PPU Version A — Noise Source, Weight Loading and Update Timing

Answers to five questions — each preceded by the block diagram or the real circuit it refers to

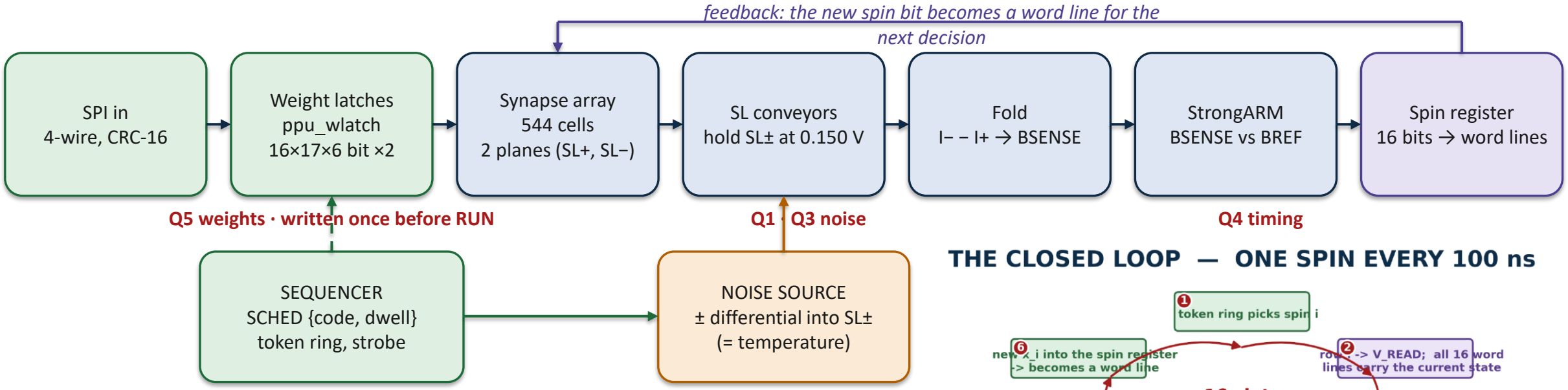
Probabilistic Processing Unit · TSMC 65 nm GP · 16-spin Ising / QUBO solver

Circuit slides are taken from PPU_integer_factorization_test (1).pptx (9 Aug 2026); block diagrams drawn from ANALOG_SPEC.md, the RTL and the post-layout measurements (26–27 Aug 2026)

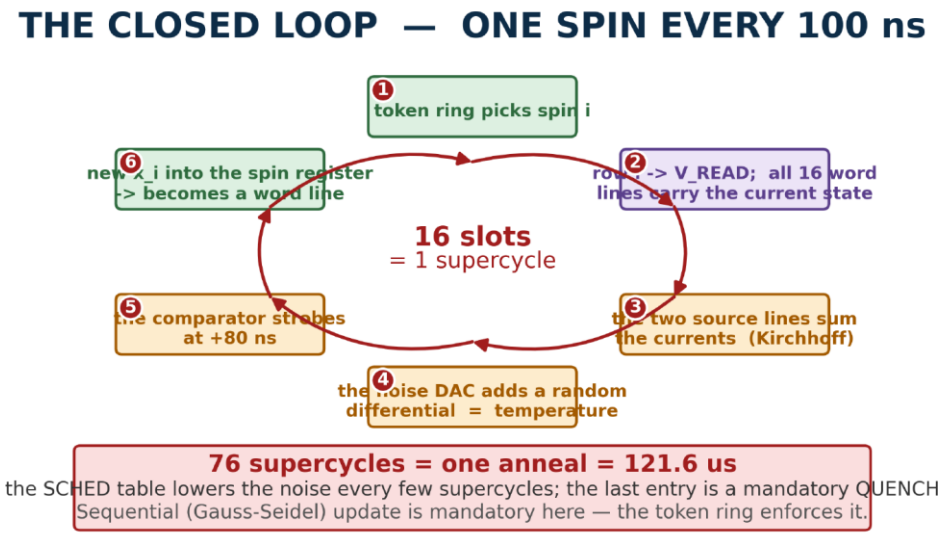
Prepared 9 Sep 2026

THE WHOLE SIGNAL PATH — WHERE EACH QUESTION LIVES

Block diagram (drawn from the design record) and the closed loop as presented in the deck



Read the chain left to right: a problem enters once (Q5), the array turns it into two currents, the noise source shakes the difference (Q1/Q3), the comparator decides one spin per slot (Q4), and that bit feeds straight back as a word line.



deck: THE CLOSED LOOP — one spin every 100 ns (16 slots = 1 supercycle)

