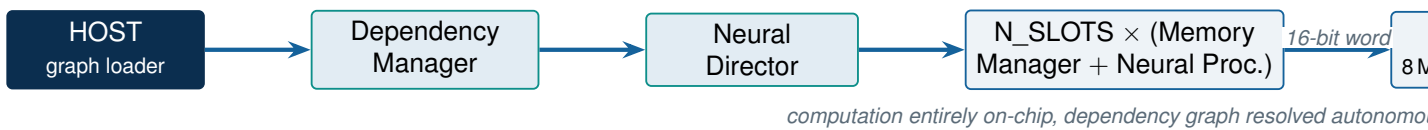


FPGA – Neural V2

Neural Multiprocessor / Dataflow Machine

N_SLOTS-way concurrent INT8 accelerator – Datasheet and reference manual

Rev. B2 • September 2026



Reference target device: Lattice ECP5 LFE5U-45F-8BG381C (speed grade –8, CABGA381) — identical device and board as V1.

Recommended configuration: INT8/INT32, P_IN=8, N_SLOTS=2 (real, measured net win — see ch. 9), same real, unmodified V1 PSRAM backend, ISSI IS66WVE4M16EBLL-70BLL.

Status: RTL verified in real Verilator simulation and real synthesis + place&route (Yosys + nextpnr-ecp5). Full benchmark campaign, two post-campaign memory optimizations, and a full alternative memory-subsystem redesign (the Neural Memory System, ch. 13) complete and measured. Document describing the project as of September 2026.

Project author: Michele Bigi • MIKILAB / manvalan.

This datasheet documents V2 of the RTL code, documentation and benchmarks present in the repository github.com/manvalan/FPGA-Neural. V1 remains frozen and unmodified as the project's golden functional/performance reference; it is documented in a separate datasheet.

FPGA-Neural V2 — General description and features

FPGA-Neural V2 is a **neural multiprocessor / dataflow machine**, the evolution of the V1 sequential accelerator (documented separately, frozen and unmodified as the project's golden reference). Where V1 executes one neuron at a time under host-driven SPI control, V2 registers a **dependency graph of neurons** and keeps `N_SLOTS` independent Neural Processor + Memory Manager pairs busy concurrently, resolving data dependencies and hiding PSRAM latency in hardware, without host intervention once a graph is loaded. Computation (INT8 MAC, ReLU, saturation) is bit-exact identical to V1's own datapath; what changed is everything *around* it.

Features

- **Dependency-graph scheduling**: nodes are registered with an explicit producer list; a node becomes eligible for execution only once every producer it depends on has genuinely completed — verified for 1-hop shared-producer/multi-consumer graphs and 2-hop transitive (diamond) graphs.
- `N_SLOTS` independent **Neural Processor + Memory Manager** pairs (default recommended: 2), each running the identical 8-stage INT8 pipeline inherited from V1.
- **Word-level burst memory backend**: fetches move a full 16-bit PSRAM word per transaction instead of one byte, reusing `memory_interface.v/psram_controller.v` directly and its already-implemented page-mode support — **2.24–2.37×** real wall-clock speedup, measured.
- **Shared on-chip activation cache**: a vector of activations shared by many neurons of the same layer is fetched from PSRAM *once*, not once per neuron — a further real **1.66–2.00×** cycle reduction on shared-input workloads.
- Same **real, unmodified V1 PSRAM backend** throughput (`memory_interface.v, psram_controller.v`) — V1 remains the frozen golden reference and was never altered to make V2 look faster.
- **Real, measured** characterization at every step: Verilator RTL simulation, Yosys synthesis, `real nextpnr-ecp5 place&route` — no theoretical number reported without a matching real measurement.

Honest, measured limitations

- The system is **memory-bound**, not compute-bound: real compute- to-memory-wait ratio on the order of 1:170–1:220. A single shared PSRAM port saturates at $\approx 90\%$ utilization regardless of `N_SLOTS` ≥ 2 — real parallel scaling beyond 2 slots is essentially flat for large workloads.
- `N_SLOTS=4` is **not recommended**: it delivers no additional real throughput once the shared PSRAM port saturates, and with the activation cache active it **fails the 80 MHz timing target outright** (65.01 MHz measured).
- Fixed, lowest-index-priority arbitration (Director and memory arbiter alike) is not fairness-balanced — a real, measured per-slot workload imbalance exists under sustained contention.

Target & toolchain

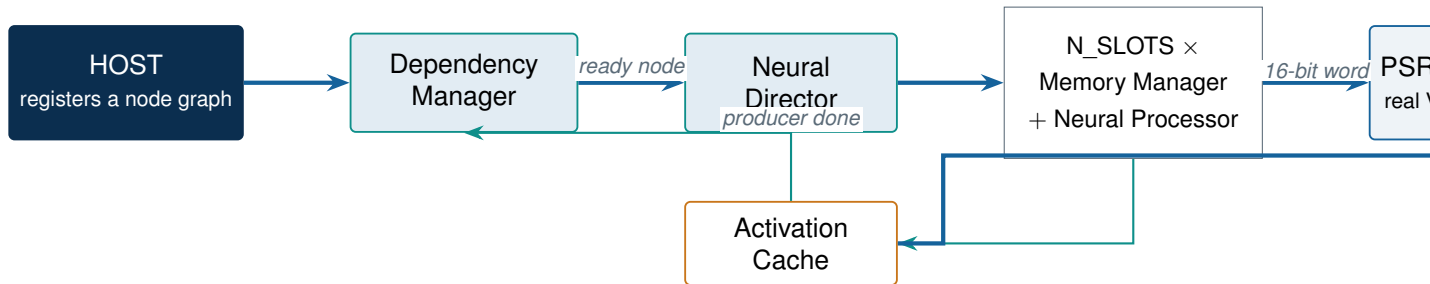
- FPGA: Lattice ECP5 LFE5U-45F-8BG381C (–8, CABGA381) — same target device as V1.
- Synthesis: Yosys; place&route: `real nextpnr-ecp5`.
- Simulation: Verilator 5.050 (`-binary -timing`) — adopted for V2 after two independent Icarus Verilog v13.0 scheduling defects were found and reproduced on minimal repros (V1's own certification, performed separately, was unaffected).
- PSRAM: ISSI IS66WVE4M16EBLL-70BLI (64 Mb, 4M $\times$ 16), real chain reused byte-for-byte from V1.

Key parameters (recommended configuration, real measured data)

Quantity	Value	Notes
Data precision	INT8 (signed)	<code>DATA_WIDTH=8</code> , identical to V1
Accumulator	INT32 (signed)	<code>ACC_WIDTH=32</code>
Dot-product width	8	<code>P_IN=8</code> parallel MAC lanes per neuron
Recommended concurrency	<code>N_SLOTS=2</code>	real, measured net win; see ch. 9
Fmax, full system (<code>N_SLOTS=2</code>)	87.72 MHz	real place&route, word-burst + activation cache active
cycles/neuron (1 neuron, 8 inputs, real PSRAM)	166 (V1: 209)	2.6× real wall-clock speedup vs V1

Combined real speedup vs baseline (N_SLOTS=2)	2.45×	word-burst + activation cache, D-Stress workload
Address space	23 bit (byte)	ADDR_WIDTH=23, unchanged from V1

System block diagram



A slot's completion feeds back to the Director (frees the slot) and to the Dependency Manager (wakes up any node waiting on it) — closing the dataflow loop entirely on-chip.

