` and 75.7 ps for a falling `D`. The curves end at the last captured point; one step later the flop keeps the old value.

2.3 Clock-input capacitance and energy per clock

The bench `tb_diffr_pwr` is the same circuit with `D` driven either by a square wave that makes Q toggle at every edge or held at 1, and `RB` pulled low from 8 to 9 ns. The clock capacitance is the charge the clock source delivers on one rising edge divided by V_{DD}, and the energy is $-V_{DD} \int i_{VDDF}\, dt$ over four cycles, divided by four.

Quantity	D held at 1	D and Q toggling
Clock-input capacitance C_{clk} (fF)	1.55	1.86
Energy from the flop's supply per clock (fJ)	0.855	4.97
Clock-pin energy $C_{clk}V_{DD}^2$ (fJ)	0.99	1.19
Total energy per clock (fJ)	**1.8**	**6.2**

Table 3: Clock load and energy of `diffr`. The toggling case includes charging the FO4 load on Q.

The clock pin drives the `I6` input (5 fins) and four transmission-gate gates (8 fins): 13 fins, or about 1.3 fF at 0.1 fF per fin, close to the measured 1.55 fF. With `D` toggling, C_{clk} rises to 1.86 fF because the internal nodes and transmission gates also switch. With `D` held, a clocked flop still costs 1.8 fJ per cycle, which matters for the 37 flops of the MAC (see Metrics). The reset works: with `D` = 1 and `RB` = 0 across the 8.51 ns edge, Q is 7–35 µV at 9.2 ns (Figure 6).

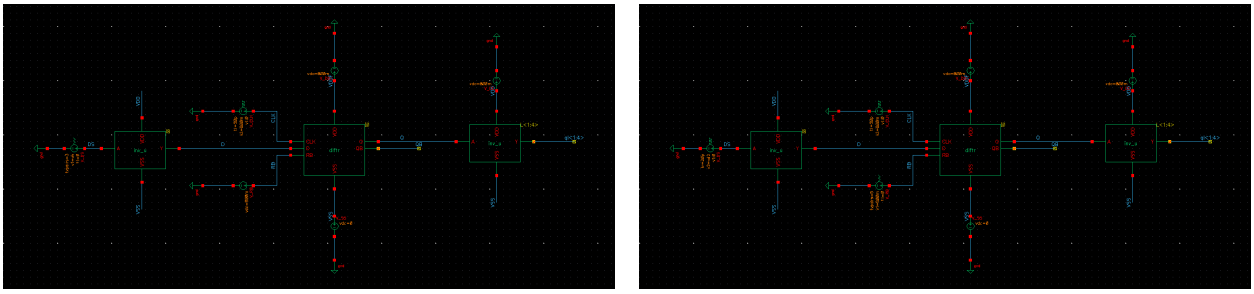


Figure 5: Left: `tb_diffr_sh`, with `D` from a PWL source through an `inv_u`, Q loaded by four `inv_u` (`L<1:4>`) and the flop on its own supply `V_DDF`. Right: `tb_diffr_pwr`, the same bench with a pulse source on `D` and a PWL source on `RB`.

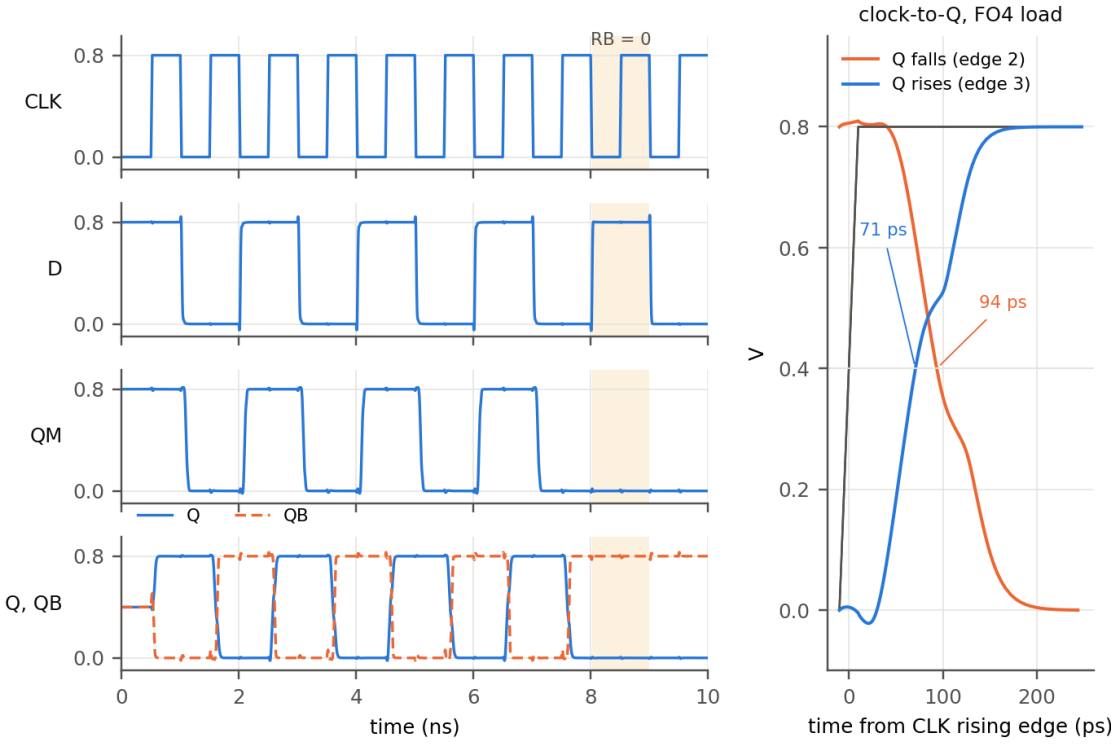


Figure 6: `tb_diffr_pwr` with `D` toggling. Left: CLK, `D`, `QM`, Q and $\overline{Q}$; in the shaded interval `RB` = 0, and the flop loads 0 although `D` = 1. Right: Q against time from the rising CLK edge, with the clock-to-Q delays at $V_{DD}/2$.

3 Datapath (3.2)

Both the multiplier and the accumulator are built from one full-adder cell, `fa_m`, so the logical-effort sizing of its carry path sets the speed of both.

3.1 Mirror full adder `fa_m`

`fa_m` is the 24-transistor mirror adder of Lecture 6 (slides 54–56) plus two output inverters: 28 transistors and 93 fins (Figure 7, Table 4). Its carry stage computes $\overline{C_{out}} = \overline{AB + C_{in}(A+B)}$ on node `COB`; the sum stage computes $\bar{S} = \overline{ABC_{in} + \overline{C_{out}}(A+B+C_{in})}$ on `SB`; two output inverters restore the polarity. The carry in is the input nearest the output in the carry stage, so a late carry sees the shortest path. A half adder is `fa_m` with C_{in} tied to VSS. `and2_u` is `nand2_u` followed by `inv_u`.

Stage	Devices	NMOS fins	PMOS fins
Carry (out COB)	NM0–NM4, PM0–PM4; 2-stacks	4	6
Sum (out SB)	NM5–NM11, PM5–PM11; up to 3-stacks	2	2
CO driver (COB → CO)	NM12, PM12	4	6
S inverter (SB → S)	inv_u	2	3

Table 4: Sizes in fa_m. Input capacitance in fins: A and B 28 each (20 in the carry stage, 8 in the sum stage), C_{in} 18 (10 + 8).

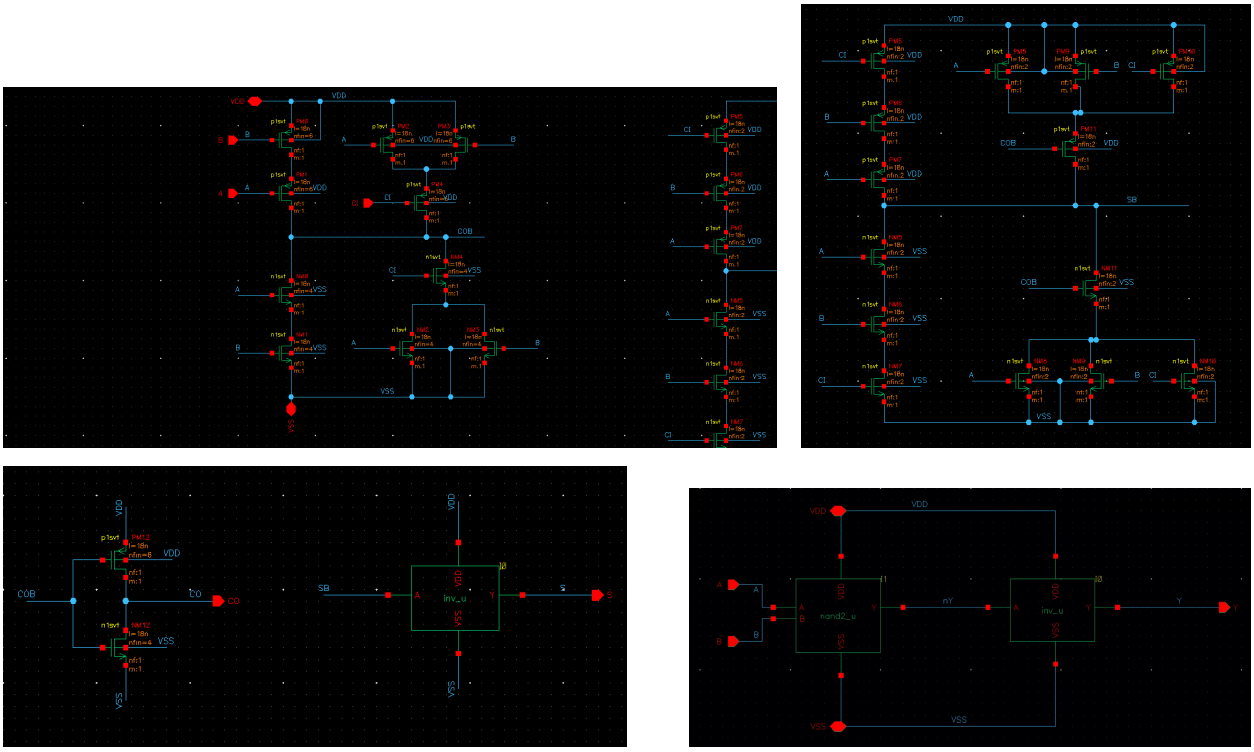


Figure 7: fa_m and and2_u. Top left: the carry stage, with the series pair PM0/PM1 and NM0/NM1 (AB term) and the parallel pairs PM2/PM3, NM2/NM3 in series with the carry-in devices PM4, NM4 next to COB; NMOS 4 fins, PMOS 6 fins. Top right: the sum stage, all 2 fins. Bottom left: the CO driver (NM12 4 fins, PM12 6 fins) and the S inverter. Bottom right: and2_u, a nand2_u (I1) driving an inv_u (I0).

3.2 Logical-effort sizing of the carry path

The critical path of both the multiplier and the accumulator is a carry chain, so we sized it with logical effort before simulating, using the calibration of Part 2: $\tau = 5.9\,\text{ps}$, $p_{inv} = 2.0$ and a unit inverter of 5 fins (NMOS 2, PMOS 3, $C_{in} \approx 0.1\,\text{fF}$ per fin). One bit of the carry chain is two stages: the carry gate (input C_{in}, output COB) and the CO inverter.