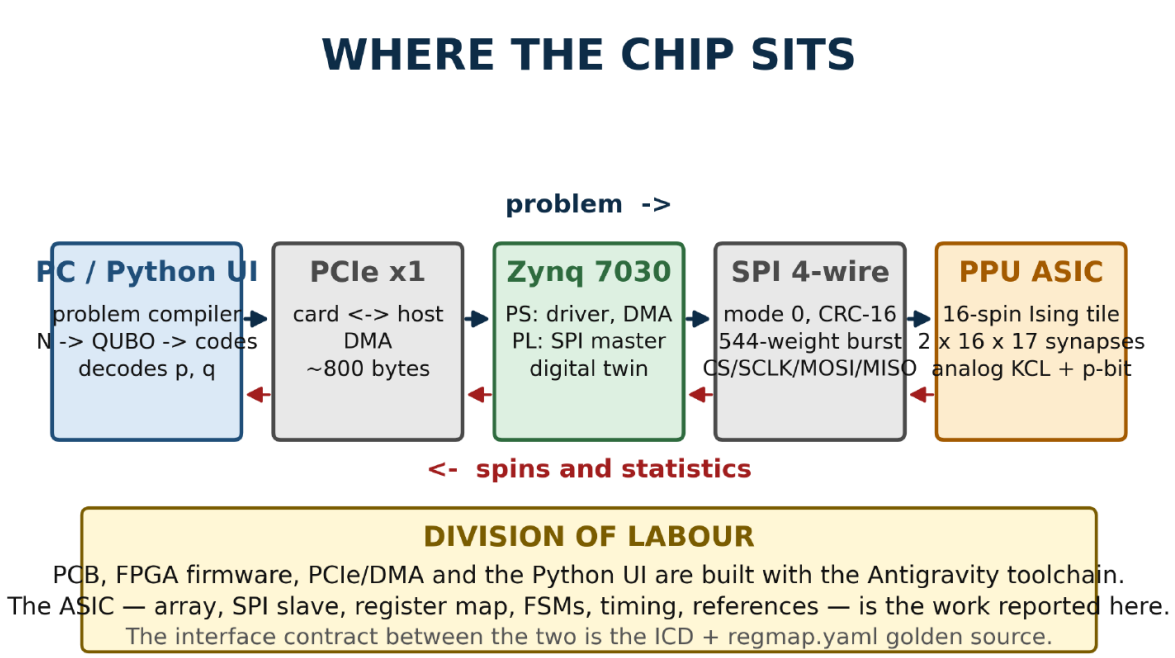
THE FIVE QUESTIONS — AND THE SHORT ANSWERS

#	Question	Short answer
1	Which noise source did we use?	Two answers: the 9-Aug presentation describes an LFSR-steered current DAC; the chip taped out on 26 Aug uses amplified THERMAL noise. The change was decided on 9 Aug.
2	Was it a TSMC 65 nm chip — a test ASIC before eFLASH?	Yes. Standard-CMOS bit array, TSMC CRN65GPLUS, 9 metal. Weights live in 6-bit digital latches, not floating gates.
3	Which noise sources exactly?	Presentation: 6 constant-sum legs, one LFSR bit each, 10.2 µA total. Taped-out: transistor thermal noise → 3-stage amplifier → 6-bit constant-sum steering into SL±, up to ~91 µA at the decision node.
4	Is the noise refreshed at every spin decision?	Yes in both. LFSR: pattern redrawn every slot. Thermal: physically fresh at every strobe. The AMPLITUDE (temperature) is NOT per spin — it steps per SCHED entry (every `dwell` supercycles).
5	Are the weights loaded once? From where?	Once, before RUN, and held. PC/Python compiler → PCIe → Zynq 7030 → 4-wire SPI (CRC-16, 544-weight burst) → on-chip ppu_wlatch (16×17×6 bit ×2 planes).

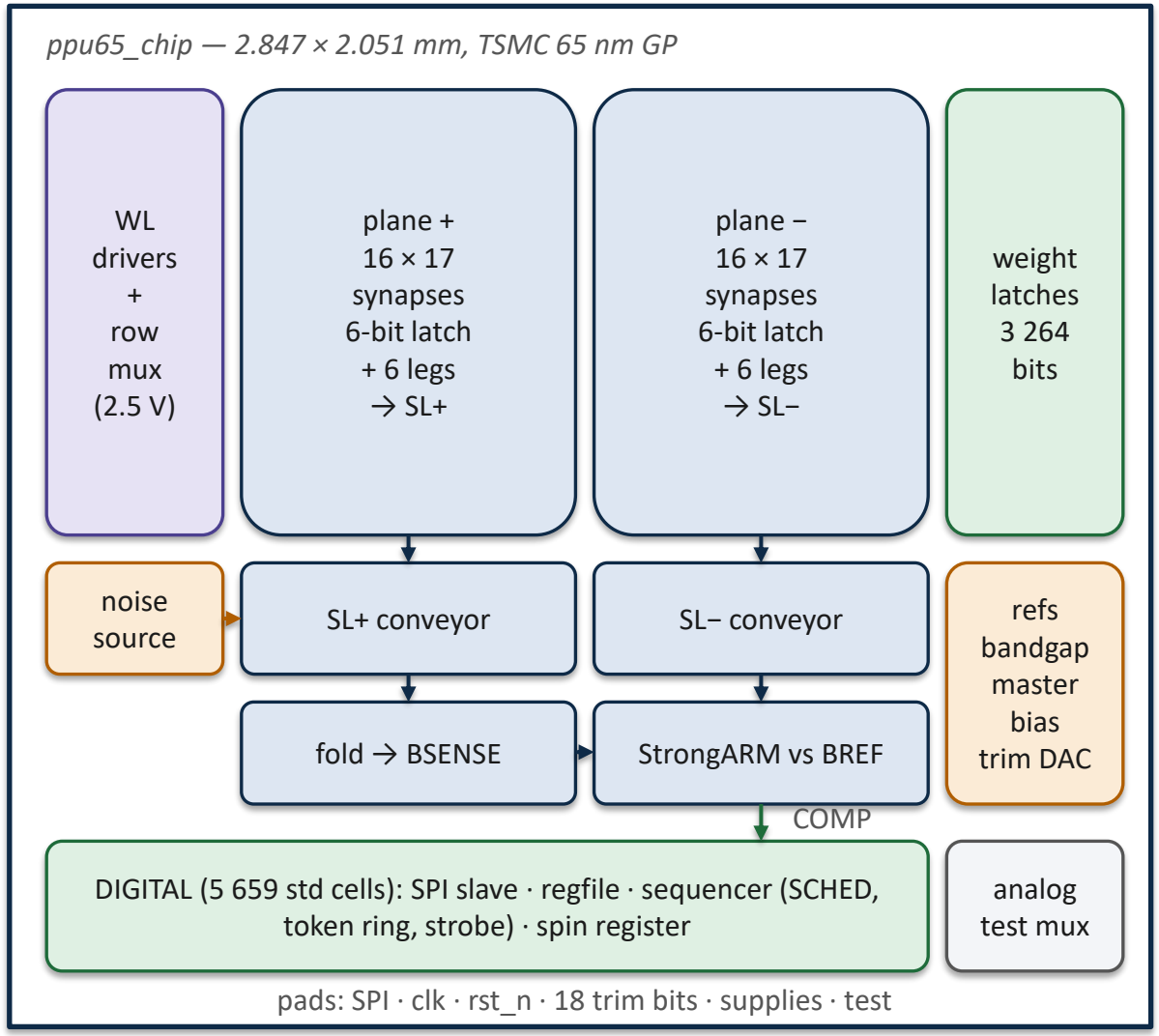
Every claim is traced to a file or a measurement; where the presentation and the taped-out chip differ, both are shown. Each answer slide is preceded by its diagram.

Q2 · DIAGRAM — WHERE THE CHIP SITS, AND WHAT IS ON THE DIE

Left: the board context as presented. Right: floor-plan-level block diagram of the standard-CMOS die (drawn)



deck: PC → PCIe → Zynq 7030 → SPI → PPU ASIC; only spins and statistics come back



no eFLASH cell anywhere: every weight is a latch + current legs — that is what makes it a pre-eFLASH test ASIC