Pinout summary — scope and honesty note

V2's top-level module, `neural_multiprocessor.v`, has been synthesized and placed&routed **unconstrained** (`nextpnr-ecp5 -lpf-allow-unconstrained`) throughout this project's own real-toolchain characterization: every Fmax/resource number in this datasheet is real and measured, but **no ball-by-ball pin assignment (.lpf) has been generated for V2's top level in this revision**. Unlike V1's own pinout chapter (which reports a real, `iodb.json`-verified ball map from a constrained place&route run), this chapter reports what is **honestly known** and nothing invented.

What is real and reusable

V2's PSRAM-facing pins (`psram_a`, `psram_dq`, `psram_ce_n/oe_n/we_n/lb_n/ub_n/zz_n`) drive the exact same, real, unmodified V1 backend chain (`memory_interface.v` → `psram_controller.v`) as V1's own `spi_neuron_top`. If V2 is deployed on the same board, **V1's own real, verified ball assignment for these signals (ch. 10 of the V1 datasheet) applies unchanged** — the controller was never touched, so its pin requirements did not change either.

What is NOT yet real

The node-registration bus (`reg_valid`, `reg_node_id`, `reg_required`, `reg_producer_ids`, `reg_x_base`, `reg_w_base`, `reg_n_tiles`, `reg_result_addr`, `reg_ready`) has no assigned physical pins in this revision: every V2 measurement to date drove this bus directly from a Verilator testbench or an unconstrained synthesis top-level, never through a real host-facing SPI (or other) interface with its own placed pinout. Framing this bus as a real, deployable host interface (analogous to V1's SPI Mode 0 slave) is explicitly **future work** — see ch. 12.

Logical (not physical) port list and field widths: ch. 11 ("Register-level interface"). Real PSRAM signal reuse and board wiring: ch. 10.

Contents

1	Overview and design philosophy	1
1.1	From sequential accelerator to dataflow machine	1
1.2	What did NOT change	1
1.3	What DID change	1
1.4	The central, measured finding	1
2	Architecture	3
2.1	Module map	3
2.2	Dependency Manager	3
2.3	Neural Director	3
2.4	Memory Manager + Neural Processor (per slot)	4
2.5	Activation Cache	4
2.6	Slot Memory Arbiter	4
2.7	Real, unmodified V1 PSRAM backend	4
3	Compute datapath	5
3.1	Bit-exact reuse of V1's arithmetic	5
3.2	8-stage pipeline	5
3.3	Accumulator width: 24 vs. 32 bits	5
3.4	Activation and saturation	6
4	Parameters and configurability	7
4.1	Build parameters (synthesis-time)	7
4.2	Word-alignment constraint (post word-burst rewrite)	7
4.3	Characterized configurations	7
4.4	Build versus runtime	8
5	Memory subsystem	9
5.1	Baseline: reused byte-level V1 backend	9
5.2	Optimization #1 — word-level burst reads	9
5.3	Optimization #2 — shared activation cache	10
5.3.1	Design notes	10
5.4	Real PSRAM chain (unmodified V1)	11
5.5	SDRAM upgrade addendum (2026-09-07) — current, authoritative memory architecture	11
5.5.1	Real, measured clock closure	11
6	Dataflow scheduling	12
6.1	Node lifecycle	12
6.2	Verified graph topologies	12
6.3	First-free dispatch	12
6.4	Measured scheduling behavior	13
6.5	Correctness guarantees (measured, not assumed)	13

7	Host / graph-loader interface	14
7.1	Node registration protocol	14
7.2	Result readback	14
7.3	What a real host driver would still need to add	14
7.4	Addendum (2026-09-07) — real physical transport and completion signal, both now closed	15
8	Top-level module	16
8.1	<code>neural_multiprocessor.v</code>	16
8.2	Internal hierarchy	16
9	ECP5 implementation & benchmarks	18
9.1	Flow and verification discipline	18
9.2	V1 vs. V2 — final comparison	18
9.3	Real <code>N_SLOTS</code> sweep — Fmax and resources	19
9.3.1	Fmax versus <code>N_SLOTS</code>	19
9.4	Real parallel scaling	19
9.5	Memory optimization #1 — word-level burst reads	20
9.6	Memory optimization #2 — shared activation cache	20
9.7	Bottleneck analysis	20
9.8	Limitations, honestly stated	21
10	Hardware and board	22
10.1	Unchanged from V1	22
10.2	What V2 has not yet placed on real hardware	22
10.3	Power supply, oscillator, configuration	22
10.4	SDRAM upgrade addendum (2026-09-07) — current, authoritative board state	22
10.4.1	Memory device	23
10.4.2	Complete AS4C32M16SB-7BIN ball assignment	23
10.4.3	FPGA ↔ SDRAM mapping (real, LPF-verified)	23
10.4.4	Real, measured clock closure (nextpnr-ecp5, 8 seeds/config)	23
10.5	Power supply design (2026-09-07) — verified against the real Lattice hardware checklist	24
10.5.1	Rail topology	24
10.5.2	Power-up sequencing — real Lattice requirement, verified compliant	24
10.5.3	Decoupling — real Lattice-recommended values (not a generic “one cap per pin” guess)	24
10.5.4	Regulator component values (real, computed from datasheet constants)	25
10.5.5	Power tree	25
10.6	Programming architecture (2026-09-07) — two independent flash devices, ESP32 over JTAG only	25
10.6.1	Two physically separate flash chips	26
10.6.2	ESP32 ↔ ECP5: JTAG only	26
10.6.3	Real ball assignments (CABGA381)	26
11	Register-level interface & internal state encodings	27
11.1	Node registration fields (quick reference)	27
11.2	Dependency Manager node state (<code>node_state</code>)	27
11.3	Neural Director state (<code>dir_state</code>)	27
11.4	Memory Manager state (<code>state</code>)	27
11.5	Neural Processor state (<code>np_state</code>)	28
11.6	Slot Memory Arbiter owner encoding	28