- **Carry gate.** Its devices are in 2-stacks, so for the drive of a unit inverter it needs NMOS 4 and PMOS 6 fins. The C_{in} input sees $4 + 6 = 10$ fins, so $g = 10/5 = 2$, and $p = 3\,p_{inv} = 6$ (the estimate $p = n\,p_{inv}$ for an n-input gate, here $n = 3$: A, B, C_{in}).

- **Load on `COB`:** the CO inverter (x fins) and the sum stage's `COB` pair (`NM11` + `PM11`, 4 fins).
- **Load on CO:** the next bit's carry gate (10 fins) and its sum-stage C_{in} devices (8 fins), 18 fins in all.

The delay per bit, in units of τ, as a function of the CO inverter size x is

$$d(x) = \underbrace{2 \cdot \frac{x+4}{10} + 6}_{\text{carry gate}} + \underbrace{\frac{18}{x} + 2}_{\text{CO inverter}}, \qquad \frac{\partial d}{\partial x} = \frac{2}{10} - \frac{18}{x^2} = 0 \;\Rightarrow\; x = \sqrt{90} = 9.5 \text{ fins.} \tag{1}$$

We chose $x = 10$ fins (NMOS 4, PMOS 6), which gives

$$d = 2.8 + 6 + 1.8 + 2 = 12.6\,\tau = 74\,\text{ps per bit.} \tag{2}$$

The parasitic contribution is 8 of the total 12.6τ. The estimate is not very sensitive to the CO-inverter size near the optimum: choosing $x = 6$ or $x = 14$ changes the delay by only 0.3–0.4τ. We kept the sum stage at minimum size because it is not on the carry path and otherwise adds load to `COB` and C_{in}.

Unit test. `tb_fa` drives one `fa_m` through all eight input states and chains three more with $A = 1$, $B = 0$, so that each passes its carry in to its carry out (a 3-bit ripple, loaded by a fifth cell like any carry in the array). The truth table is correct in all 8 states at full rail. The measured carry delay is **66.5 ps (rise) and 67.6 ps (fall) per bit**, about 10% below the 74 ps estimate, and carry in to sum is 129 ps (rise) and 105 ps (fall) with one `inv_u` as load.

3.3 Multiplier `mult4`

`mult4` is a 4×4 ripple-row array multiplier (Figure 8). Sixteen `and2_u` (`G0`–`G15`) form the partial products: gate $k = 4j + i$ computes `pp<k>` $= A_i B_j$, with A_i running down the columns and B_j along the rows; `G0` gives P_0 directly. Twelve `fa_m` (`F1`–`F12`) add the rows: `F1`–`F4` add rows 0 and 1 (half adder, two full adders, half adder), and `F5`–`F8` and `F9`–`F12` each add one more row (half adder, three full adders). The carry out of `F12` is P_7. We checked the netlist of connections in Python against all 256 products before drawing the schematic.

Critical path. With t_{AND} for a partial product, t_c for a carry and t_s for a sum, the arrival time of every output follows from the connections. The latest is P_6:

$$\text{AND} \to \texttt{F1}\,(c) \to \texttt{F2}\,(s) \to \texttt{F5}\,(c) \to \texttt{F6}\,(c) \to \texttt{F7}\,(c) \to \texttt{F8}\,(s) \to \texttt{F11}\,(c) \to \texttt{F12}\,(s) \to P_6,$$

$$t_{P6} = t_{AND} + 5\,t_c + 3\,t_s. \tag{3}$$

(Other paths tie with it; for example, `F3`'s sum through `F6`, `F9` and `F10` reaches `F11` at the same time.) Along this path the carries enter the next cell at C_{in}, as in the sizing above, while two of the sums (`F2` $\to$ `F5`, `F8` $\to$ `F11`) enter at A. The AND gate is a `nand2_u` ($g = 7/5$, input 7 fins) driving an `inv_u` (5 fins), which drives one A or B input of an adder (28 fins):

$$t_{AND} = \left(\tfrac{7}{5} \cdot \tfrac{5}{7} + 2p_{inv}\right) + \left(\tfrac{28}{5} + p_{inv}\right) = 5 + 7.6 = 12.6\,\tau = 74\,\text{ps.} \tag{4}$$

With $t_c = 67\,\text{ps}$ and $t_s = 117\,\text{ps}$ (the mean of the measured 129 and 105 ps),

$$t_{mult} \approx 74 + 5 \times 67 + 3 \times 117 = 760\,\text{ps.} \tag{5}$$

Unit test. `tb_mult4` applies all 256 input pairs, one per nanosecond, through two `inv_u` per input, with an `inv_u` on each product bit. All 256 products are correct (`scripts/mult_check.py`). The longest settling time is **957 ps** after the vector edge, for 11×11 after 11×10 (Figure 9). The inputs reach `mult4` about 67 ps after the vector edge (two `inv_u`), so the multiplier's own critical path is about **890 ps**, 17% above the estimate. The estimate takes t_s from `tb_fa`, where S drives one `inv_u` (5 fins). In the array, two of the three sums on the path drive an A input (28 fins), which adds about $(28 - 5)/5 = 4.6\,\tau = 27\,\text{ps}$ to each S inverter; with that correction the estimate is $760 + 2 \times 27 \approx 810\,\text{ps}$. The remaining difference is likely caused by slower input transitions in the multiplier. Every A and B input is a 28-fin load, while the logical-effort calculation assumes an idealized input transition. The accumulator does not have the same discrepancy because its carry outputs feed the lighter C_{in} inputs directly.

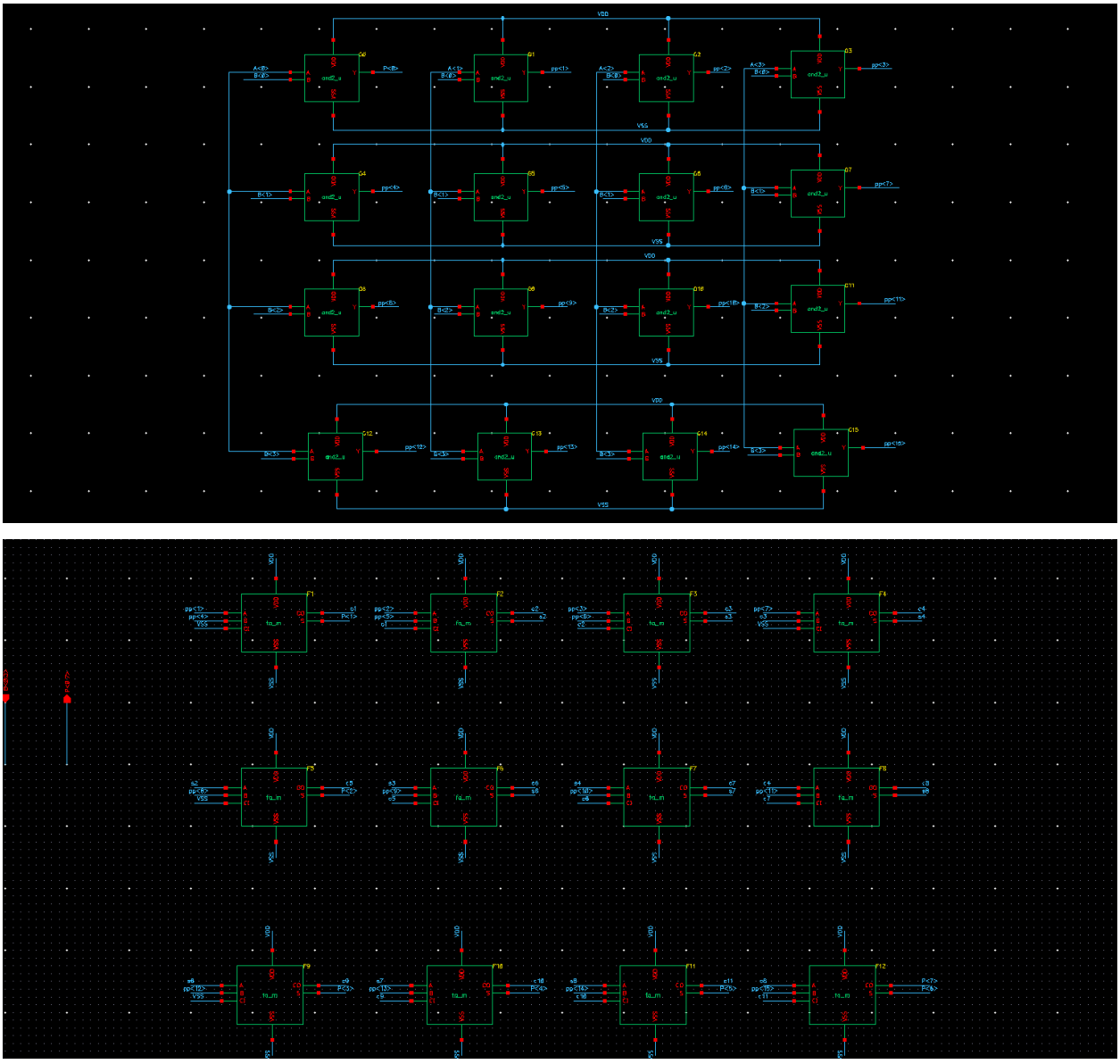


Figure 8: `mult4`. Top: the sixteen `and2_u` `G0`–`G15` as a crossbar; $A\langle i\rangle$ runs down column i and $B\langle j\rangle$ along row j, so gate $4j+i$ gives `pp<`$4j+i$`>` and `G0` gives $P\langle 0\rangle$. Bottom: the three adder rows. `F1`–`F4` add rows 0 and 1 and produce $P\langle 1\rangle$; `F5`–`F8` add row 2 and produce $P\langle 2\rangle$; `F9`–`F12` add row 3 and produce $P\langle 3{:}7\rangle$. `F1`, `F4`, `F5` and `F9` are half adders (C_{in} = VSS).