Q2 — WHAT CHIP IS THIS? A STANDARD-CMOS TEST ASIC BEFORE eFLASH

Yes: TSMC 65 nm GP, no eFLASH, weights in latches

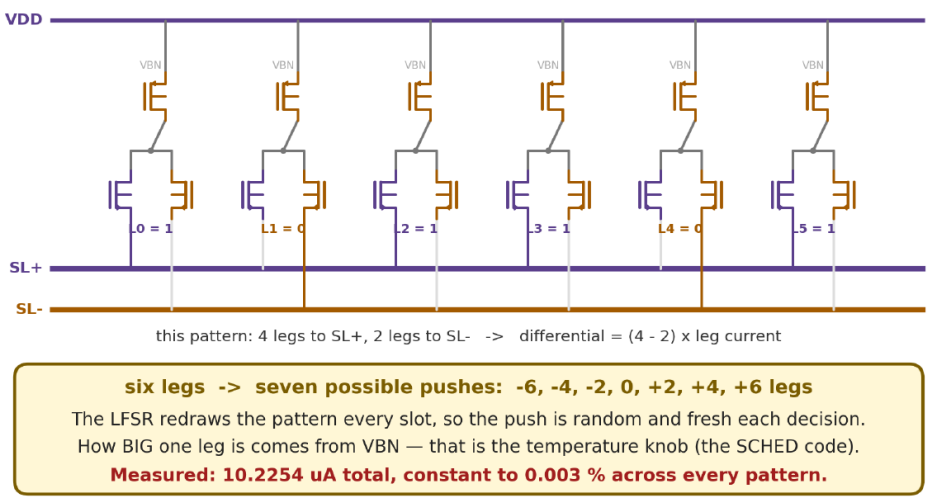
- Technology: TSMC CRN65GPLUS (65 nm GP), 9 metal layers (CN65S_9M), wire-bond, die 2.847 × 2.051 mm
- 16-spin Ising / QUBO tile: 544 synapses = 2 planes (SL+, SL−) × 16 rows × 17 columns (16 couplings + 1 bias column)
- Why standard CMOS — the design record says it explicitly (ANALOG_SPEC §0, decision dated 9 Aug 2026):
  - "Because the eFLASH PDK is not ready, the bit array is built from standard CMOS transistors (Version A was already so; the legacy FG-proxy netlists are reference only)."
- Consequence 1 — weights are DIGITAL: each synapse holds a 6-bit code in latches (ppu_wlatch), realised as six binary-weighted current legs (1:2:4:8:16:32)
- Consequence 2 — entropy cannot come from RTN of a floating gate (there is none); a separate noise source had to be designed — see Q1/Q3
- Presentation title confirms the same: "PPU — Probabilistic Processing Unit, Version A, TSMC 65 nm GP"

So: a fully working standard-CMOS annealer, built to prove the architecture before the eFLASH weight cell exists.

Q1 · DIAGRAM — THE TWO NOISE SOURCES SIDE BY SIDE

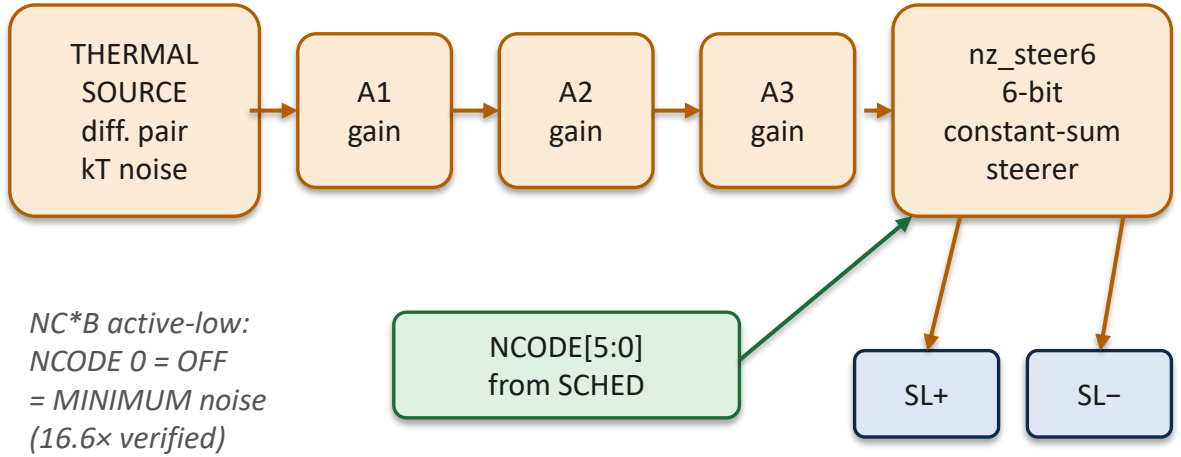
Left: the LFSR DAC exactly as drawn in the 9-Aug deck. Right: the thermal chain that was taped out (block diagram from ppu_noise_therm2.scs)

SIX LEGS TOGETHER — ppu_noise_dac



AS PRESENTED — ppu_noise_dac: six legs, one LFSR bit each, constant-sum steering, 10.2 µA total

nz_gain3 × 3 (3 differential stages)



*NC*B active-low:*
NCODE 0 = OFF
= MINIMUM noise
(16.6× verified)

AS TAPED OUT — same constant-sum idea, but the random bit is replaced by amplified thermal noise; the 6-bit code sets how much of it is steered into SL±

Same injection principle (move current, never add it) — different entropy: pseudo-random LFSR bits vs. true thermal noise

Proof the LFSR was not shipped: ppu_lfsr.v exists in digital/rtl/ but is not instantiated in ppu_digital_top; the P&R netlist (38 404 cells) contains 0 LFSR cells.

Q1 — THE NOISE SOURCE: TWO ANSWERS, ONE DAY APART

The presentation and the taped-out silicon describe different noise sources

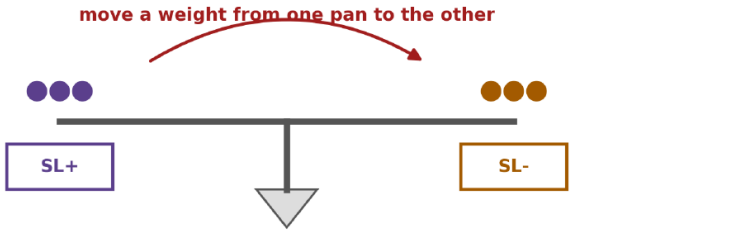
	Presentation (9 Aug 2026)	Taped-out chip (26 Aug 2026)
Cell name	ppu_noise_dac = 6 × ppu_noise_leg	ppu_noise_therm2 v1.3 (nz_gain3 ×3 + nz_steer6)
Entropy	Galois LFSR (pseudo-random, digital)	Transistor THERMAL noise (true random, analog)
Injection	Constant-sum steering: each leg lands on SL+ or SL−	Constant-sum steering: amplified noise steered into SL±
Amplitude control	VBN (leg size) set by SCHED code	6-bit NCODE from SCHED (NC*B active-low; NCODE 0 = minimum)
Full scale	10.2254 µA total (measured; constant to 0.003 %)	~91 µA at the decision node at NCODE 63 (post-layout, static)
Verdict in the deck	"Version A cannot shake hard enough" — needs ≥ ~40 µA	≥ 40 µA static-equivalent reached at NCODE ≈ 31 (51.5 µA); in-situ value at high codes not yet measured

Timeline: the deck is dated 9 Aug; ANALOG_SPEC §0 records the decision on the SAME day: "not RTN (no eFLASH) and NOT LFSR (true randomness wanted): the transistors' own thermal noise". Implemented 10 Aug (M5), shipped as v1.3.