12 Roadmap and development status	29
12.1 Milestones M1–M10	29
12.2 Post-campaign: user-requested optimizations	29
12.3 Open work items (real, not hidden)	29
13 The Neural Memory System (NMS)	31
13.1 Design goal	31
13.2 STEP1 — real bandwidth requirement study	31
13.3 STEP2 — closed-form traffic model	31
13.4 STEP3 — real bank-contention sweep	32
13.5 STEP4–7 — real candidate synthesis and selection	32
13.6 STEP8 — full integration	32
13.7 STEP9–10 — real end-to-end benchmark vs. Current V2	33
13.8 Real per-metric detail, N_SLOTS=2 (D-Stress)	34
13.9 Recommendation	34
13.10 Open work (real, not hidden)	34
A Module and file map	36
A.1 V2 RTL (hardware/v2/rtl/)	36
A.2 NMS RTL (hardware/v2/nms/rtl/, ch. 13)	36
A.3 Reused, unmodified V1 (hardware/v1/rtl/)	37
A.4 Simulation (hardware/v2/sim/)	37
A.5 Documentation and logs (hardware/v2/docs/, hardware/v2/logs/)	37

CHAPTER 1

Overview and design philosophy

1.1 From sequential accelerator to dataflow machine

V1 is, structurally, a single pipeline: one neuron computes at a time, driven by the host over SPI, one MAC group at a time, one layer at a time. It is fast for what it is (the V1 datasheet’s own “ECP5 implementation” chapter documents its real Fmax/timing-closure history), but it cannot keep more than one computational unit genuinely busy at once, and it has no notion of a dependency graph — the host sequences everything.

V2 keeps V1’s own proven INT8 datapath (bit-exact, byte-for-byte reused math) but wraps it in a fundamentally different control architecture: a **Dependency Manager** tracks a graph of neuron “jobs”, each with an explicit list of producer nodes it depends on; a **Neural Director** dispatches every node whose dependencies have resolved to whichever of `N_SLOTS` concurrent (Memory Manager + Neural Processor) pairs is free; a slot’s completion feeds back to wake up any node that was waiting on it. Once a graph is loaded, the whole system runs autonomously — no per-neuron host intervention.

1.2 What did NOT change

- The `INT8×INT8→INT32` MAC math, the balanced adder tree, ReLU/linear activation with saturation — `neural_processor.v` is a direct, bit-exact-verified port of V1’s own `neuron_parallel.v/mac8.v/mac_unit.v`.
- The real PSRAM backend: `memory_interface.v` and `psram_controller.v` are reused **byte-for-byte, unmodified** from V1 throughout every V2 milestone — including the two post-campaign optimizations (ch. 5). V1 itself, as a tree (`hardware/v1/`), is frozen and was never touched.
- The target device (Lattice ECP5 LFE5U-45F-8BG381C) and the real-toolchain-only measurement discipline: every number in this datasheet is labelled THEORETICAL, SIMULATED, POST-P&R MEASURED, or DERIVED, and no result was invented to make V2 look better than it measured (§9).

1.3 What DID change

- **Concurrency**: from one active neuron to `N_SLOTS` independent Neural Processor instances, each fed by its own Memory Manager.
- **Scheduling**: from host-sequenced SPI opcodes to an on-chip dependency graph, resolved autonomously.
- **Memory backend granularity**: from byte-at-a-time fetches (through `int8_memory_access.v`, still frozen V1 but no longer instantiated in V2’s own datapath) to word-level bursts talking to `memory_interface.v` directly — a real, measured 2.24–2.37× speedup (ch. 5).
- **Memory traffic pattern**: a new shared on-chip **activation cache** eliminates redundant re-fetching of an input vector shared by many neurons of the same layer — a further real 1.66–2.00× cycle reduction, at a real, honestly reported Fmax cost (ch. 5).

1.4 The central, measured finding

The single most important result of this project’s own benchmark campaign is that **V2 is memory-bound, not compute-bound**: the real compute-to-memory-wait ratio is on the order of 1:170–1:220, and the one physical PSRAM port saturates at ≈90% utilization regardless of `N_SLOTS` ≥ 2. Real parallel scaling from `N_SLOTS`=1 to `N_SLOTS`=8 is essentially flat for large/sustained workloads

(1.05–1.06×), and once real, place&route-measured Fmax degradation from added routing congestion is also accounted for, `N_SLOTS=4` measures as *slower* in real wall-clock time than `N_SLOTS=1` for the largest workload tested — more hardware parallelism made that specific configuration worse, not better, because the bottleneck was never compute. This finding directly shaped both post-campaign optimizations in ch. 5 and the `N_SLOTS=2` recommendation carried throughout this datasheet.

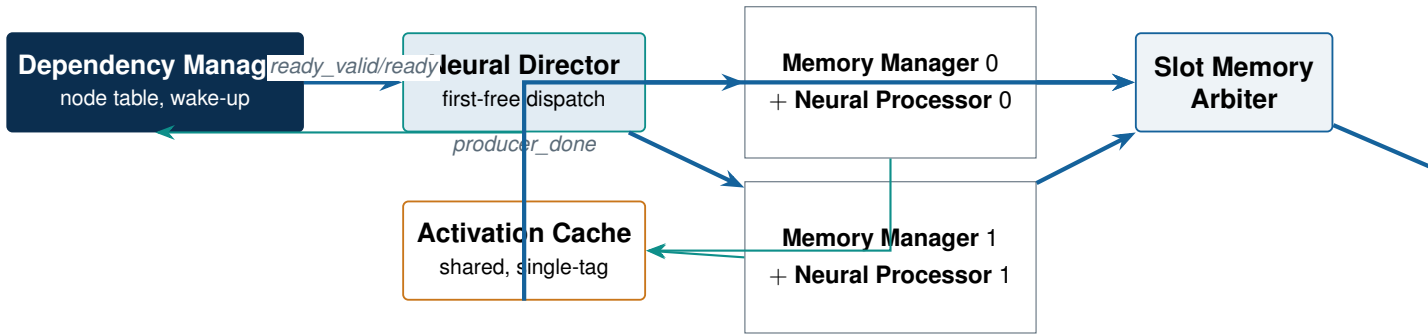
Reproducibility

Every real number in this datasheet traces to a specific, append-only log entry (`EXP-NNNN`, `DEC-NNNN`, `ERR-NNNN`) in `hardware/v2/logs/`, a specific git commit, and an exact toolchain command — the same discipline applied throughout V1’s own development.

CHAPTER 2

Architecture

2.1 Module map



N_SLOTS=2 shown (the recommended configuration); the architecture is parametric in N_SLOTS. Every arrow is a real signal path verified in Verilator simulation and real Yosys/nextpnr-ecp5 synthesis.

2.2 Dependency Manager

Holds a table of `N_NODES` job descriptors, each tracking: node id, state (`EMPTY`/`WAITING`/`READY`/`DISPATCHED`), required-dependency count, resolved-dependency count, up to `MAX_DEPS` producer node ids, and the job descriptor fields (`x_base`, `w_base`, `n_tiles`, `result_addr`). A node with zero required dependencies is immediately `READY` on registration. When a producer completes, every `WAITING` node listing it among its own producers gets its resolved-dependency count incremented — a single producer can satisfy several waiting consumers (shared-producer/multi-consumer), and a node depending on several producers accumulates resolution across separate events (multiple dependencies). Verified for both 1-hop and 2-hop transitive (diamond) graphs. Ready nodes are handed to the Neural Director one at a time over a backpressure-safe `valid/ready` interface.