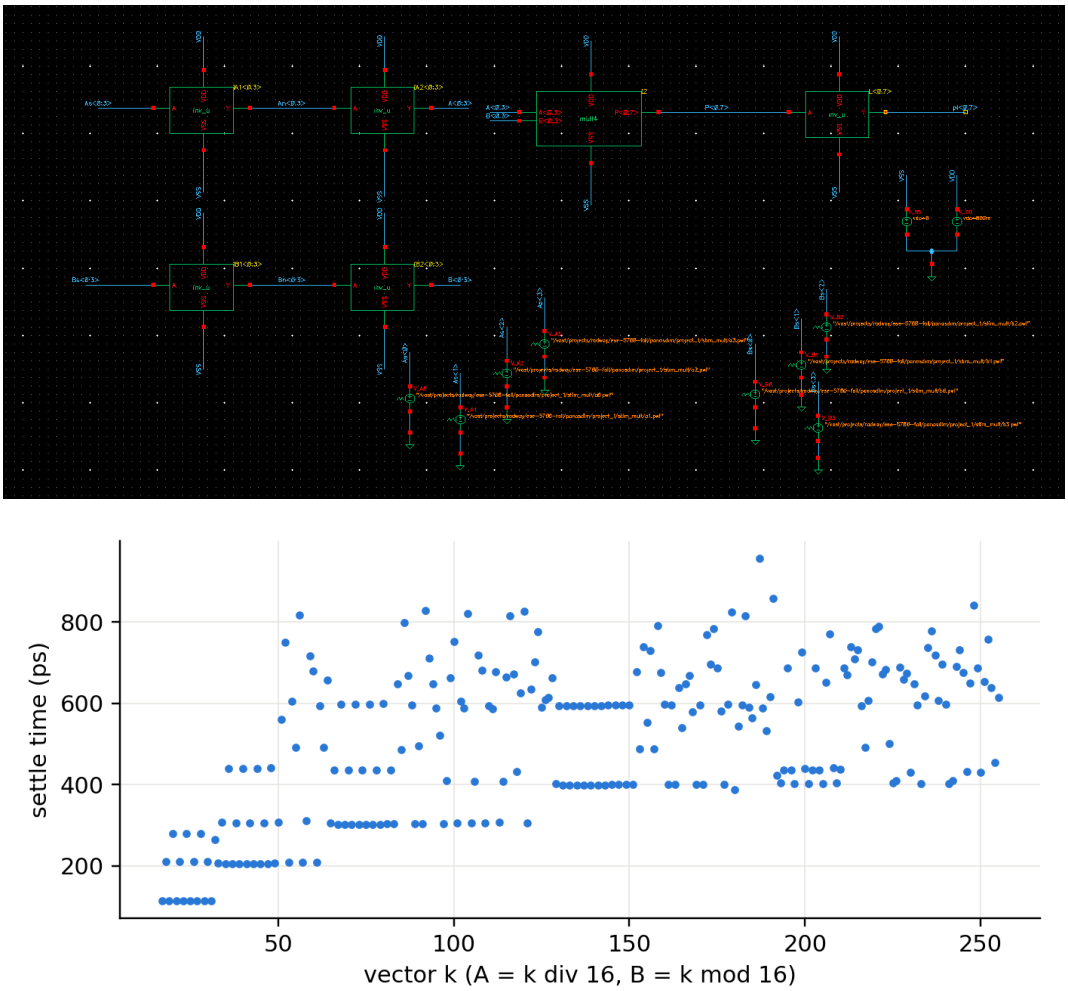


Figure 9: Top: `tb_mult4`, with eight PWL sources, two `inv_u` per input (`IA1`, `IA2`, `IB1`, `IB2`) and an `inv_u` load on each product bit (`L<0:7>`). Bottom: time after each vector edge at which the last product bit settles, for vector k ($A = k \operatorname{div} 16$, $B = k \bmod 16$). All 256 products are correct; the slowest, 957 ps, is 11×11 after 11×10.

3.4 Accumulator adder `add12`

`add12` is a 12-bit ripple-carry adder: one iterated instance `AD<0:11>` of `fa_m` with $A =$ `ACC<0:11>`, $B =$ `P<0:7>` padded with four VSS, $C_{in} =$ `VSS`, `c<1:11>` and $C_{out} =$ `c<1:11>`, `cout` (Figure 10). Bits 8–11 add 0 and act as an incrementer. `cout` is unused: the largest result is $16 \times 15 \times 15 = 3600 < 4096$. The worst case is a carry that ripples from bit 0 to bit 11: 11 carry stages and one carry-in-to-sum,

$$t_{add} \approx 11\, t_c + t_s = 11 \times 67 + 129 \approx 870\,\text{ps}. \tag{6}$$

A ripple adder is enough here: it is the smallest adder, and with a separate accumulate stage it is no slower than the multiplier.

Unit test. `tb_add12` holds `ACC` at 2047 (bits 0–10 set) and drives $P = 0, 1, 0, 1, 2, \ldots, 255$, one value per nanosecond. The step $0 \to 1$ ripples a carry through all twelve bits ($2047 \to 2048$), and the step $1 \to 0$ ripples it back ($2048 \to 2047$). All 258 sums are correct (`scripts/add_check.py`).

In the full ripple the sum bits settle one after the other, every 66–67 ps, from 213 ps to 885 ps after the vector edge (Figure 11). Measured from the `P` edge at the adder input, SUM⟨11⟩ settles 818.8 ps after it for $2047 + 1$ and **884.6 ps** after it for the return $2048 \to 2047$, the worst case. The estimate of 870 ps is within 2%, because here the carries go from CO to C_{in} exactly as in `tb_fa`.

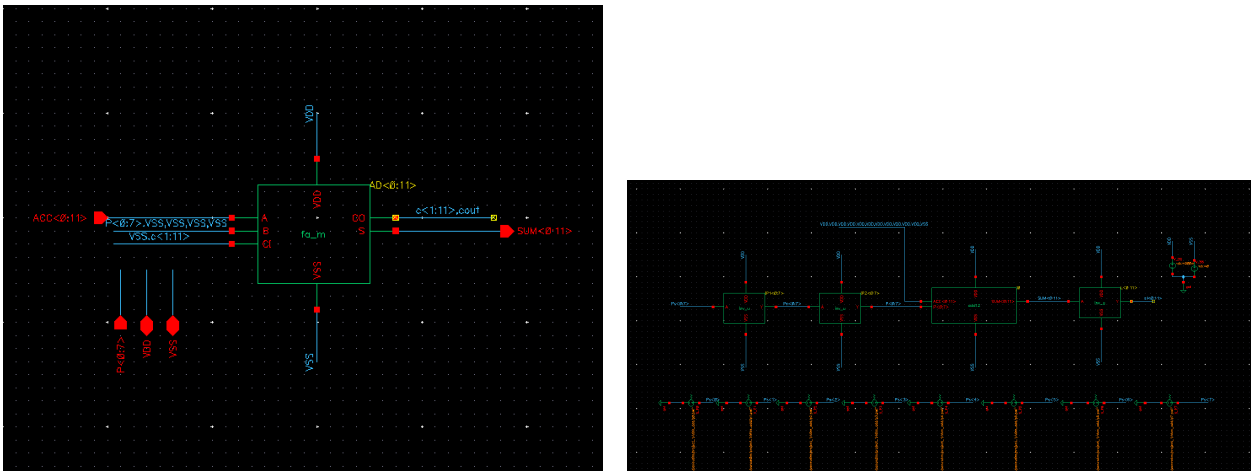


Figure 10: Left: `add12`, one iterated `fa_m` (`AD<0:11>`); B is `P<0:7>` followed by four VSS, C_{in} is VSS followed by `c<1:11>`. Right: `tb_add12`, with `ACC` tied to 2047 (`VDD` ×11, `VSS` on bit 11), eight PWL sources on `P` through two `inv_u`, and an `inv_u` load on each sum bit.

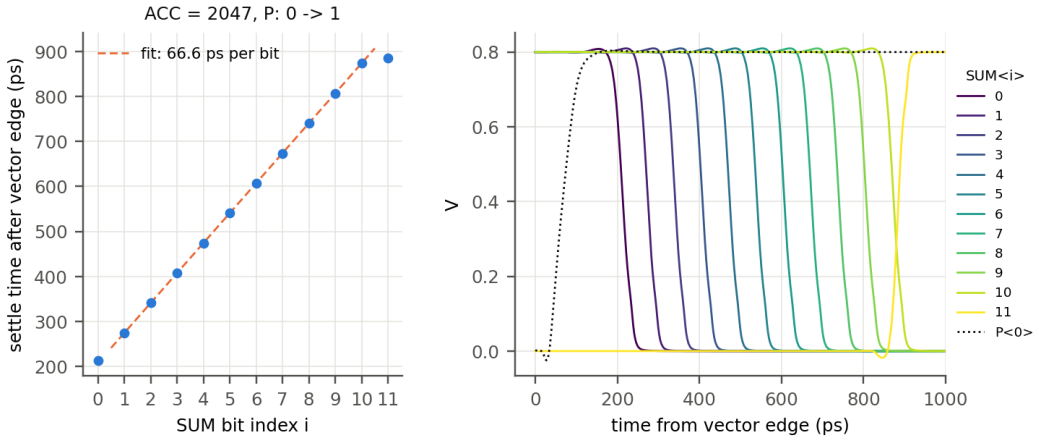


Figure 11: The full ripple $2047 + 1$ in `tb_add12`. Left: time after the vector edge at which each sum bit settles; each bit adds one carry delay (fit 66.6 ps per bit). Right: the sum bits and $P\langle 0 \rangle$. Times are from the vector edge, so they include about 66 ps of input buffers ($885 - 66 \approx 819$ ps from the adder input).

4 Sequencer (3.3)

4.1 State machine

The controller has four states (Figure 12), encoded by the state flop S, the counter `CNT` and `DONE`:

- **IDLE** ($S = 0$, `DONE` $= 0$): the load port owns the memories; `CNT` is held at 0.

- **COMPUTE** ($S = 1$, 16 cycles, CNT = 0 to 15): entered on the edge that samples START; that edge also clears ACC and DONE. Leaves when CNT = 15.
- **DRAIN** ($S = 0$, 3 cycles): the last three products move through the pipeline. This state is not a separate flop: it is the valid chain V_1–V_3 emptying.
- **DONE** (DONE = 1): ACC holds the result until the next START (back to COMPUTE) or RST.

The RST arrow into IDLE applies from every state.

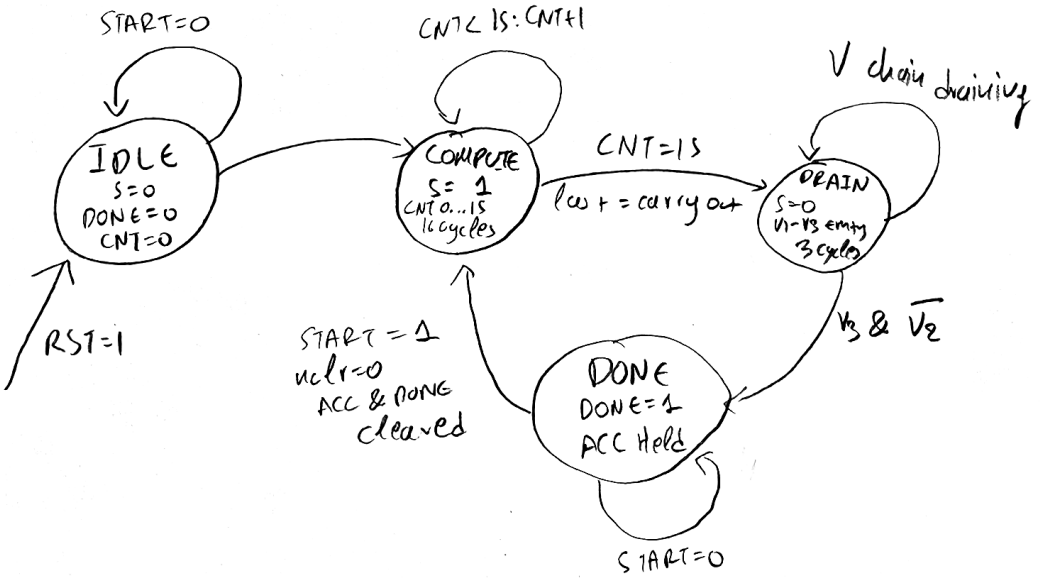


Figure 12: State diagram of the sequencer. Transitions happen at rising CLK edges.