Q3a · CIRCUITS — THE LFSR DAC AS DRAWN IN THE DECK

Left: what the noise must and must not do. Right: one leg (ppu_noise_leg) — one pch current source, two nch switches, one LFSR bit

WHAT THE NOISE HAS TO DO — AND WHAT IT MUST NOT



the total must not change

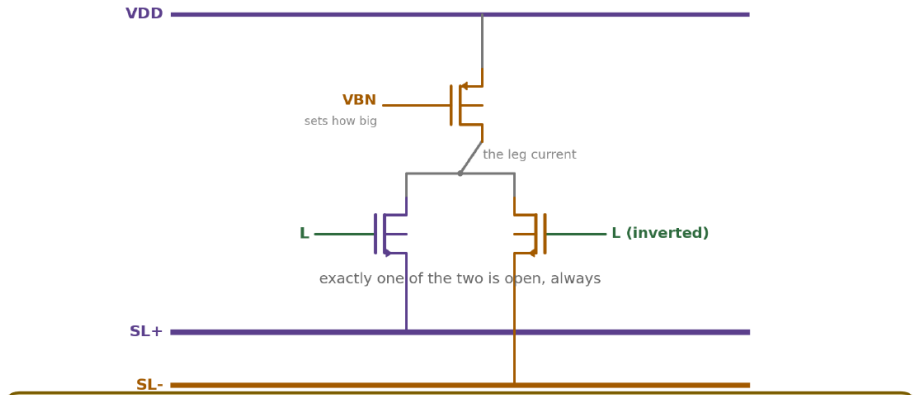
if the noise added current, it would look like extra weight and shift every decision.
If it only MOVES current between the two wires, the sum is untouched
and only the difference — the thing being judged — is randomised.

So the noise source is built as a set of legs that always conduct, and are merely steered left or right.

That is why it is called a constant-sum steering DAC.

the balance: a weight moved from one pan to the other — the total never changes

ONE LEG — ppu_noise_leg



one pch current source, two nch switches, one LFSR bit

The pch never stops conducting. The bit only decides which wire the current lands on.

exactly one switch is open, always: the current lands on SL+ or SL−, never nowhere

Six of these legs = ppu_noise_dac (previous diagram): seven possible pushes −6 … +6 legs, redrawn by the LFSR every slot

Q3a — THE NOISE SOURCE AS PRESENTED: LFSR-STEERED CONSTANT-SUM DAC

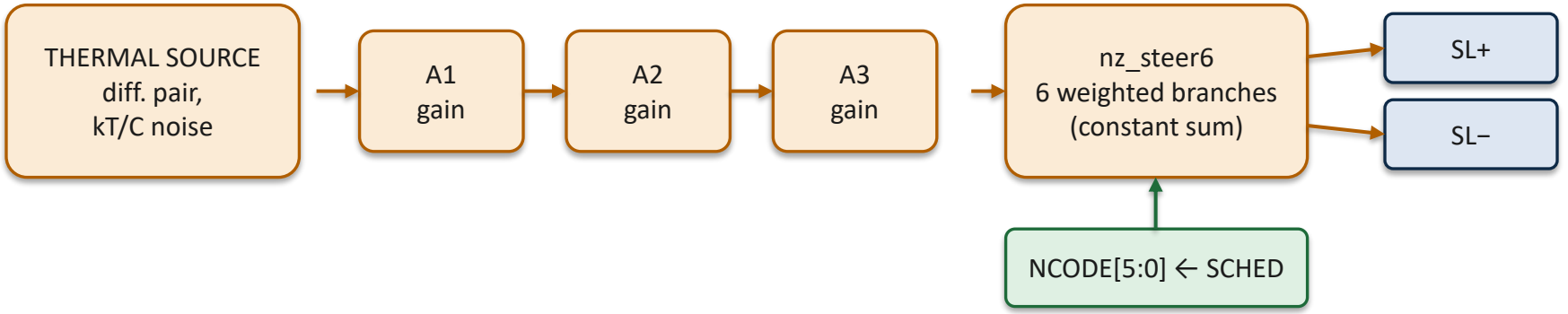
"the temperature as a current" — deck pages 75–77

- One leg (ppu_noise_leg): one pch current source (VBN sets its size) + two nch switches driven by L and L-inverted
  - Exactly one switch is open at all times → the leg current lands on SL+ or on SL−, never nowhere
- Six legs together (ppu_noise_dac): one LFSR bit per leg
  - k legs to SL+, 6−k to SL− → differential = (2k − 6) × leg current → seven pushes: −6, −4, −2, 0, +2, +4, +6 legs
- Why constant-sum: if the noise ADDED current it would look like extra weight and bias every decision; it only MOVES current, so the sum is untouched and only the difference — the thing being judged — is randomised
- Measured: 10.2254 µA total, constant to 0.003 % across every LFSR pattern
- Timing: "The LFSR redraws the pattern every slot, so the push is random and fresh each decision"
- Temperature knob = VBN, i.e. the SCHED code sets how big one leg is

The limit this problem exposed: factorization needs A_max ≈ 40 µA; Version A silicon had 10.2 µA (sized for ferro4 / af4, which solve by a plain quench).

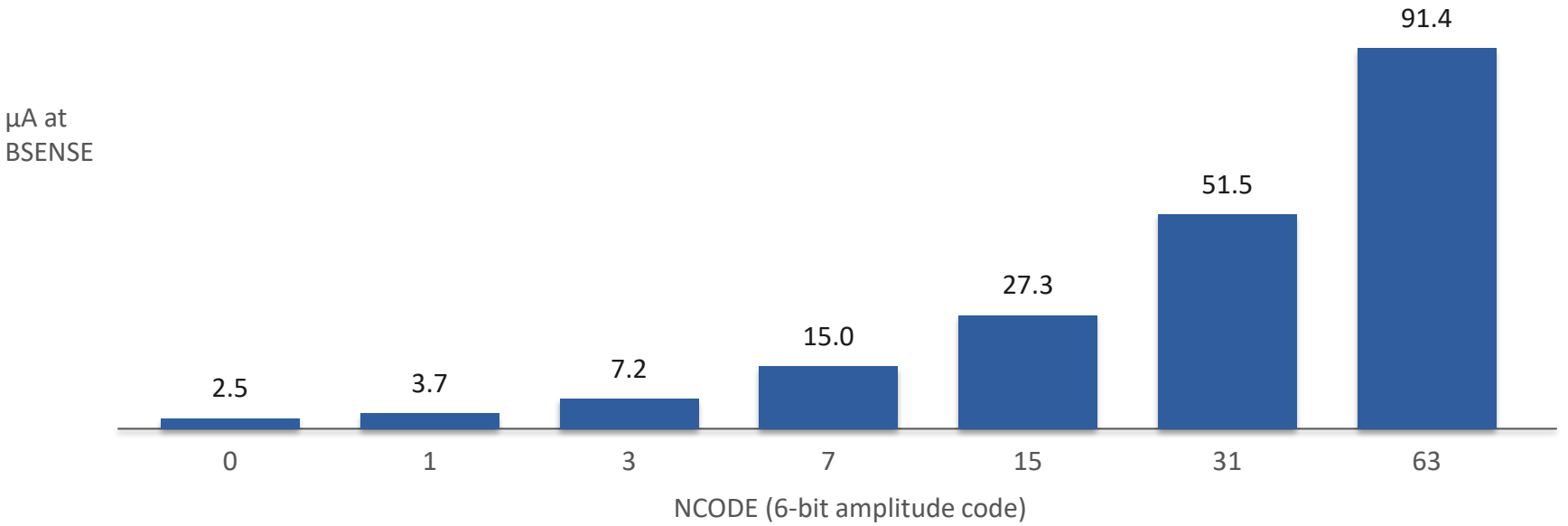
Q3b · DIAGRAM — THE THERMAL CHAIN, AND HOW LOUD EACH CODE IS

Top: ppu_noise_therm2 block by block. Bottom: measured noise amplitude at the decision node for each NCODE (post-layout, static)



Bias VCM/VB1/VBT from the real bandgap → master bias → R-string chain. NC*B is active-low: NCODE 0 → switches OFF → minimum noise (16.6× verified).