No slot reclamation

`ST_DISPATCHED` is terminal: node table slots are never reused once dispatched. A long-running system that keeps registering new nodes without limit will eventually exhaust `N_NODES` — this is a real, measured consequence (a benchmark testbench hit exactly this deadlock via node-id wraparound before `N_NODES` was sized generously enough). Slot reclamation is explicitly deferred, not forgotten.

2.3 Neural Director

Dispatches ready job descriptors to whichever of `N_SLOTS` Memory Manager instances is currently free — **first-free** scheduling: a fixed, lowest-index-wins priority scan, not load-balanced. Slot-busy tracking and completion detection are always-active, independent of whatever the allocate/scan control state happens to be that cycle (the same “don’t gate a per-unit event behind one shared FSM state” principle applied throughout this design). A completed slot’s node id is tracked (`slot_node_id`) so its completion can be resolved back to a `producer_done` event for the Dependency Manager, closing the wake-up loop without any external glue logic.

Measured scheduling imbalance

Real per-slot data ($N_SLOTS=4$, a 128-neuron workload) shows slots 0/1 delivering 1008 real tiles each while slots 2/3 deliver only 16 each, despite all four slots reporting near-100% “busy” utilization — direct, measured evidence that fixed lowest-index priority does not distribute load evenly once the shared PSRAM port is the real constraint. See ch. 9.

2.4 Memory Manager + Neural Processor (per slot)

Each slot pairs one `memory_manager.v` instance with one `neural_processor.v` instance. The Memory Manager double-buffers tile fetches (compute tile N while prefetching tile $N+1$) and presents the Neural Processor with a simple “data available” interface (`operand_valid/ready`, `tile_last`) — the processor never sees PSRAM request/wait cycles directly. A tile’s activation half is requested from the shared Activation Cache; its weight half is fetched directly (weights are per-neuron, never shared, so caching them would not help). A bank is presentable to the processor only once *both* halves have arrived (`bank_ready = bank_x_ready & bank_w_ready`).

2.5 Activation Cache

A single shared instance (not one per slot) serving every Memory Manager’s activation-fetch requests. Single-tag design: one cached `x_base` at a time, filled tile-by-tile on first use, served directly from an on-chip buffer on every subsequent request for the same vector — no PSRAM access on a hit. A request for a different `x_base` invalidates the cache and restarts filling from tile 0; this is always *correct* (never serves stale data) but can thrash under interleaved, genuinely-different-`x_base` concurrent traffic — an honestly documented limitation, not exercised by this project’s own realistic dense-layer workloads (where many neurons of one layer share one input vector, dispatched together). Full detail, including the real Fmax cost this module introduces, in ch. 5.

2.6 Slot Memory Arbiter

Funnels $N_SLOTS+1$ independent backend ports (one per Memory Manager’s weight/write-back traffic, plus one for the Activation Cache’s own traffic) down to the one real physical PSRAM port. Fixed lowest-index priority, same convention as the Director. Every incoming request is latched into a per-port pending register regardless of arbiter state — a byte-level backend protocol quirk discovered by real simulation (a fire-and-forget single-cycle request pulse can arrive while the shared bus is owned by another port; a naive “grant only while live” arbiter would silently drop it) made this latch a correctness requirement, not an optimization.

2.7 Real, unmodified V1 PSRAM backend

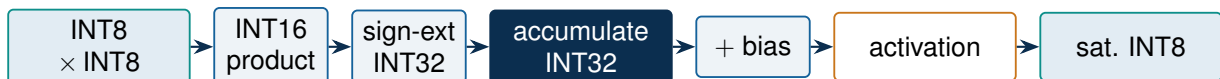
`memory_interface.v` and `psram_controller.v` are reused byte-for-byte from the frozen `hardware/v1/` tree. The controller’s own real page-mode support (fast same-page continuation vs. a slower cold access) was already implemented in V1 and is exploited more effectively by V2’s word-level burst rewrite (ch. 5) — no change to the controller itself was needed or made.

CHAPTER 3

Compute datapath

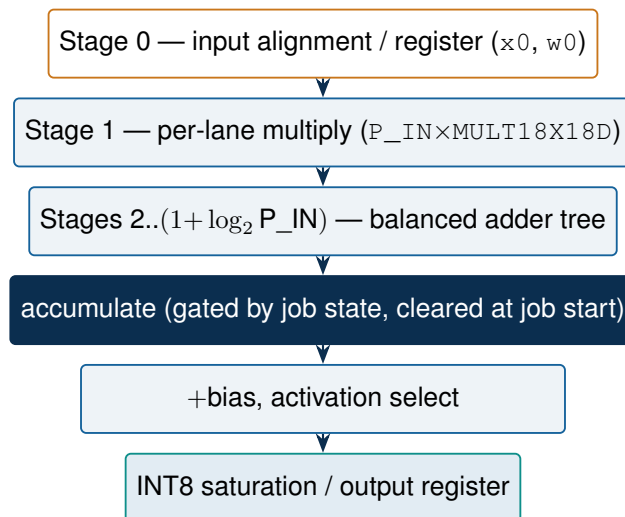
3.1 Bit-exact reuse of V1's arithmetic

`neural_processor.v` implements the identical INT8/INT32 arithmetic chain as V1's own `neuron_parallel.v/mac8.v/mac_unit.v` — verified bit-exact against V1's own modules, instantiated side-by-side in the same testbench, across 7 test cases including extreme INT8 values, back-to-back zero-gap tiles, and multi-tile jobs. What changed is the *pipelining*, not the math.



3.2 8-stage pipeline

`neural_processor.v` is fully pipelined, throughput-oriented (one new tile accepted per cycle in steady state, given a continuous operand stream):



With `P_IN=8` the adder tree has 3 levels, giving an 8-stage pipeline overall. `tile_last` is gated identically to `valid` at every stage (`last0 <= (operand_valid && operand_ready) ? tile_last : 1'b0;`) — an early draft left it ungated, letting a “last” tag propagate one cycle ahead of its own valid/data pair on jobs where `tile_last` was asserted before `operand_ready` rose (legal valid-before-ready producer behavior); found and fixed via a cycle-by-cycle dump of the pipeline's own internal valid/last signals, re-verified against the full 7-test regression.

3.3 Accumulator width: 24 vs. 32 bits

A real, 6-seed placement sweep (reusing already-synthesized netlists, real `nextpnr-ecp5` place&route only) resolved an earlier single-seed measurement that had suggested `ACC_WIDTH=32` was marginally faster:

ACC_WIDTH	mean Fmax	min	max	stdev
32	170.12 MHz	145.73	183.96	14.16

24	180.71 MHz	175.16	185.49	4.21
----	------------	--------	--------	------

Real place&route, P_IN=8, 6 seeds each (default plus 5 explicit).