4.2 Logic

With $go = \texttt{START} \cdot \bar{S}$ and $\overline{\texttt{RST}} = \texttt{RSTb}$, the next-state logic is

$$S^+ = go + S \cdot \overline{last}, \qquad (\texttt{FS}, \text{RB} = \texttt{RSTb}) \tag{7}$$

$$\texttt{CNT}^+ = \texttt{CNT} + 1, \qquad (\texttt{FC<0:3>}, \text{RB} = \texttt{cntrb} = S \cdot \texttt{RSTb}) \tag{8}$$

$$V_1^+ = S, \quad V_2^+ = V_1, \quad V_3^+ = V_2, \qquad (\texttt{FV<1:3>}, \text{RB} = \texttt{RSTb}) \tag{9}$$

$$\texttt{DONE}^+ = V_3 \cdot \overline{V_2} + \texttt{DONE}, \qquad (\texttt{FD}, \text{RB} = \texttt{nclr}) \tag{10}$$

$$\texttt{nclr} = \texttt{RSTb} \cdot \overline{go}. \tag{11}$$

- **Counter.** Four diffr and a 4-bit incrementer built from four fa_m (INC<0:3>, B = VSS, C_{in} = VDD,ic<1:3>). Its carry out, *last*, is 1 exactly when CNT = 15, so the same cell gives the next count and the end of the compute phase. The counter's reset is $S \cdot$ RSTb, so CNT is 0 whenever the unit is idle and the first read is address 0 without any extra logic.
- **Valid chain.** V_1–V_3 are S delayed by 1, 2 and 3 cycles. V_2 is high for exactly the 16 edges at which the product register holds a valid product, so V_2 drives the product register's reset: RP loads zeros outside those cycles. The accumulator therefore needs no enable: it adds 0 before the first product and after the last, and holds its result after DONE.

- **DONE.** $V_3 \cdot \overline{V_2}$ is high for the one cycle in which the last product is in the adder, so DONE rises on the same edge that stores the final sum; the OR with DONE holds it.
- **Clear.** nclr is low during RST and during the cycle in which START is first seen. It resets ACC and DONE on the same edge that starts the count. If START stays high for longer, S is already 1 and $go = 0$, so a long START pulse does not restart the unit.

The logic is built from two inv_u (I0 for RSTb, IL for lastb), five nand2_u (N1–N3, I11, I12) and two and2_u (A1 for nclr, I7 for cntrb) around nine diffr (Figure 13). S^+ is the NAND–NAND form of the sum of products: N1 gives $\overline{go}$, N2 gives $\overline{S \cdot \overline{last}}$, and N3 combines them; I11 and I12 do the same for DONE. A cycle-level model of the whole MAC (scripts/seqsim.py) gives ACC = 845, 3600 and 800 for Tests A, B and C and holds after DONE; we checked it before drawing the schematic.

Address mux. mux2_u is three nand2_u: $Y = \overline{\overline{A \cdot \mathrm{SB}} \cdot \overline{B \cdot S}} = A\,\overline{S} + B\,S$ (Figure 14). It takes both S and SB from the state flop's Q and $\overline{Q}$, so it needs no inverter of its own.

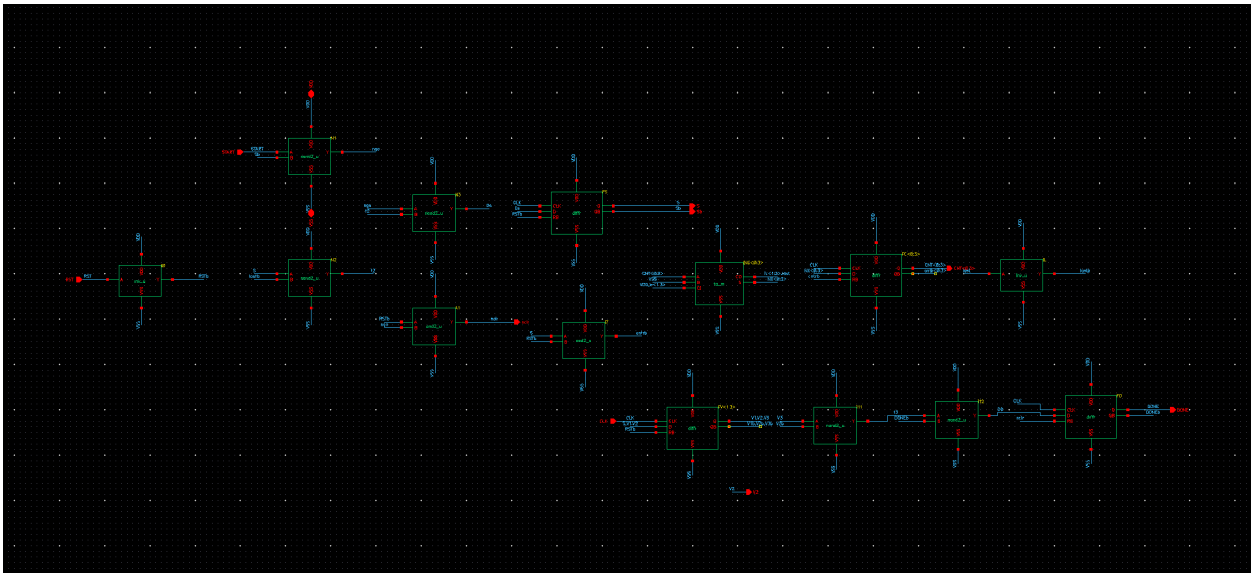


Figure 13: seq. Upper left: I0 (RSTb), the state logic N1–N3 and the state flop FS; middle: A1 (nclr), I7 (cntrb), the incrementer INC<0:3>, the counter flops FC<0:3> and IL (lastb); bottom: the valid chain FV<1:3>, the DONE logic I11, I12 and the DONE flop FD.

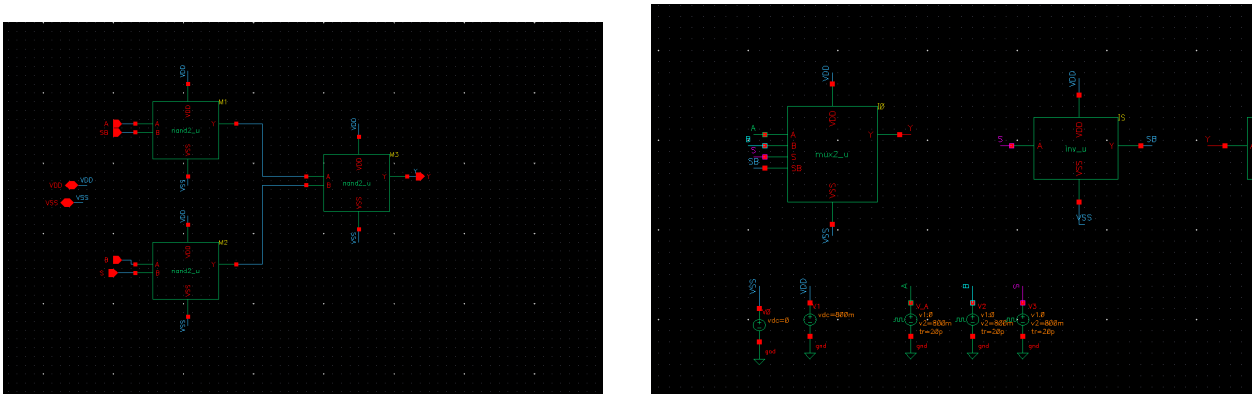


Figure 14: Left: `mux2_u`, `M1` = NAND(A, SB), `M2` = NAND(B, S), `M3` = NAND(`M1`, `M2`). Right: `tb_mux`, with pulse sources on A, B and S, `SB` made by an `inv_u` and an `inv_u` load on Y.

4.3 Unit tests

`tb_seq`. The sequencer is tested at $T = 1.2$ ns with an `inv_u` on each of its nine outputs (Figure 15). `RST` is high for the first edges; `START` is sampled at the 6.6 ns edge (E_0). Results:

- S rises at 6.684 ns: 84 ps after the nominal 6.6 ns source edge, or 74 ps after CLK crosses $V_{DD}/2$ at 6.61 ns, consistent with $t_{ccq} = 71$ ps.
- `CNT` is 0 at E_0, advances by one at every edge from the 7.8 ns edge (E_1), reaches 15 at E_{15} and returns to 0 at E_{16}, when S falls.
- V_2 is high from 9.0 ns (E_2) to 28.2 ns (E_{18}): 16 cycles.
- `DONE` rises at 29.46 ns, just after $E_{19} = 6.6 + 19 \times 1.2 = 29.4$ ns (50 ps after the CLK crossing at 29.41 ns), and is still 0.80 V at 39 ns. Measured from the CLK crossing at E_0 (6.61 ns) this is 19.04 cycles.
- `nclr` is low during `RST` and during the `START` cycle only.

The decoded `CNT` shows short glitches when several bits change on one edge. They occur while CLK is high, when the macros' transparent-low address latches are closed, so the memories never see them.

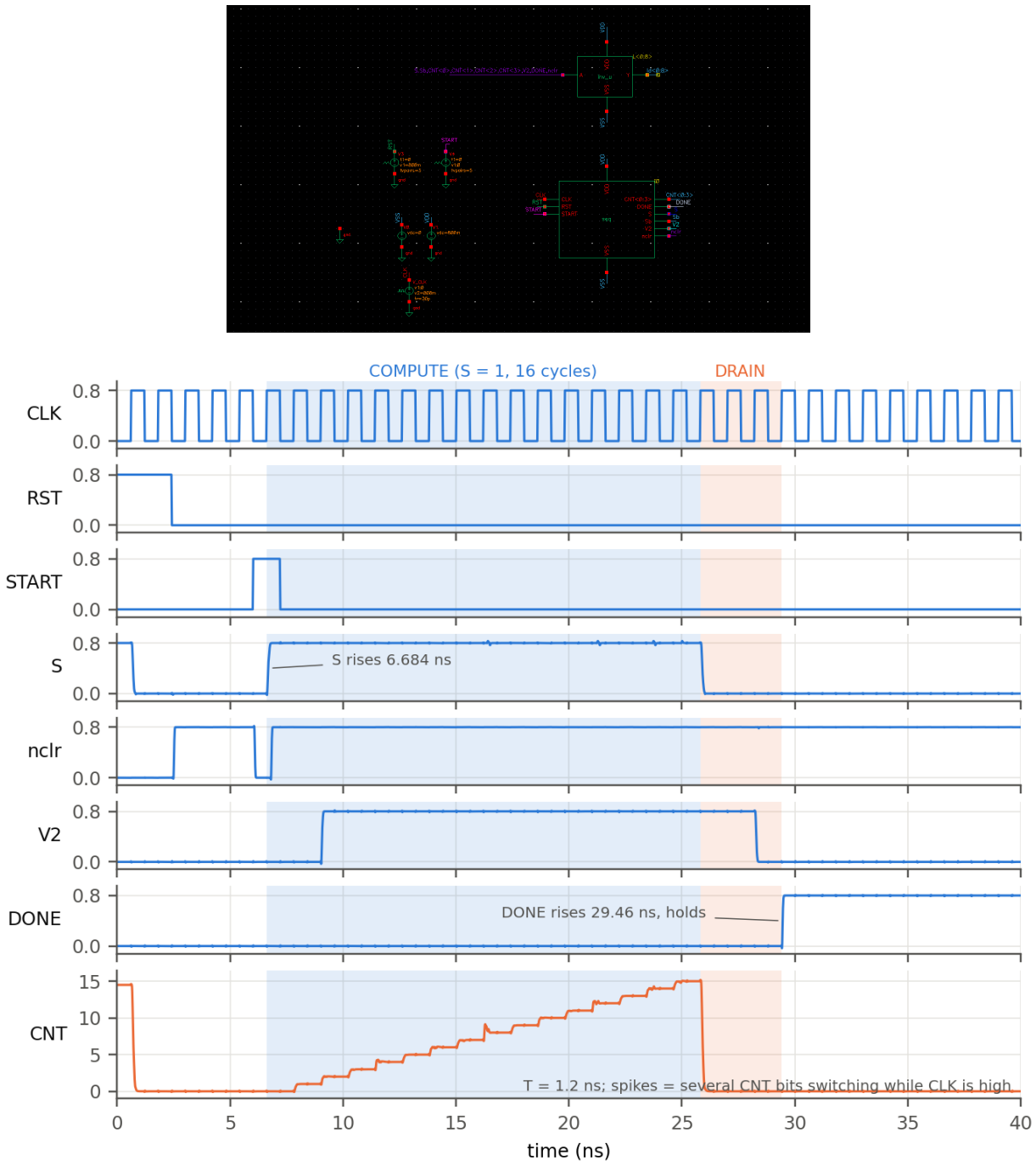


Figure 15: Top: `tb_seq`, the sequencer with PWL sources on `RST` and `START`, a 1.2 ns clock and an `inv_u` load on each output (`L<0:8>`). Bottom: one dot product. `START` is sampled at 6.6 ns, the count runs from the next edge, V_2 marks the 16 valid products and `DONE` rises 19 cycles after `START` and stays high.