Noise amplitude at the decision node vs NCODE (µA equiv., 3.6 mV/µA; static)



reading the bars
monotonic, ≈ 1.8× per bit
0.34 → 12.3 weight units
≥ 40 µA between NCODE 15 and 31
caveat: static values — in-situ σ at the decision instant is 2.4× lower at NCODE 0 (N = 24)

Source: analog/sim/sigma_bsense_dac.txt (post-layout ppu_fold_sa, real bias chain, tt/27); methodology caveat in analog/sim/karar_tohum_sayimi.txt

Q3b — THE NOISE SOURCE AS TAPED OUT: AMPLIFIED THERMAL NOISE

ppu_noise_therm2 v1.3 — analog/netlists/ppu_noise_therm2.scs

- Entropy: the transistors' own thermal noise — no LFSR, no RTN. ANALOG_SPEC §0: "gerçek rastgelelik istenir" (true randomness is wanted)
- Chain: 3 differential gain stages (nz_gain3: A1 → A2 → A3) → nz_steer6, a 6-bit constant-sum current steerer into SL+ / SL−
- Amplitude = 6-bit NCODE from the sequencer's SCHED table. Polarity trap recorded: NC*B is active-low; NCODE = 0 → switches OFF → MINIMUM noise (verified 16.6× ratio)
- Post-layout, static operating point, σ at the decision node BSENSE (3.6 mV/µA):

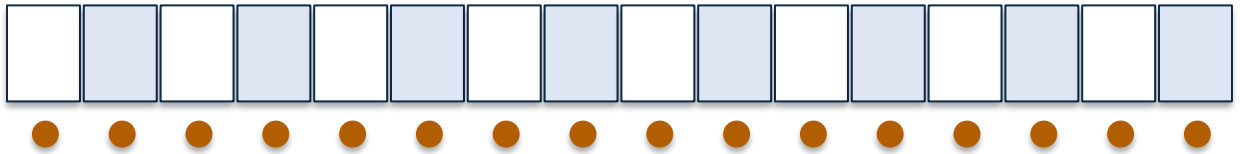
NCODE	0	1	3	7	15	31	63
σ(BSENSE) mV	9.1	13.2	26.0	54.0	98.2	185.4	328.9
equivalent µA	2.5	3.7	7.2	15.0	27.3	51.5	91.4
in weight units (7.42 µA)	0.34	0.49	0.97	2.02	3.67	6.94	12.3

Monotonic, ~1.8× per bit. Methodology note (27 Aug): these are STATIC values; the in-situ σ at the decision instant is ~2.4× smaller (NCODE 0: 5.0 mV, N=24 seeds). Use the trend, not the absolute values, for a margin budget.

Q4 · DIAGRAM — WHEN THE NOISE CHANGES: EVERY SLOT vs EVERY SCHED STEP

Left: one supercycle of 16 slots and the SCHED staircase (drawn). Right: the anneal as presented in the deck

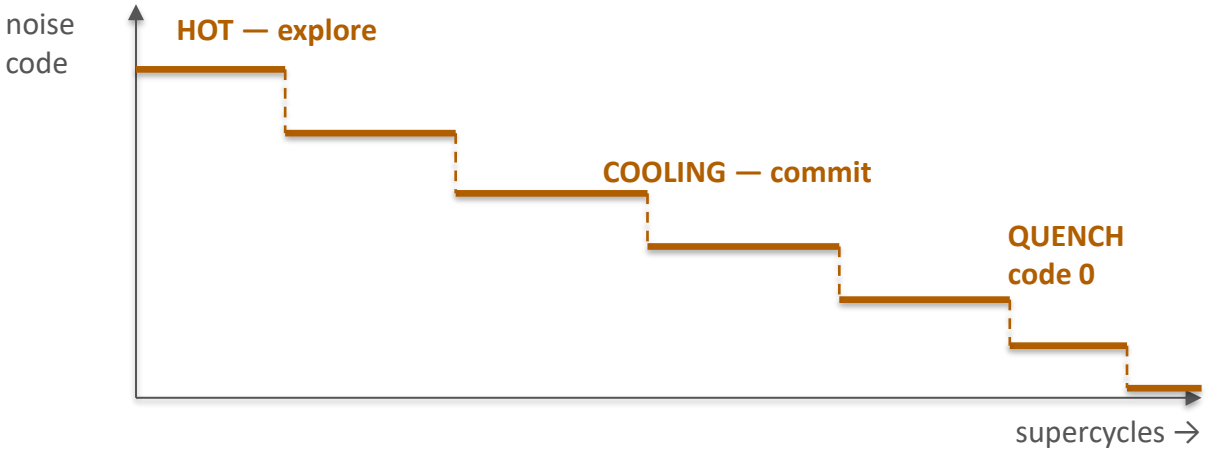
one supercycle = 16 slots (one decision per slot)



● = a fresh random push in every slot (LFSR redraw / physical noise)

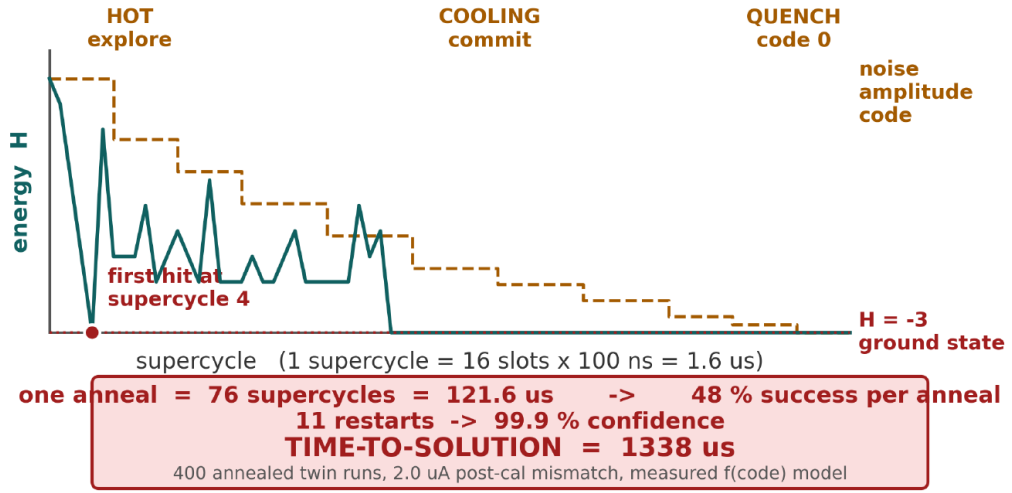
spin 0 · spin 1 · … · spin 15 → each new bit becomes a word line before the next decision (Gauss–Seidel)

SCHED = {noise code, dwell}: the code holds for `dwell` supercycles, then steps



a step = one SCHED entry, `dwell` × 16 slots wide — the random push is renewed inside every one of those slots

THE ANNEAL, SUPERCYCLE BY SUPERCYCLE



deck: the anneal supercycle by supercycle — 76 supercycles = 121.6 µs, staircase from HOT to QUENCH

Two clocks: the random value changes every slot; the temperature changes every SCHED step

Q4 — IS THE NOISE REFRESHED AT EVERY SPIN DECISION?