Why a single seed misled

Over 6 real placement seeds, ACC_WIDTH=24 has both a higher mean Fmax (+6.2%) and a much tighter seed-to-seed spread ($\approx 3.4\times$ tighter) than ACC_WIDTH=32 — combined with fewer LUT/FF/CCU2C at 24 bits and identical bit-exact correctness, ACC_WIDTH=24 is recommended for any new P_IN=8 INT8 configuration, where product magnitudes never need more than 24 bits of accumulator headroom.

3.4 Activation and saturation

Identical encoding and bit-test logic to V1 (bilateral saturation for ACT_NONE, positive-only for ACT_RELU, both INT8-range). Every job dispatched by `dataflow_core.v` currently hardcodes `job_bias=0`, `job_activation=ACT_RELU` — a documented simplification carried through every milestone since M4/M5, not yet exposed per-node by the Dependency Manager's own job descriptor.

Encoding	Value	Behavior
ACT_NONE	2' d0	Linear: bilateral saturation to $[-128, +127]$.
ACT_RELU	2' d1	$\max(0, x)$, positive saturation to +127 (used by every V2 job today).

CHAPTER 4

Parameters and configurability

4.1 Build parameters (synthesis-time)

Parameter	Default	Meaning
DATA_WIDTH	8	Data width (INT8), unchanged from V1.
ACC_WIDTH	32	Accumulator width; 24 recommended for new P_IN=8 configurations (ch. 3).
P_IN	8	Parallel MAC lanes per neuron per tile; must be even (word-level burst constraint, ch. 5).
ADDR_WIDTH	23	Byte-address width (8 MB), unchanged from V1.
N_SLOTS	4 (RTL default)	Concurrent Memory Manager+Neural Processor pairs. 2 recommended — see the honesty note below.
N_NODES	16	Dependency Manager node-table depth. Sized to the largest node-id range a graph will ever use; never reclaimed (ch. 2).
MAX_DEPS	4	Maximum producers a single node can list.
QUEUE_DEPTH	8	Neural Director's own ready-job FIFO depth.
MAX_TILES	16 (internal, activation_cache.v)	Longest activation vector the shared cache can hold; not yet exposed as a top-level parameter.
PSRAM_DATA_WID	16	Physical PSRAM data bus width, unchanged from V1.
CLK_FREQ_MHZ	80	Frequency used in <code>psram_controller.v</code> 's own timing formulas (unmodified V1 module).

N_SLOTS is a real, measured trade-off, not a free parameter

Unlike V1's PARALLEL (a pure resource/frequency trade-off), N_SLOTS interacts with a real, measured system bottleneck (the one physical PSRAM port). N_SLOTS=1 and N_SLOTS=2 both show a real net wall-clock win over the pre-optimization baseline; N_SLOTS=4 shows *no* additional real throughput and, with the activation cache active, **fails the 80 MHz timing target outright** (ch. 9). Do not simply raise N_SLOTS for more perceived parallelism without re-running the real benchmark suite.

4.2 Word-alignment constraint (post word-burst rewrite)

Since the memory backend now moves 16-bit words rather than bytes (ch. 5), every tile base address the system computes ($x_base + tile_idx * P_IN$, and equivalently for weights) must land on an even byte address. P_IN even and x_base / w_base themselves even together guarantee this for every tile of every job — true of every address this project's own testbenches use, and a trivial constraint for any real loader/host to satisfy.

4.3 Characterized configurations

N_SLOTS	Fmax, word-burst only	Fmax, +activation cache	Notes
1	152.44 MHz	131.79 MHz	Best real wall-clock speedup ($3.86\times$ vs baseline); no arbitration contention possible.
2	133.58 MHz	87.72 MHz	Recommended default — real $2.45\times$ speedup vs baseline, still comfortably above 80 MHz.
4	112.07 MHz	65.01 MHz (FAIL)	No additional real throughput; fails 80 MHz with the cache active. Not recommended.
8	92.63 MHz (dataflow_core only, no real PSRAM chain)	not re-measured	Real DSP ceiling for P_IN=8 (64/72 MULT18X18D); a resource ceiling, not a useful operating point.

4.4 Build versus runtime



Full field-level description of the runtime (node registration) interface: [ch. 11](#).

CHAPTER 5

Memory subsystem

5.1 Baseline: reused byte-level V1 backend

V2's first working milestones connected each Memory Manager's own `prefetch_engine.v` to the real, unmodified V1 chain `int8_memory_access.v` → `memory_interface.v` → `psram_controller.v`, fetching one INT8 byte per transaction — exactly the contract V1's own `neuron_memory.v` already used against the same backend. This was correct and fully verified (bit-exact end-to-end through the real PSRAM chain), but it was not the fastest possible use of that chain.

5.2 Optimization #1 — word-level burst reads

Direct inspection of `int8_memory_access.v` shows it already converts every 8-bit logical request into a **full 16-bit PSRAM word access** internally (`mem_addr <= addr >> 1`, one byte lane selected via `lb_n/ub_n`) — so a byte-at-a-time fetch was already paying for two bytes of real PSRAM bandwidth per transaction while using only one, and paying `int8_memory_access.v`'s own request/wait round-trip twice for every real word instead of once.

`prefetch_engine.v` (weights) and `activation_cache.v` (activations, §5.3) now talk directly to `memory_interface.v`'s own 16-bit word interface, **skipping `int8_memory_access.v` entirely**. Both files remain frozen, byte-for-byte unmodified V1 — V2 simply chooses to reuse the lower (word-level) layer of the same frozen stack instead of the byte-splitting layer on top of it, the same precedent already set by `slot_mem_arbiter.v` not reusing V1's own `mem_arbiter.v` verbatim.

Real, measured result — single job, real PSRAM

n_tiles	cycles, before	cycles, after	Δ
1	166	84	−49%
3	446	204	−54%
5	728	322	−56%

Real Verilator simulation, real V1 PSRAM chain, all results still bit-exact.

Combined real wall-clock effect (256-neuron sustained workload, $\text{cycles} \div \text{real POST-P\&R Fmax}$): a **2.24–2.37×** speedup across every `N_SLOTS` tested, at a negligible real Fmax cost (unchanged at `N_SLOTS=1`; −6.2% at `N_SLOTS=2`; −1.2% at `N_SLOTS=4`).

Why not just pipeline more requests instead?

`int8_memory_access.v`'s own `STATE_IDLE` only samples a new `req` once back in `STATE_IDLE` after the previous transaction's `mem_ready` — it fundamentally does not support request pipelining. No wrapper built *on top of* it can avoid paying its round-trip cost twice per word; only bypassing it (talking to `memory_interface.v` directly) actually removes the redundancy. This is why the fix reaches one layer lower in the stack rather than adding queuing logic in front of the existing byte-level port.

5.3 Optimization #2 — shared activation cache

In the realistic dense-layer workloads this project benchmarks, many neurons of the same layer share the *exact same* activation vector. Before this optimization, each of `N_SLOTS` Memory Manager instances re-fetched that identical vector from PSRAM independently — real, measured, redundant traffic on the one shared PSRAM port. `activation_cache.v` (a new, single shared instance per `dataflow_core`, not one per slot) fetches a given `x_base` vector once, tile by tile on first use, and serves every subsequent request for the same vector directly from an on-chip buffer.