`tb_mux`. A toggles every 0.4 ns, B every 0.8 ns and S every 1.6 ns, so every combination occurs. Y follows A while $S = 0$ and B while $S = 1$ in every case. The delay is about 42 ps from A and 45 ps from B. When S falls with $A = B = 1$, Y dips to 0.685 V: `SB` comes one inverter after S, so for a

moment both are low and both first-level NANDs output 1. The dip never crosses $V_{DD}/2$, and in the MAC S changes only just after a rising edge, while the address latches are closed.

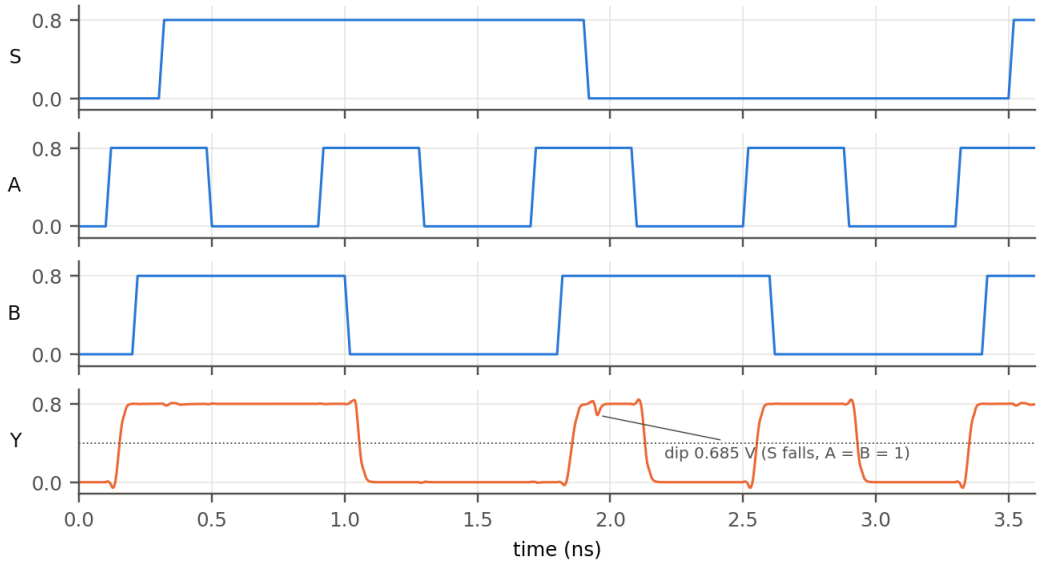


Figure 16: `tb_mux`: Y follows A for $S = 0$ and B for $S = 1$. The small dip on Y when S falls with $A = B = 1$ reaches only 0.685 V.

4.4 One dot product

Figure 17 shows the timing of `START`, the counter, the memory reads, the accumulator and `DONE` over one dot product, in edges from E_0. The address presented to the macros is `MA` = `CNT` from E_0 on: `CNT` is 0 when S rises, so E_1 samples address 0, and `CNT` has advanced to k when E_{k+1} samples address k. Word k is on `DOUT` 442 ps after E_{k+1} and stays valid until the next read's sense enable opens, about 268 ps after E_{k+2}, so `RW/RX` capture it at E_{k+2}. The product is captured at E_{k+3} and added at E_{k+4}. S falls at E_{16}, after address 15 has been sampled; V_2 covers E_3 to E_{18}, the 16 edges at which the product register loads; the last product is added at E_{19}, and `DONE` rises after the same edge.

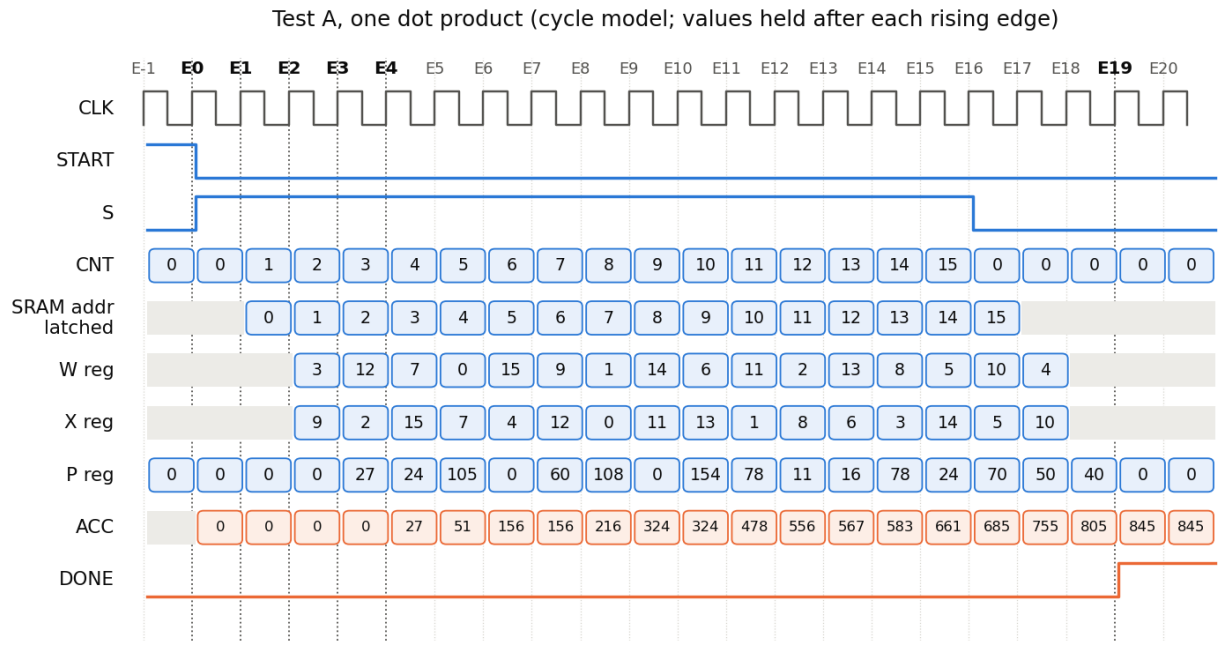


Figure 17: Timing of one dot product (Test A, cycle level; drawn by `scripts/report_figs.py` with the same pipeline rules as the cycle model `scripts/seqsim.py`), in clock edges from E_0, the edge that samples `START`. Each box is the value held after the edge on its left. The macros latch address k at E_{k+1}, the operand registers hold word k after E_{k+2}, the product register holds $W[k]X[k]$ after E_{k+3} and `ACC` includes it after E_{k+4}; `DONE` rises after E_{19} with ACC $= 845$.

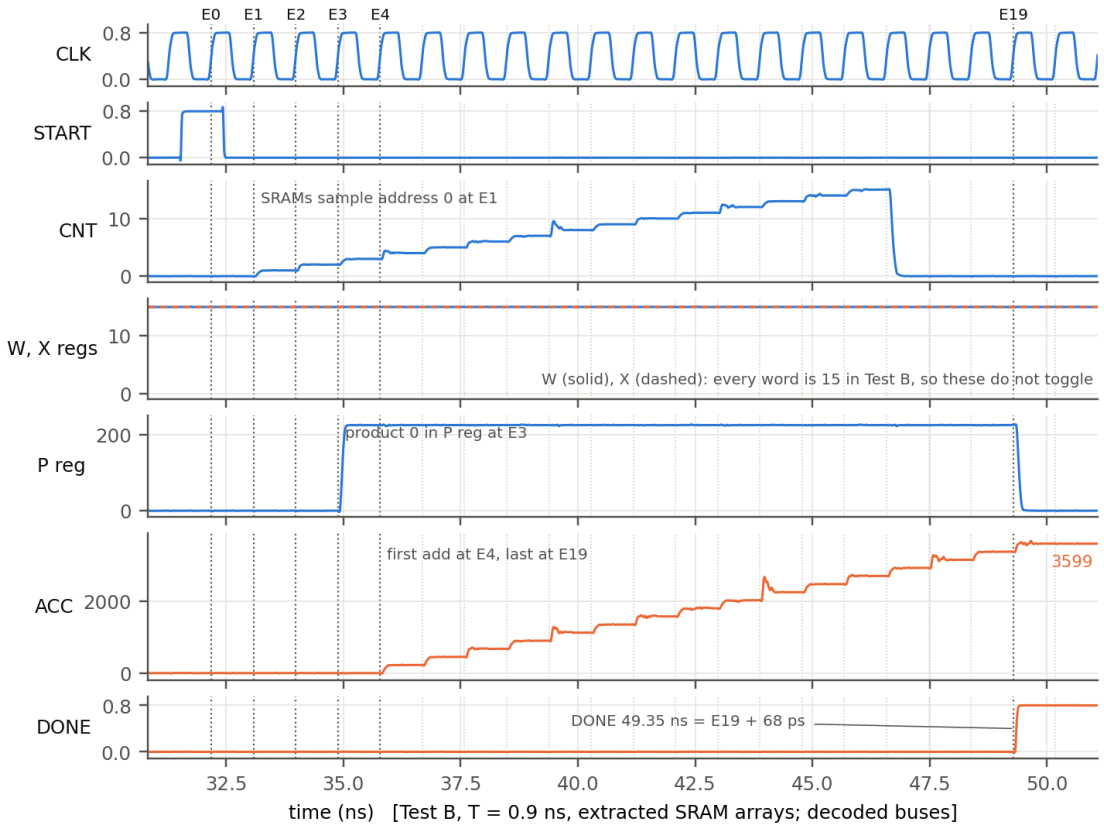


Figure 18: The same sequence measured in `tb_mac` (Test B, $T = 0.9$ ns, extracted arrays; buses decoded from the bit voltages). `CNT` advances from E_1, the product register holds $15 \times 15 = 225$ from E_3 to E_{18} and clears at E_{19}, `ACC` rises in sixteen steps from E_4 to E_{19}, and `DONE` rises at 49.35 ns, 68 ps after E_{19}. The operand registers stay at 15 throughout, since every word is 15. The analog decode reads 3599 at the end; the bit-rounded value is 3600.

5 Timing closure (3.4)

5.1 Hand estimate

Each pipeline stage must satisfy $T \geq t_{pcq} + t_{logic} + t_{setup}$, with $t_{pcq} + t_{setup} = 94 + 87 = 181$ ps for `diffr`. All registers and both macros take the same clock node, so the skew is 0 (Section 5.3).

Stage	t_{logic}, LE / unit test (ps)	T_{min}, LE (ns)	T_{min}, unit test (ns)
Read: CLK → DOUT → `RW/RX`	$t_{access} = 442$	0.53; the macro needs 0.70	
Multiply: `RW/RX` → `MUL` → `RP`	760 / 890	0.94	1.07
Accumulate: `RP`, `RA` → `ADD` → `RA`	870 / 885	1.05	1.07

Table 5: Hand estimate of the minimum period of each stage. The read is timed from the CLK edge at the macro, so it adds only t_{setup}: $442 + 87 = 529$ ps; the macro's own precharge and latch timing needs 0.7 ns (Part 2).