Two different things: the random draw (per slot) and the temperature (per SCHED step)

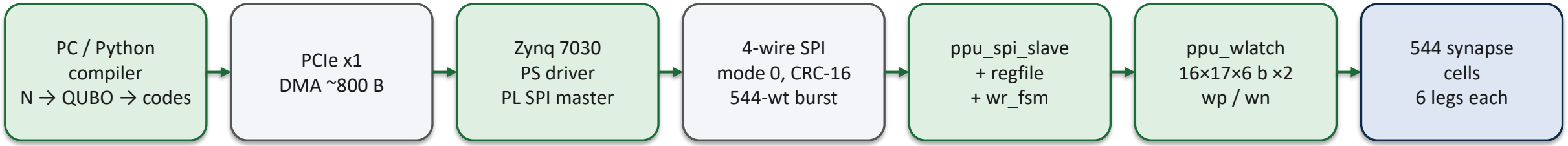
	Presentation (LFSR DAC)	Taped-out chip (thermal)
Random value per decision?	YES — "the LFSR redraws the pattern every slot"	YES — a physical noise process is fresh at every strobe by nature
Independence between consecutive slots	Guaranteed by the LFSR sequence	Measured: τ rises to 83 ns at high codes → ~23 % correlation between consecutive decisions at the hot end (open item)
Amplitude / temperature per decision?	NO — VBN follows the SCHED code	NO — NCODE follows the SCHED code

- How the temperature actually steps (ppu_sequencer.v): sched_data = {noise_code[5:0], dwell[9:0]}. The code is held for `dwell` supercycles (16 spins each), then the next SCHED entry is loaded. The last entry is a mandatory QUENCH (code 0).
- Worked example in the deck: 76 supercycles = 121.6 µs at 100 ns/slot, staircase from HOT (explore) → COOLING (commit) → QUENCH.
- Update order: sequential Gauss–Seidel, enforced by the token ring — each new spin bit becomes a word line before the next spin is decided.

So: a new random push at every slot, but the SAME temperature for a whole dwell — that is what makes it an anneal rather than a random walk.

Q5 · DIAGRAM — HOW A PROBLEM GETS INTO THE CHIP, AND WHERE IT STAYS

Top: the write path, host to latch (drawn from the deck's board slide and the RTL). Bottom: the two tables that ARE the problem, as presented



written ONCE before RUN — then only the 16-bit spin register moves; readback = spin register only (SPI → PCIe → Python decodes p, q)

SO A WHOLE PROBLEM IS JUST TWO TABLES OF SMALL NUMBERS

table for wire +

	p1	p2	q1	q2	q3	w11	w12	w13
p1			6	3	3			
p2			3	6	3	3		
q1	6	3						
q2	3	6				3		
q3	3	3					3	
w11		3		3				
w12					3			
w13								

(the top-left corner of the real 16 x 17 table)

table for wire -

	p1	p2	q1	q2	q3	w11	w12	w13
p1						6	6	6
p2								
q1						6		
q2							6	
q3								6
w11	6		6					
w12	6			6				
w13	6				6			

(the top-left corner of the real 16 x 17 table)

Each square is one cell of the chip; the number in it is that cell's code.
Empty means code 0 — that cell contributes nothing.

Loading a problem into the chip = writing these numbers in. Nothing else changes.

the same silicon solves any problem — only the tables differ

row label = the spin this row decides column label = the other spin it is coupled to

deck: plane + and plane − tables (top-left of the real 16×17)

THE EXTRA 17th COLUMN: EACH SPIN'S OWN PREFERENCE

sixteen columns are the couplings to the other spins. The seventeenth is the bias a_i.

spin	bias a_i	code on +	code on -	difference
p1	+1	9 (59.2 uA)	8 (52.7 uA)	**+6.5 uA**
w11	+5	15 (97.7 uA)	8 (52.7 uA)	**+45.0 uA**
c3_0	+9	21 (135.6 uA)	8 (52.7 uA)	**+82.9 uA**

why is there a big number on BOTH sides?

both planes carry the same fixed pedestal (code 8 = 52.7 uA) so that neither wire is ever empty.
Since only the DIFFERENCE matters, the pedestal cancels exactly and changes nothing.

What survives is precisely the bias a_i.

So the two tables together hold the whole problem: every coupling Q_ij and every bias a_i.

deck: the 17th column — the bias a_i on a cancelling pedestal (code 8 on both planes)

Q5 — WEIGHTS: LOADED ONCE, HELD IN LATCHES, WRITTEN OVER SPI

"Loading a problem into the chip = writing these numbers in. Nothing else changes."

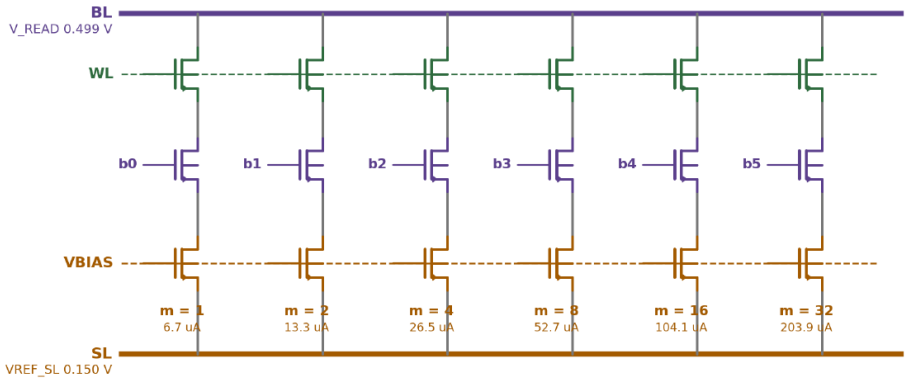
- Loaded ONCE before RUN and never touched during the anneal — only the 16 spin bits change
- Where they come from (deck slide "Where the chip sits"):
  - PC / Python compiler (N → QUBO → 6-bit codes) → PCIe x1 DMA (~800 B) → Zynq 7030 (PS driver, PL SPI master) → 4-wire SPI, mode 0, CRC-16, 544-weight burst → PPU SPI slave
- Where they live on chip: ppu_wlatch — 16 rows × 17 columns × 6 bits × 2 planes = 3 264 latch bits
  - Plane + (wp) feeds SL+, plane − (wn) feeds SL−; column 17 is the bias a_i on a cancelling pedestal (code 8 = 52.7 µA on both planes)
- Write protocol (RTL): jwr_stb with jaddr_i, jaddr_j, jplane_p[5:0], jplane_m[5:0] for couplings; bwr_stb with bias_idx, bias_code[11:0] (6 + 6 bits) for biases
- Read back: the 16-bit spin register only — SPI → PCIe → Python decodes p, q

The same silicon solves any problem — only the two 16×17 tables differ. Nothing in the chip is specific to factorization.