Real, measured result — 256-neuron sustained workload, D-Stress

N_SLOTS	cycles, burst only	cycles, +cache	Fmax, burst	Fmax, +cache
1	348682	174610	152.44	131.79
2	307602	185428	133.58	87.72
4	307346	184795	112.07	65.01 (FAIL)

A further real 1.66–2.00× cycle reduction on top of optimization #1, ≈4× combined vs. the original byte-level baseline.

Real, measured Fmax cost — read this before raising N_SLOTS

The shared cache's real Fmax cost is **much steeper** than optimization #1's: a single central resource with `N_SLOTS` request ports, a broadcast-capable hit-check evaluated combinatorially every cycle for every port, and a shared `tile_store` array create a genuine fan-in/routing hot spot that grows with `N_SLOTS`. `N_SLOTS=2` (recommended) still passes 80 MHz (87.72 MHz, margin down from +67% to +9.7%); `N_SLOTS=4` **fails outright** (65.01 MHz). This is the central input to ch. 12's own open work item on cache pipelining.

Combined real wall-clock speedup vs. the original byte-level baseline (both optimizations together): `N_SLOTS=1` **3.86×**; `N_SLOTS=2` **2.45×** (the recommended configuration); `N_SLOTS=4` **2.29×** but a real *regression* versus optimization #1 alone, since its own Fmax now fails 80 MHz.

5.3.1 Design notes

Single-tag, tile-granular: a request tag mismatch invalidates the cache and restarts filling from tile 0 for the new `x_base` — always correct, never serves stale data, but can thrash under interleaved, genuinely-different-`x_base` concurrent traffic (not exercised by this project's own dense-layer workloads, where sharing is real and sustained). Requests are latched per-slot on arrival (the same “queue, don't drop” idiom used by the arbiter, §2.5 of ch. 2) and served with a broadcast ack the cycle a matching tile becomes valid, so multiple slots pending on the same, about-to-arrive tile are all served the same cycle.

Two real bugs found and fixed during implementation

- (1) A target-bank/pending-bank race: a later handoff could queue a new cache request (targeting a different double-buffer bank) in the same cycle an earlier request was still awaiting its own ack, and non-blocking-assignment “last write wins” semantics silently misattributed which bank the earlier request's data landed in — the same bug class already found once for the weight-side `pf_target_bank` register, fixed with the identical two-register (pending/target) staging pattern.
- (2) A zero-width Verilog replication at `N_SLOTS=1` (`{ $clog2(1) { 1'b0 } } = { 0 { ... } }`, illegal outside a concatenation), the same class already found once in `neural_director.v` and

fixed with the same width-agnostic '0 literal. Both found via real simulation, not by inspection.

5.4 Real PSRAM chain (unmodified V1)

`memory_interface.v` and `psram_controller.v` are byte-for-byte identical to V1's own copies throughout this chapter — the real page-mode support they already implement (fast same-page continuation, slower cold access) is exploited more effectively by the word-level rewrite, not changed. The real ISSI IS66WVE4M16EBLL-70BLI chip and its board wiring are unchanged from V1 (ch. 10).

5.5 SDRAM upgrade addendum (2026-09-07) — current, authoritative memory architecture

Superseded architecture

The PSRAM-based chain described above (§§5.2–5.3) belongs to an earlier V2 milestone. The project has since closed on a single-external-memory architecture (real `decisions.log` DEC-0034): **one SDR SDRAM device, one `sdr_controller.v` instance**, serving weights, activations, AND results through `sdr_unified_backend.v`'s two logical ports (W: 64-bit weight read; AR: 16-bit, byte-maskable activation-read/result-write), arbitrated 2-way priority (W wins when both pending). No PSRAM, no second physical memory device, in the current, frozen hardware path.

The device itself was upgraded mid-project from an 8 MB part (AS4C4M16SA-6TIN) to the current **AS4C32M16SB-7BIN, 64 MB (512 Mbit), 54-ball FBGA** — both the row/column/bank geometry (`sdr_controller.v`'s ROW_BITS/COL_BITS/BANK_BITS parameters, now 13/10/2) and the SPI host protocol's own address-field width (23→26-bit byte address; WRITE_JOB payload grew 15→18 bytes) changed accordingly. Full electrical/pinout data and the complete FPGA↔SDRAM ball mapping are in ch. 10, §10.4 (kept in one place to avoid two copies of the same real data).

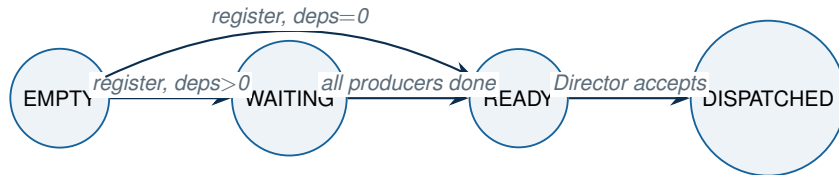
5.5.1 Real, measured clock closure

N_SLOTS=4 @ 64 MHz is the frozen production configuration: real `nextpnr-ecp5` P&R, 8/8 tested seeds PASS (worst 66.58 MHz, worst WNS +0.605 ns). **N_SLOTS=8 @ 64 MHz remains an open item:** 5/8 seeds PASS (worst 60.12 MHz, worst WNS −1.009 ns) after a real critical-path optimization (`sdr_unified_backend.v`'s weight-cache hit-index encoder, rewritten from a serially-dependent priority scan to a flat, parallel one-hot compare — real `errors.log` ERR-0029/`decisions.log` DEC-0040). 80 MHz was tested with a genuinely regenerated PLL (not merely a `-freq` flag) and is **not achievable** at either processor count (0/8 seeds pass, both before and after the ERR-0029 optimization) — the achievable Fmax is a property of the routed fabric, confirmed identical between the 64 MHz- and 80 MHz-targeted netlists. Bit-exact functional correctness (D-Stress, 256/256 neurons vs. golden model) is unaffected at every configuration tested, including through this optimization.

CHAPTER 6

Dataflow scheduling

6.1 Node lifecycle



DISPATCHED is terminal (§2): a real, honest consequence, not an oversight — see the roadmap (ch. 12) for the deferred slot-reclamation work item.