The accumulate stage limits: the hand estimate is $T_{CLK,min} = 94 + 870 + 87 \approx 1.05$ ns from

logical effort, or $94 + 885 + 87 \approx 1.07$ ns from the measured unit-test delays, so about **1.1 ns**. The multiply stage is 0.11 ns shorter by logical effort and equal by the unit-test delays, and the memory is not on the critical path.

5.2 Measured $T_{CLK,min}$

We ran Test A in the full bench `tb_mac` (Section 6) at 1.2, 1.0, 0.9 and 0.8 ns, and Tests B and C at 1.2 and 0.9 ns, each with the stimulus regenerated for the period. Test A passes at 1.2, 1.0 and 0.9 ns and fails at 0.8 ns (ACC = 909 instead of 845); B and C pass at 0.9 ns (Table 7). **We take $T_{CLK,min} = 0.9$ ns**, the shortest period at which all three tests pass. We did not run B and C at 1.0 or 0.8 ns. A longer period only adds slack to every path, so a pass at 0.9 ns implies a pass at 1.0 ns; and because Test A already fails at 0.8 ns, that period cannot be $T_{CLK,min}$ whatever B and C give.

Why the measured period is shorter. The 1.07 ns estimate combines the worst multiplier vector (11×11 after 11×10) or the worst ripple ($2047 + 1$) on the register overhead. The three tests never excite those exact paths: no product in them follows its worst predecessor, and no accumulation ripples a carry through all twelve bits (the longest chain, in Test C, is $750 + 50 = 800$: a carry generated at bit 1 ripples through bit 7 into bit 8, seven carry stages against eleven for $2047 + 1$; in Test B, $2025 + 225 = 2250$ ripples from bit 5 into bit 11, six stages). The measured 0.9 ns is therefore the minimum period for Tests A, B and C. A period guaranteed for every input would need about 1.07 ns.

The failure at 0.8 ns. We read `ACC` after every compute-phase edge of the 0.8 ns run (exported as `tb_mac_A_T0p8_diag.csv`) and compared it with the running sum (Table 6, Figure 19).

Cycle k	$W[k] \times X[k]$	Correct product	Added at 0.8 ns	Error	Cumulative error
0–2	$3 \cdot 9$, $12 \cdot 2$, $7 \cdot 15$	27, 24, 105	same	–	0
3	$0 \cdot 7$	0	16	+16	+16
4–6	$15 \cdot 4$, $9 \cdot 12$, $1 \cdot 0$	60, 108, 0	same	–	+16
7	$14 \cdot 11$	154	186	+32	+48
8	$6 \cdot 13$	78	94	+16	+64
9–15			same	–	+64

Table 6: Test A at $T = 0.8$ ns, cycle by cycle. The final value is $845 + 64 = 909$.

The memory read is not the cause. In cycle 3, one wrong word would give a multiple of 7 (if W were misread) or 0 (if X were misread), never 16; in cycles 7 and 8, the values added, $186 = 2 \cdot 3 \cdot 31$ and $94 = 2 \cdot 47$, are not the product of any two 4-bit numbers, so no misread word could produce them. Each error is instead a single extra power of two (16, 32, 16), which is what one bit that has not reached its final value at the capturing edge looks like: a carry still rippling in the adder or a product bit still settling in the multiplier. The limit is therefore the datapath, the multiply or accumulate stage, as the hand estimate predicts. The 0.8 ns run did not save `PR`, so we cannot tell which of the two stages failed. The memory itself runs at 0.7 ns (Part 2).

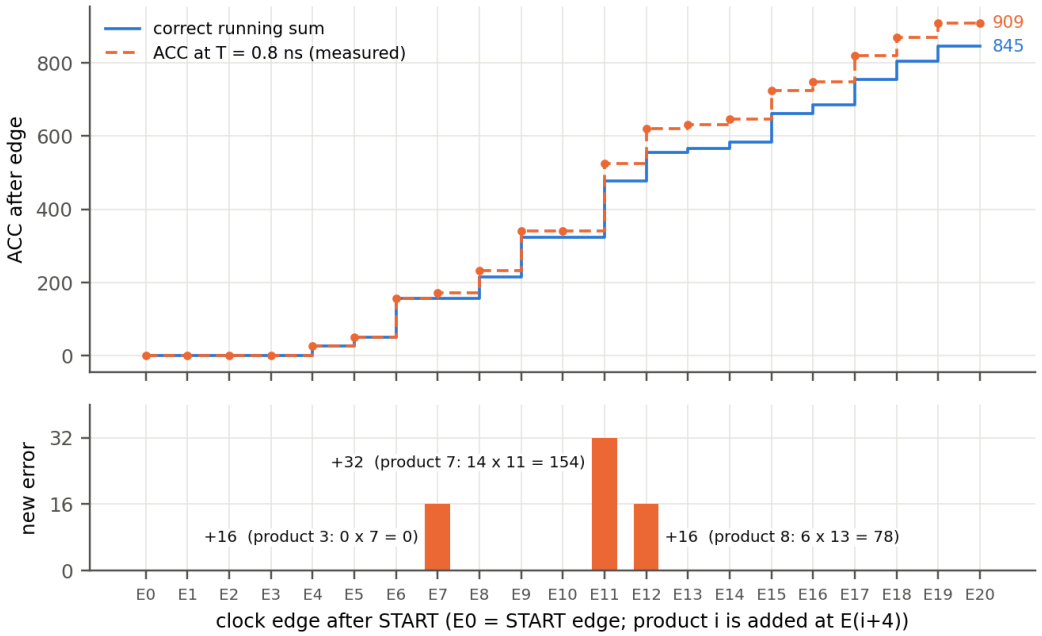


Figure 19: Test A at $T = 0.8$ ns: `ACC` after each compute cycle against the correct running sum. The errors appear in cycles 3, 7 and 8 and add up to +64.

5.3 Clock distribution

The CLK net carries 37 flops and both macros, about 100 fF. Our first driver, an `inv_u` followed by two `inv_drv_f`, could not drive it: CLK became a triangle wave from 65 to 690 mV and `DONE` came 0.35 cycles late. The bench now drives CLK through four inverting stages: V_CLK → `inv_u` → one `inv_drv_f` → 16 parallel `inv_drv_f` → 16 parallel `inv_drv_f` → CLK (Figure 20). CLK now swings from −6 to 806 mV with sharp edges, and the insertion delay from V_CLK to CLK is $t_{ins} = 233$ ps. Every flop and macro connects to the same CLK node (schematic, no wire RC), so the skew is 0; the insertion delay only shifts every edge by the same amount. The clock buffer runs from the bench supply, and its energy is reported separately (see Metrics).

5.4 Hold check

The fastest path is from the counter to the macros' address latches: `FC` → `MX` → `latch_lo`. The latch is transparent while CLK is low and closes at the rising edge. Its PMOS (gate CLK) turns off at once and its NMOS (gate CLK_b) one `inv_u` delay later, so its effective hold time is about 20 ps. The earliest change at the latch input after the edge is

$$t_{ccq} + t_{mux} = 71 + 42 \approx 110\,\text{ps}. \tag{12}$$

For hold, the fastest path is the worst case, so we take the faster of the two measured mux delays (42 ps from A) rather than the delay from B, where `CNT` actually enters (45 ps). This leaves a hold margin of about **90 ps**. The counter and the latches share the clock node, so there is no skew to subtract.

The register-to-register paths are safe even with no logic at all: the valid chain `FV<1:3>` connects Q to D directly, and $t_{ccq} = 71$ ps exceeds $t_{hold} = -49$ ps by 120 ps. The memory output is also safe: `DOUT` holds until the next read's sense enable opens, about 268 ps after the edge at which `RW/RX`

capture it, against a hold time of −49 ps. Functionally, every run from 0.9 to 1.2 ns returns the exact result, which a corrupted address would break, and the errors at 0.8 ns cannot come from a misread word (above).

6 Verification (3.5)

6.1 Bench

`tb_mac` (Figure 20) instantiates `mac` with both macros bound to the extracted array through a config view (`sram_array` → `av_extracted`; every other cell uses its schematic). Twelve PWL sources drive `RST`, `START`, `WE_W`, `WE_X`, `ADDR<3:0>` and `DIN<3:0>`, each through two `inv_u`; `ACC<11:0>` and `DONE` each drive an `inv_u`. The stimulus (`scripts/mac_stim.py`) changes inputs at kT, halfway between rising edges of the source clock. Because CLK reaches the macros $t_{ins} = 233$ ps after the source edge, the inputs get about $T/2 + 0.23$ ns of setup, less the input-buffer delay, and $T/2 - 0.23$ ns of hold (217 ps at $T = 0.9$ ns), plus that delay. The stimulus sequence is:

- cycles 0–1: `RST`;
- cycles 2–17: write $W[0..15]$ (`WE_W` = 1);
- cycles 18–33: write $X[0..15]$ (`WE_X` = 1);
- cycle 34: idle; cycle 35: `START`, sampled at the edge at $35.5\,T$ (E_0);
- `DONE` expected just after $54.5\,T$ (E_{19}); the run stops at $60\,T$.

`ACC` is read with `ACC_dig`, which rounds each bit to 0 or 1 and weights it by 2^b, at $57.75\,T + 0.25$ ns: about a quarter cycle after E_{22}, the third edge after `DONE`.