CIRCUIT — ppu_synapse6: ONE STORED WEIGHT BECOMES A CURRENT

Left: the real cell schematic from the deck. Right: the six doubling legs — choosing which legs are on IS the weight

BLOCK 1 — ppu_synapse6 : ONE WEIGHT BECOMES A CURRENT



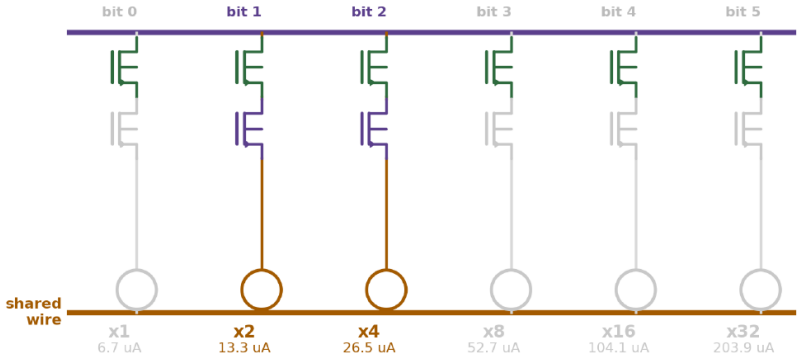
WL = spin bit x_j b0..b5 = the 6-bit weight code VBIAS sets the size of one unit

Three nch in series per leg: spin switch (1u/0.06u), weight-bit switch (1u/0.06u), current source (4u/1.0u).
Leg k is built m = 2^k times, so the six legs give 1 : 2 : 4 : 8 : 16 : 32 -> any code 0...63.

cell current = x_j x code x 6.68 uA (18 transistors, 544 cells on the chip)

three nch per leg: spin switch (WL) · weight-bit switch (b_k) · current source (VBIAS); legs built ×1 … ×32

SIX LEGS WITH DOUBLING CURRENTS = ANY WEIGHT 0...63



this example
code 6
= 4 + 2
legs 2 and 1 are on
39.9 uA
flows into the wire

Six switchable currents in the ratio 1 : 2 : 4 : 8 : 16 : 32 can make ANY whole number of units from 0 to 63.
Choosing which legs are on IS writing the weight. That is what "6-bit weight" means.

One cell = one number from the matrix, stored as six switches.

code 6 = legs 2 and 1 on → 39.9 µA into the shared wire

cell current = x_j × code × one unit — two switches in series are the AND, the wire does the sum (Kirchhoff): no multiplier, no adder on the chip

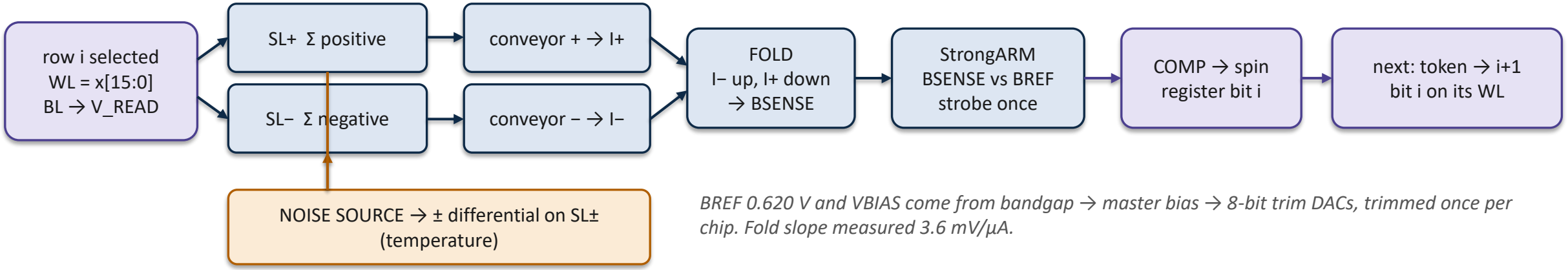
HOW A STORED WEIGHT BECOMES A CURRENT — ppu_synapse6

Block 1 of the deck; 18 transistors per cell, 544 cells

- Six legs in parallel, leg k built m = 2^k times: 1 : 2 : 4 : 8 : 16 : 32 → any code 0…63
- Each leg = three nch in series between BL (V_READ 0.499 V) and SL (VREF_SL 0.150 V):
  - spin switch — gate = WL = x_j (1u/0.06u)
  - weight-bit switch — gate = b_k from the latch (1u/0.06u)
  - current source — gate = VBIAS, sets the unit (4u/1.0u)
- Cell current = x_j × code × one unit. Unit = 6.68 µA (Version A measured), 7.42 µA post-layout
- Two switches in series = an AND — the same AND as a partial product, made of real switches. That is the entire multiply.
- All cells of a row pour into the same SL: Kirchhoff does the sum — there is no adder on the chip. Row current budget 400 µA (worst row in the example: 351 µA)
- The SL is held at 0.150 V by ppu_vg_conveyor2 (measured −3.2 … +1.1 mV from 50 to 400 µA) — otherwise the weights would compress with the answer (measured 35 % without it)

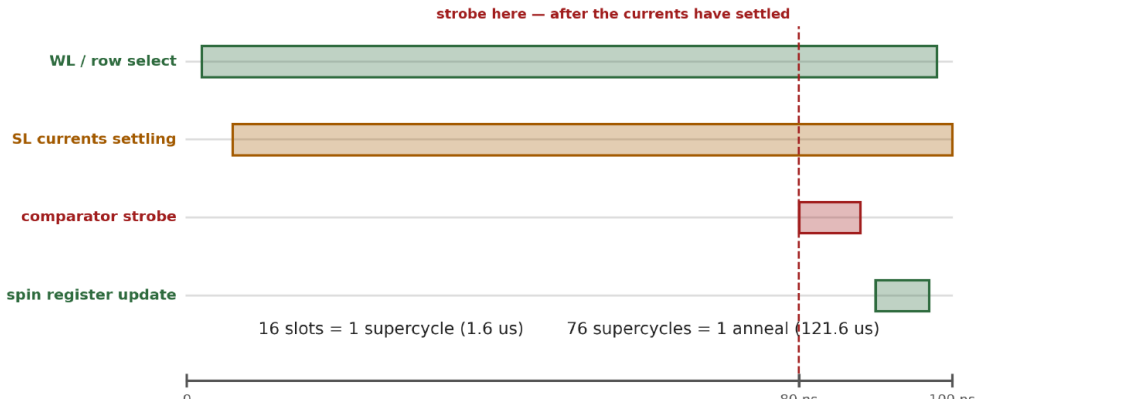
ONE SLOT · DIAGRAM — FROM ROW SELECT TO SPIN UPDATE

Top: the decision chain, block by block (drawn). Bottom: the 100 ns slot timeline as presented



BREF 0.620 V and VBIAS come from bandgap → master bias → 8-bit trim DACs, trimmed once per chip. Fold slope measured 3.6 mV/µA.