6.2 Verified graph topologies

Topology	What it proves
Shared producer, 2 consumers	One node's completion resolves the dependency count of <i>two</i> different waiting nodes independently.
Multiple producers, 1 consumer	A node with <code>required>1</code> only becomes <code>READY</code> once <i>every</i> listed producer has completed, tracked across separate wake-up events.
2-hop transitive diamond (<i>A</i> , <i>B</i> independent; <i>C</i> dep- <i>A</i> ; <i>D</i> dep- <i>B</i> ; <i>E</i> dep- <i>C</i> , <i>D</i>)	Correct cascading wake-up two hops deep — <i>E</i> does not fire until <i>C</i> and <i>D</i> have <i>themselves</i> genuinely completed, not merely been marked ready.
Mixed-depth fan-in (node depending on both a root and a 1-hop descendant)	Dependency resolution does not assume a uniform graph depth.
Multilayer (8 layer-1 neurons, random INT8 data, feeding 2 layer-2 neurons reading their real shared result bytes)	Real cross-node <i>data</i> forwarding through real PSRAM — layer-2's golden values are computed from the real bytes layer-1 actually wrote, not from an independent expectation.

All topologies above were exercised with the real, full `neural_multiprocessor.v` (real V1 PSRAM chain, real `slot_mem_arbiter.v`) and verified bit-exact against a software golden model.

6.3 First-free dispatch

The Neural Director's own scheduling policy is deliberately the simplest one that is provably correct: a fixed, lowest-index priority scan over currently-free slots. Round-robin, least-loaded, or any fairness-aware alternative was explicitly deferred until real measured data showed whether it mattered (§6.4).

6.4 Measured scheduling behavior

Real per-slot data (`N_SLOTS=4`, a 128-neuron dense-layer workload) shows a striking imbalance: slots 0 and 1 each deliver 1008 real tiles, while slots 2 and 3 deliver only 16 each — despite all four slots reporting near-100% “busy” utilization. The cause is not unfairness in isolation: once the shared PSRAM port is saturated (ch. 5), there is rarely a moment where the low-index slots are simultaneously busy *and* the high-index slots have nothing to do, so the fixed low-index-first scan keeps re-selecting the same two slots. This is a real, measured limitation of the current scheduler, carried into ch. 12 as an open item rather than patched without first measuring whether it is worth the added complexity for real workloads.

6.5 Correctness guarantees (measured, not assumed)

Across the full final benchmark campaign (6 workloads $\times$ 4 `N_SLOTS` configurations, re-verified after both memory optimizations): **zero** lost jobs, **zero** duplicated jobs (`jobs_allocated == jobs_completed == neurons_completed` exactly, every run), **zero** deadlocks, **zero** timeouts, correct multi-hop dependency wake-up in every topology tested.

CHAPTER 7

Host / graph-loader interface

Scope of this chapter

V1's own host interface is a real, placed, physically-verified SPI Mode 0 slave (ch. 7 of the V1 datasheet). V2's equivalent — a node-registration bus into `neural_multiprocessor.v` — has, in this revision, been exercised exclusively from Verilator testbenches and unconstrained synthesis top-levels. This chapter describes the **logical** protocol only; no real host-side driver (SPI or otherwise) has been built or placed yet. See ch. 12.

7.1 Node registration protocol

A simple valid/ready producer interface, backpressure-safe: the loader holds `reg_valid` and the node's own fields until `reg_ready` is observed high on the same cycle, exactly like registering into any FIFO. `reg_ready` for a given `reg_node_id` is asserted whenever that node's own table slot is `EMPTY` (§6).

Field	Width	Meaning
<code>reg_node_id</code>	$\lceil \log_2 N_NODES \rceil$	Node's own id — doubles as its table slot index.
<code>reg_required</code>	$\lceil \log_2 (MAX_DEPS + 1) \rceil$	How many of <code>reg_producer_ids</code> are meaningful (0 $\Rightarrow$ immediately READY).
<code>reg_producer_ids</code>	$MAX_DEPS \times \lceil \log_2 N_NODES \rceil$	Packed array of producer node ids this node depends on.
<code>reg_x_base</code>	<code>ADDR_WID</code>	Base byte address of this node's activation vector.
<code>reg_w_base</code>	<code>ADDR_WID</code>	Base byte address of this node's weight vector.
<code>reg_n_tiles</code>	16	Number of <code>P_IN</code> -wide tiles to accumulate.
<code>reg_result_addr</code>	<code>ADDR_WID</code>	Byte address the computed INT8 result is written to.

A node id is a real, finite resource

Because dispatched node table slots are never reclaimed (§6), a loader driving many independent jobs over a long session must use a fresh `reg_node_id` for each one, within `N_NODES`. Reusing a value before the system has been reset will simply be refused (`reg_ready` stays low for an occupied, non-`EMPTY` node id) — it will not corrupt anything, but it will also not register.

7.2 Result readback

The computed INT8 result is written to `reg_result_addr` through the same real PSRAM chain every other memory access uses — there is no separate result-readback port; the host/loader reads the result byte back from PSRAM directly, the same convention every V2 testbench in this project uses for verification.

7.3 What a real host driver would still need to add

- Per-job `bias/activation` selection, currently hardcoded to `bias=0/ACT_RELU` for every job (§3).

7.4 Addendum (2026-09-07) — real physical transport and completion signal, both now closed

Supersedes the two items removed from the list above

Both real gaps this chapter used to list are closed. This section is the current, real state.

Physical transport: `spi_host_bridge.v`, a real SPI Mode 0 slave, is the board's actual node-registration transport — real ball assignments (`spi_sclk/spi_mosi/spi_miso/spi_cs_n`) verified, real place&route (see ch. 10). `WRITE_JOB` carries the full table from §7 above as an 18-byte payload (grew from 15 after the 64MB memory upgrade widened every address field from 3 to 4 bytes — `decisions.log` DEC-0039).

Completion notification: `FPGA_DATA_READY`, a real output pin (ball G3, bank 7), closes the exact gap this chapter used to flag. It is a system-idle detector, not a per-job pulse — deliberately, since “the whole graph has an answer” and “one neuron finished” are different questions and only the former is useful to a host waiting on a result:

$$\text{sys_busy} = (\vee \text{job_active}) \vee \neg \text{queue_empty} \vee \text{any_pending}$$

where `job_active` is per-slot (already real, §6), `queue_empty` is `neural_director.v`'s own dispatch-queue occupancy, and `any_pending` (new) is an OR-reduce over `dependency_manager`'s own node table for any node still `WAITING` or `READY` (i.e. registered but not yet dispatched — `DISPATCHED` nodes are tracked by the two signals above instead, not here). `FPGA_DATA_READY` is a sticky register: set on the `sys_busy` 1 → 0 edge, cleared the instant `sys_busy` goes high again — self-clearing, no host acknowledgement command needed.

Real, disclosed assumption

This is correct only if the host finishes registering every node of a graph before the first one completes. Realistic for this architecture's own real timing (SPI registration: microseconds; per-neuron compute: ~195 real measured cycles, §9) but not proven for every conceivable host registration pattern — a host that deliberately staggers registration across a long enough gap could observe a premature `FPGA_DATA_READY` pulse after only the first node completes.

Bit-exact regression re-verified with an explicit assertion on this signal (`N_SLOTS=4` and `8`, both `PASS`, see `decisions.log` DEC-0041) and a real `nextpnr-ecp5` placement check (0 errors, `data_ready` placed at G3).