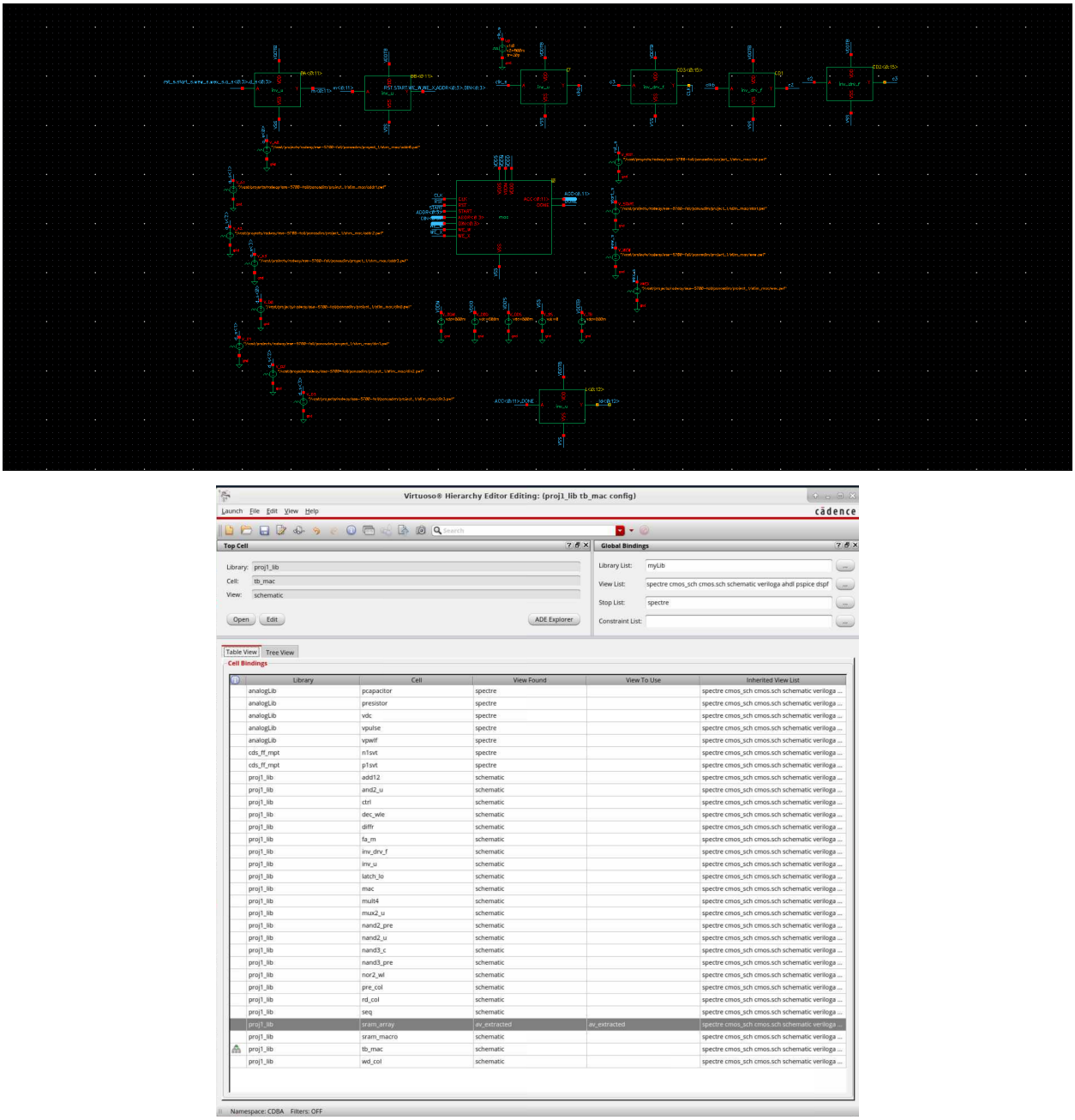


Figure 20: Top: `tb_mac`. Left: the twelve PWL sources and the input buffers `BA<0:11>`, `BB<0:11>`; top right: the clock chain, `inv_u I7`, `CD1`, `CD2<0:15>` and `CD3<0:15>` (`inv_drv_f`); centre: `mac`; bottom: the supplies VDDM, VDDD, VDDS, VSS and VDDTB and the output loads `L<0:12>`. All bench drivers and loads are on VDDTB. Bottom: the config view, with `sram_array` bound to `av_extracted`.

6.2 Test vectors and results

The three tests of the specification:

- **A:** $W = 3, 12, 7, 0, 15, 9, 1, 14, 6, 11, 2, 13, 8, 5, 10, 4$; $X = 9, 2, 15, 7, 4, 12, 0, 11, 13, 1, 8, 6, 3, 14, 5, 10$; expected 845 (`0x34D`).

- **B:** all $W = X = 15$, the full-scale case; expected 3600 (`0xE10`).
- **C:** $W = 5, 10, 5, 10, \ldots$; $X = 10, 5, 10, 5, \ldots$, which toggles every memory output and multiplier input every cycle; expected 800 (`0x320`).

T	Test A (845)	Test B (3600)	Test C (800)
1.2 ns	pass (845)	pass (3600)	pass (800)
1.0 ns	pass (845)	–	–
0.9 ns	**pass (845)**	**pass (3600)**	**pass (800)**
0.8 ns	**fail (909)**	–	–

Table 7: Pass/fail for the three tests, full MAC with the extracted SRAM arrays; – marks a run we did not do (Section 5). In every run `DONE` rises 19.06–19.09 cycles after E_0: 19 cycles plus the `DONE` flop's clock-to-Q.

`DONE` rises at 65.7 ns at $T = 1.2$ ns and at 49.35 ns at $T = 0.9$ ns. Both equal $54.5\,T + t_{ins} + t_{cq}$, for example $54.5 \times 0.9 + 0.233 + 0.07 = 49.35$ ns.

6.3 Test A waveforms

Figure 21 shows `ACC` and `DONE` for Test A at $T = 1.2$ ns. `ACC` starts from the simulator's arbitrary initial state, is cleared by `RST`, stays at 0 through the load phase and the first four compute cycles while the pipeline fills, and then climbs to 845, where it stays. There are sixteen additions, but two of them add 0 (0×7 and 1×0), so the staircase has fourteen visible steps. `DONE` rises at 65.70 ns, after the last step, and holds. In the same run, ADE reports 19.06 cycles from E_0 to `DONE`, CLK from -5.9 to 806.2 mV and $t_{ins} = 232.7$ ps.

We exported the `ACC` waveform of Test A only at 1.2 ns. At $T_{CLK,min} = 0.9$ ns, Test A is checked by its final value (Figure 22: `ACC_dig` = 845, `DONE` at 49.35 ns), and the cycle-by-cycle staircase at that period is shown for Test B in Figure 18. The step-by-step values of `ACC` follow the cycle model of Figure 17.

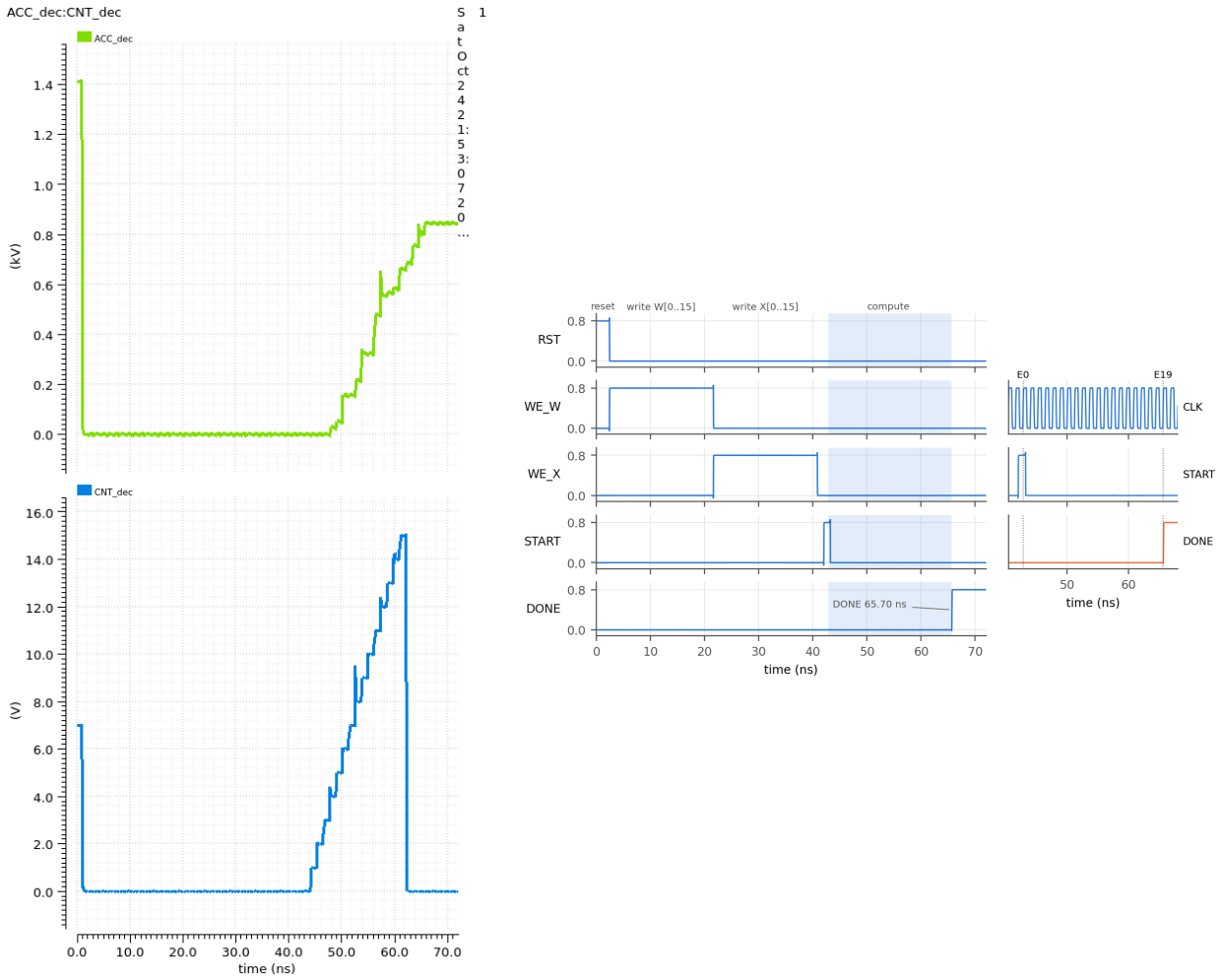
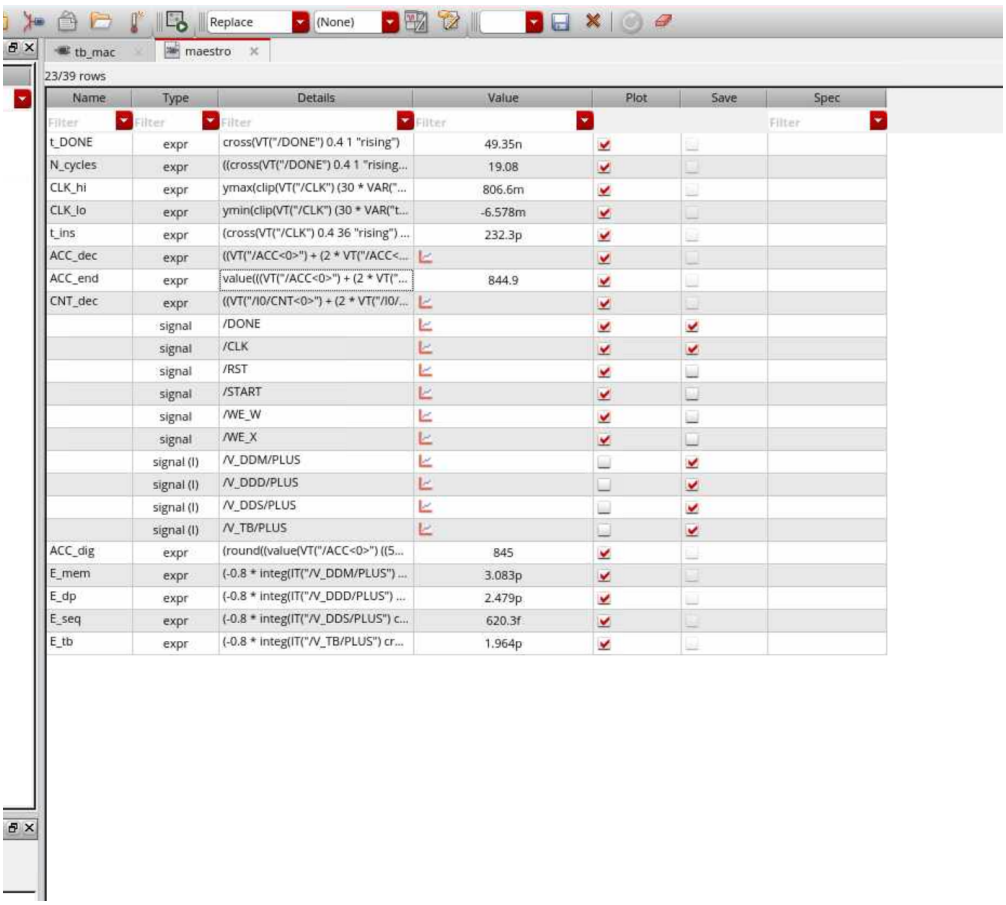


Figure 21: Test A in `tb_mac`, $T = 1.2$ ns. Left: decoded `ACC` and `CNT` in ADE (the axis is labelled kV because the decode divides voltages by 0.8 V; 0.845 k is 845). The short spikes on `ACC` occur when several bits switch on one edge and are sampled by the analog decode at slightly different times. Right: the control inputs and `DONE` over the whole run (reset, writes of W and X, compute) and, on the right, CLK, `START` and `DONE` from E_0 to E_{19}; `DONE` rises at 65.70 ns.