BLOCK 8 — ONE 100 ns SLOT : THE LOOP CLOSED IN TIME



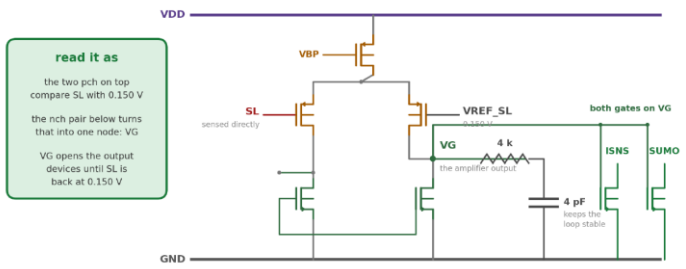
The row is opened, the two wires are given time to settle, the comparator is strobed once, and the result is latched.
Then the token moves on and the next spin is decided — with the previous answer already on its word line.
That single 100 ns loop, repeated, is the entire machine.

deck: WL/row select → SL settling → comparator strobe (+80 ns) → spin register update, in a 100 ns slot

CIRCUITS — THE THREE BLOCKS THAT TURN TWO CURRENTS INTO ONE BIT

As drawn in the deck: ppu_vg_conveyor2 (holds the wire still) · the fold (one node that rises or falls) · the StrongARM comparator

THE REAL CIRCUIT — ppu_vg_conveyor2



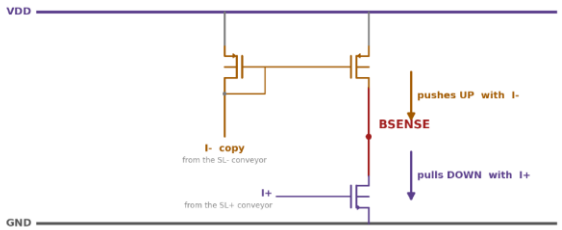
ISNS is the sink that really pulls the array current, taken BEYOND the read switch; SUMO is a copy for the next stage.
SL itself is sensed directly at the pch gate — that is what "Kelvin" means here.

Measured: SL held to -3.2 ... +1.1 mV from 50 to 400 uA, settling in 40-60 ns at the real 80 pF.

holding the shared wire still - 3 / 3

SL sensed at the pch gate (Kelvin); ISNS sinks the array current; SUMO is the copy

THE FOLD: ONE WIRE THAT RISES OR FALLS



so the voltage of that one node already IS the answer

I- bigger -> BSENSE rises I+ bigger -> BSENSE falls

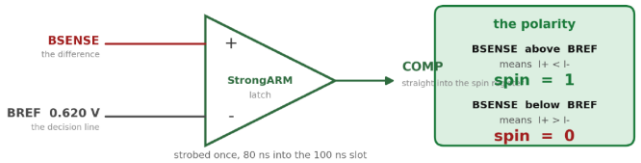
A diode-connected pch turns the I- copy into a mirrored current pushing the node up;
the I+ copy is simply an nch pulling the same node down.

No subtractor, no arithmetic — two mirrors meeting on a wire. Measured slope 3.6 mV per uA.

two currents into one bit - 2 / 3

I− copy pushes BSENSE up, I+ pulls it down — the voltage IS the answer

THE COMPARATOR: BSENSE AGAINST BREF



and that is precisely the rule the mathematics asked for:
switch the spin ON when turning it on LOWERS the score (a_i + sum_j Q_ij x_j < 0)

The latch is small and fast, and its own noise is what makes the decision probabilistic near the threshold.
BREF is trimmed to the measured balance point of the chain, so "equal currents" really means "equal".

One strobe per slot: sixteen decisions per supercycle.

two currents into one bit - 3 / 3

BSENSE above BREF ⇒ I+ < I− ⇒ spin = 1; strobed once per slot

No subtractor, no arithmetic: two mirrors meeting on a wire, then one latch. The noise rides on SL±, so it is already inside BSENSE when the latch fires.

Measured after trim: VREF_SL 0.150 V · VREAD 0.499 V · BREF 0.620 V · VBIAS 0.539 V — "these are not nice to have, they set the answer"

ONE DECISION SLOT — WHERE NOISE, WEIGHTS AND SPINS MEET

Block 8 of the deck, plus what the taped-out sequencer actually does

- ① token ring picks spin i → ② its row goes to V_READ; all 16 word lines carry the current state → ③ SL+ and SL− sum the currents → ④ the noise source adds a random differential (= temperature) → ⑤ StrongARM comparator strobes once, late in the slot (BSENSE vs BREF 0.620 V) → ⑥ result latched into the spin register, becomes a word line
- Deck: 100 ns slot, strobe at +80 ns. 16 slots = 1 supercycle (1.6 µs); 76 supercycles = 1 anneal (121.6 µs)
- Taped-out sequencer: SLOT_DIV = 12 clock cycles (raised from 8 after a post-layout strobe sweep showed marginal rows fail at 80 ns). At 50 MHz → 240 ns/slot, at 100 MHz → 120 ns/slot
- Open item: the operating clock is not written down anywhere (AMS testbenches and the RTL comment assume 50 MHz; the SDC and the gate-level power run use 100 MHz). Measured consequence at the worst corner: narrowest decision margin 8.0 mV at 240 ns vs 50.2 mV at 120 ns.
- Noise and decision are aligned in time: the strobe samples BSENSE while the noise differential is present on SL — the comparator's own latch noise adds to it near threshold

Weights: fixed. Spins: one bit rewritten per slot. Noise: a fresh random push per slot at a temperature that steps per SCHED entry.

SUMMARY — WHAT VERSION A IS, IN ONE TABLE

Item	Presentation (9 Aug)	Taped-out chip (26 Aug)	Source
Technology	TSMC 65 nm GP, standard CMOS	same — CRN65GPLUS, 9M, 2.847 × 2.051 mm	deck p1; DELIVERY.md
Pre-eFLASH test ASIC?	yes	yes — "eFLASH PDK not ready"	ANALOG_SPEC §0
Noise source	LFSR-steered constant-sum DAC, 6 legs	Thermal noise → 3-stage amp → 6-bit steering	deck p75–77; ppu_noise_therm2.scs
Noise full scale	10.2 µA (too small: needs ≥ 40)	~91 µA equiv. at decision node (static)	deck p59; sigma_bsense_dac.txt
Fresh draw per spin?	yes (LFSR redraws every slot)	yes (physical); ~23 % slot correlation at hot end	deck p77; noise_tau.txt
Temperature per spin?	no — per SCHED code	no — NCODE per SCHED entry × dwell	ppu_sequencer.v
Weights loaded	once, SPI burst, before RUN	same	deck p61; ppu_wlatch.v
Weight storage	6-bit latches, 2 planes, 16×17	same — 3 264 bits	ppu_wlatch.v
Unit current	6.68 µA	7.42 µA post-layout	deck p67; hwcal.py
Slot	100 ns, strobe +80 ns	12 cycles: 240 ns @ 50 MHz / 120 ns @ 100 MHz — clock undecided	ppu_sequencer.v; SAAT_KADANS_CELISKISI.txt

Repository: github.com/ozgurkazanci/p-ASIC — docs/spec/ANALOG_SPEC.md, TAPEOUT_OPEN_ITEMS.md; analog/sim/*.txt hold the measurements quoted above.