Name	Type	Details	Value	Plot	Save	Spec
t_DONE	expr	cross(VT("/DONE") 0.4 1 "rising")	49.35n	☑	☐	
N_cycles	expr	((cross(VT("/DONE") 0.4 1 "rising...	19.08	☑	☐	
CLK_hi	expr	ymax(clip(VT("/CLK") (30 * VAR("...	806.6m	☑	☐	
CLK_lo	expr	ymin(clip(VT("/CLK") (30 * VAR("t...	-6.578m	☑	☐	
t_ins	expr	(cross(VT("/CLK") 0.4 36 "rising") ...	232.3p	☑	☐	
ACC_dec	expr	((VT("/ACC<0>") + (2 * VT("/ACC<...		☑	☐	
ACC_end	expr	value(((VT("/ACC<0>") + (2 * VT("...	844.9	☑	☐	
CNT_dec	expr	((VT("/I0/CNT<0>") + (2 * VT("/I0/...		☑	☐	
	signal	/DONE		☑	☑	
	signal	/CLK		☑	☑	
	signal	/RST		☑	☐	
	signal	/START		☑	☐	
	signal	/WE_W		☑	☐	
	signal	/WE_X		☑	☐	
	signal (I)	/V_DDM/PLUS		☐	☑	
	signal (I)	/V_DDD/PLUS		☐	☑	
	signal (I)	/V_DDS/PLUS		☐	☑	
	signal (I)	/V_TB/PLUS		☐	☑	
ACC_dig	expr	(round((value(VT("/ACC<0>") ((5...	845	☑	☐	
E_mem	expr	(-0.8 * integ(IT("/V_DDM/PLUS") ...	3.083p	☑	☐	
E_dp	expr	(-0.8 * integ(IT("/V_DDD/PLUS") ...	2.479p	☑	☐	
E_seq	expr	(-0.8 * integ(IT("/V_DDS/PLUS") c...	620.3f	☑	☐	
E_tb	expr	(-0.8 * integ(IT("/V_TB/PLUS") cr...	1.964p	☑	☐	

Figure 22: Test A at $T_{CLK,min} = 0.9$ ns, ADE outputs: `ACC_dig` = 845 (`ACC_end` = 844.9 is the raw weighted sum of the bit voltages), $t_{DONE} = 49.35$ ns, 19.08 cycles, CLK from −6.6 to 806.6 mV, $t_{ins} = 232.3$ ps, and the energies of the three MAC supplies and the bench supply.

Metrics and figure of merit

All metrics are for the full MAC with the extracted SRAM arrays at $T_{CLK,min} = 0.9$ ns and $V_{DD} = 0.8$ V.

Metric	Value
V_{DD}	0.8 V
Density factor D (Part 1 cell)	79
$T_{CLK,min}$ (A fails at 0.8 ns)	0.9 ns
N_{cycles}, `START` to `DONE`	19
$t_{dot} = N_{cycles}\, T_{CLK,min}$	$19 \times 0.9 = 17.1$ ns
E_{dot}, Test A, MAC supplies (VDDM + VDDD + VDDS)	6.2 pJ
Clock distribution, Test A (VDDTB)	2.0 pJ
E_{dot}, Test A, MAC supplies + clock	8.1 pJ
Energy per MAC, Test A, with memory (MAC supplies / + clock)	0.39 / 0.51 pJ
Energy per MAC, Test A, without memory (VDDD + VDDS / + clock)	0.19 / 0.32 pJ
Total area	505 µm^2
$\text{FOM}_{MAC} = \text{Area} \cdot E_{dot} \cdot t_{dot}$, with clock	7.0×10^4 µm^2 · pJ · ns
FOM_{MAC}, MAC supplies only	5.3×10^4 µm^2 · pJ · ns

Table 8: Part 3 metrics, to two significant figures (the area and the exact t_{dot} to three). Table 9 gives the energies to three; the FOM is computed from those.

Delay. $t_{dot} = 19 \times 0.9\,\text{ns} = 17.1\,\text{ns}$. The measured interval from the `START` edge to `DONE` is 19.08 cycles, 17.2 ns; the extra 0.08 cycle is the `DONE` flop's clock-to-Q.

Energy. Each supply's energy is $-V_{DD} \int i\, dt$ from the rising CLK edge that samples `START` to `DONE` rising.

Test	E_{mem} (VDDM)	E_{dp} (VDDD)	E_{seq} (VDDS)	E_{dot} (MAC)	Clock (VDDTB)
A	3.08	2.48	0.62	**6.18**	1.96
B	3.40	1.49	0.62	5.51	1.96
C	3.08	2.53	0.62	6.23	1.97

Table 9: Energy per dot product at $T = 0.9$ ns, in pJ. E_{dot} is the sum of the three MAC supplies. With the clock, Test A totals 8.15 pJ (unrounded: $3.083 + 2.479 + 0.620 + 1.964$, Figure 22). At $T = 1.0$ ns, Test A gives 3.10, 2.47 and 0.62 pJ.

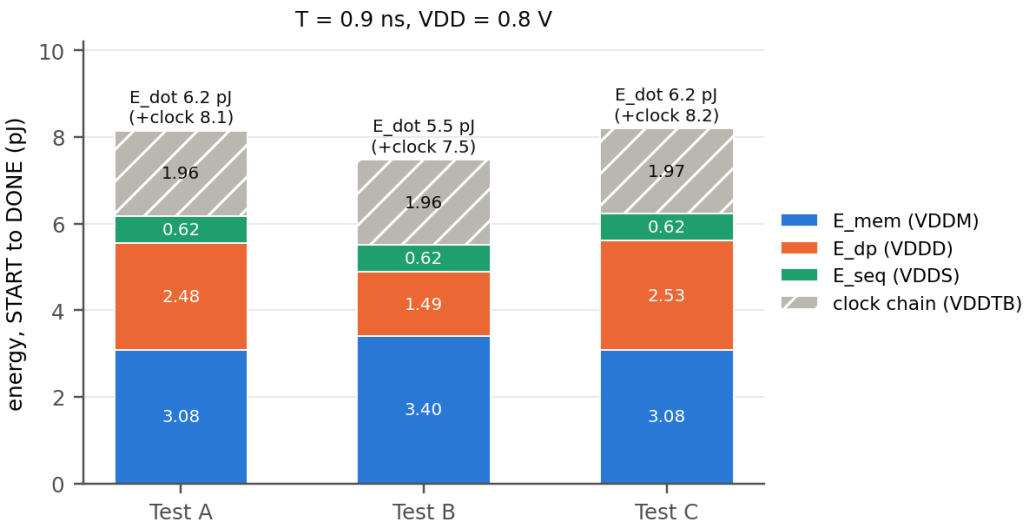


Figure 23: Energy per dot product by supply at $T = 0.9$ ns.

Energy per MAC. One dot product is 16 multiply-accumulates. For Test A, $6.18/16 = 0.39$ pJ per MAC with the memory reads, or $8.15/16 = 0.51$ pJ including the clock. Without the memory accesses, $E_{dp} + E_{seq} = 2.48 + 0.62 = 3.10$ pJ, or 0.19 pJ per MAC ($(3.10 + 1.96)/16 = 0.32$ pJ with the clock).

Where the energy goes. The two memories take about half of E_{dot} (3.08 of 6.18 pJ in Test A), the datapath about 40% and the sequencer 10%. The energy is dynamic and set by switching activity, not by the period: at 1.0 ns Test A uses the same energy within 1%. The three tests show which activity costs what.

- **Memory: B is highest** (3.40 pJ). Our read path senses $\overline{\text{BL}}$ single-ended, and reading a 1 discharges $\overline{\text{BL}}$. With all-1 words, every read swings the bitline of every column and the precharge must restore it.
- **Datapath: B is lowest** (1.49 pJ). $W = X = 15$ in every word, so the operand registers and the multiplier inputs never toggle; only the product register (cleared and loaded by V_2), the adder and the accumulator switch.
- **Datapath: C is highest** (2.53 pJ), although every product is 50: W alternates between 0101 and 1010 and X the opposite way, so every operand bit toggles every cycle.
- **Sequencer: flat** at 0.62 pJ. The counter and control do the same work for any data. Its nine flops are clocked every cycle; the four address muxes and the two write-enable gates are also on VDDS.

The clock is a real cost. The bench clock chain uses 1.96 pJ, which is $1.96/19 = 0.10$ pJ per cycle, or 0.10 pJ/$V_{DD}^2 \approx 160$ fF switched every cycle (the CLK net and the buffers). That is a quarter of the total (1.96 of 8.15 pJ). The MAC cannot run without charging its clock net, so our primary figure of merit includes it.

Area. The memories use the Part 2 macro area: the laid-out array, 33.74 µm^2, plus the periphery estimated as $D\sum W_iL_i = 77.9\,\mu\text{m}^2$, so 111.6 µm^2 per macro. The logic outside the macros is estimated the same way from the `tb_mac` netlist (1,864 transistors), with $D = 79$, $L = 18$ nm and $W = 0.062$ µm for 2 fins plus 0.048 µm per extra fin (Table 10).

Block	Instances	$D\sum W_iL_i$ (µm^2)
Sequencer (9 `diffr`, incrementer, logic)	`seq`	52.7
Address muxes	4 × `mux2_u`	9.1
Write-enable gating	2 × `and2_u`	2.0
W, X registers	8 × `diffr`	23.5
Product register	8 × `diffr`	23.5
Accumulator register	12 × `diffr`	35.2
Multiplier	`mult4`	76.0
Adder	`add12`	59.9
Logic total		281.9
Two SRAM macros	$2 \times (33.74 + 77.9)$	223.3
Total		**505**

Table 10: Area. The logic is 56% of the total; the 37 flops alone are $37 \times 2.9 \approx 109\,\mu\text{m}^2$ (28 in the datapath, 9 in the sequencer).

Figure of merit. With the clock included,

$$\text{FOM}_{MAC} = 505\ \mu\text{m}^2 \times 8.15\ \text{pJ} \times 17.1\ \text{ns} = 7.0 \times 10^4\ \mu\text{m}^2 \cdot \text{pJ} \cdot \text{ns}, \tag{13}$$

and with the MAC supplies only, $505 \times 6.18 \times 17.1 = 5.3 \times 10^4\ \mu\text{m}^2 \cdot \text{pJ} \cdot \text{ns}$.

What limited the design

The ripple-carry datapath is the main limit. Each carry stage takes about 67 ps, and both the 12-bit accumulator and the 4×4 multiplier take about 0.89 ns in their slow cases. This is longer than the 0.7 ns period of the SRAM macros. At 0.8 ns, the errors are single powers of two, which is consistent with product or carry bits that had not settled before the clock edge. The design could be sped up by routing sum outputs into the lighter C_{in} input, replacing the upper accumulator bits with carry-select or carry-lookahead logic, using a carry-save multiplier, or adding one more pipeline stage. The clock is the second major cost: the roughly 100 fF clock net required two banks of 16 parallel drivers and uses about one quarter of the total dot-product energy.