CHAPTER 8

Top-level module

8.1 neural_multiprocessor.v

The real, hardware-facing top level: `dataflow_core.v` (Dependency Manager + Neural Director + $N_SLOTS \times$ (Memory Manager + Neural Processor) + Activation Cache) with its $N_SLOTS+1$ Memory Backend Interface ports funneled through `slot_mem_arbiter.v` down to the real, unmodified V1 PSRAM chain (`memory_interface.v` $\rightarrow$ `psram_controller.v`).

Port	Dir	Width	Function
<code>clk, rst</code>	IN	1	System clock, synchronous reset.
<code>reg_valid</code>	IN	1	Node registration request (ch. 7).
<code>reg_ready</code>	OUT	1	This node id's table slot is <code>EMPTY</code> .
<code>reg_node_id</code>	IN	$\lceil \log_2 N_NC \rceil$	Node id.
<code>reg_required</code>	IN	$\lceil \log_2 (MAX_DEPS + 1) \rceil$	Producer count.
<code>reg_producer_ids</code>	IN	MAX_DEPS	Packed producer id list.
<code>reg_x_base,</code> <code>reg_w_base,</code> <code>reg_result_addr</code>	IN	$ADDR_WIDTH$ each	Job descriptor addresses.
<code>reg_n_tiles</code>	IN	16	Tile count.
<code>psram_a</code>	OUT	$ADDR_WIDTH$	PSRAM address bus (real V1 controller, unmodified).
<code>psram_dq</code>	INOUT	$PSRAM_DATA_WIDTH$	PSRAM bidirectional data bus.
<code>psram_ce_n,</code> <code>psram_oe_n,</code> <code>psram_we_n,</code> <code>psram_lb_n,</code> <code>psram_ub_n,</code> <code>psram_zz_n</code>	OUT	1 each	PSRAM control, identical to V1's own real, verified signal set.

No `int8_memory_access.v` in this datapath

Earlier milestones instantiated V1's `int8_memory_access.v` between the arbiter and `memory_interface.v`. Post word-burst rewrite (ch. 5), it is no longer instantiated here — the file itself is untouched (still frozen V1); V2 simply reuses one layer lower in the same frozen stack.

8.2 Internal hierarchy

`neural_multiprocessor.v`

- `u_dataflow_core`: `dataflow_core.v`
 - `u_dep_mgr`: `dependency_manager.v`
 - `u_director`: `neural_director.v`
 - `GEN_SLOT[0..N_SLOTS-1]`: `memory_manager.v` + `neural_processor.v`
 - `u_prefetch`: `prefetch_engine.v` (weights only, word-level)
 - `u_activation_cache`: `activation_cache.v`

-
- `u_arbiter: slot_mem_arbiter.v (N_SLOTS+1 ports)`
 - `u_memif: memory_interface.v (frozen V1)`
 - `u_psram_ctrl: psram_controller.v (frozen V1)`

CHAPTER 9

ECP5 implementation, real benchmark campaign & measured results

9.1 Flow and verification discipline

Every number in this chapter is labelled THEORETICAL, SIMULATED, POST-P&R MEASURED, or DERIVED (a combination of two real measurements, e.g. cycles ÷ real Fmax). No result is invented, approximated to look better, or reported without a matching real measurement.

Verification stage	Outcome	Covers
RTL simulation (Verilator 5.050)	PASS	bit-exact correctness vs. a software golden model
ECP5 synthesis (Yosys synth_ecp5)	PASS , 0 problems	synthesizability, resource mapping
Place&route (real nextpnr-ecp5)	PASS at $N_SLOTS \leq 2$	LUT/FF/DSP, real timing
Full benchmark campaign	24/24 bit-exact	6 workloads × 4 N_SLOTS configurations

Verilator, not Icarus, for V2

Two independent Icarus Verilog v13.0 scheduling defects were found and reproduced on minimal repros during V2's own M1 milestone (a task/scope-entry desync and a spurious condition evaluation, both edge-parity dependent) — Verilator gives correct results on the same repros. V1's own certification (performed separately, with Icarus) was unaffected, since its own testbenches already avoided the trigger pattern by convention; this is flagged honestly, not glossed over.

9.2 V1 vs. V2 — final comparison

Both systems full-system (not isolated modules), same PARALLEL/P_IN=8, same real, unmodified V1 PSRAM chain.

Metric	V1	V2 ($N_SLOTS=2$)	Class
Fmax	68.65 MHz (FAIL)	87.72 MHz (PASS)	Post-P&R
LUT (Total LUT4s)	8907	4359	Post-P&R
FF (Total DFFs)	4900	3924	Post-P&R
DSP (MULT18X18D)	16	16	Post-P&R
BRAM (DP16KD)	2	0	Post-P&R
cycles/neuron (1 neuron, 8 inputs, real PSRAM)	209	166	Simulated

Real wall-clock speedup vs. V1	1.00×	2.6×	Derived
--------------------------------	-------	-------------	---------

V1's own figures are its already-certified, frozen baseline (not re-measured this session); V2's figures are real, current measurements including both post-campaign optimizations.

Where the win comes from — and where it does not

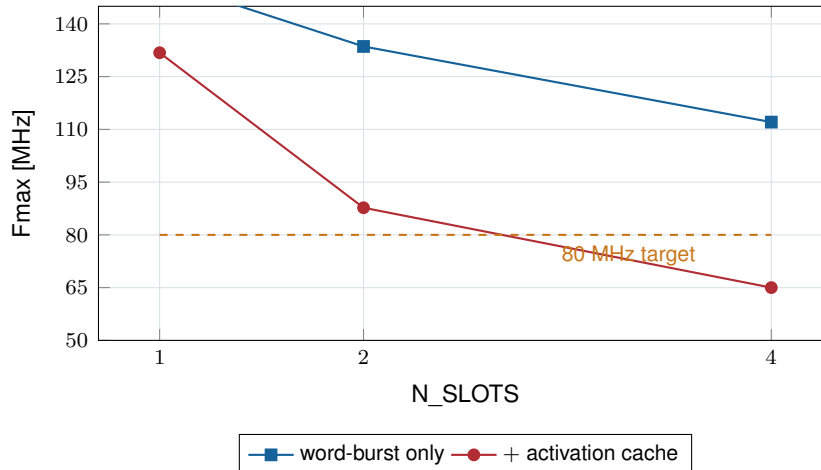
V2's advantage comes from a faster pipeline and a higher achievable clock, **not** primarily from the multi-processor concurrency the architecture was built to add. That concurrency's own real payoff, given the single-PSRAM-port memory subsystem, is much smaller than a naive $N_SLOTS \times P_IN$ calculation would suggest — §9.4.

9.3 Real N_SLOTS sweep — Fmax and resources

Full system, real place&route, both memory optimizations active (word-burst §9.5 + activation cache §9.6).

N_SLOTS	Fmax	LUT4	FF	DSP	BRAM	80 MHz
1	131.79 MHz	2760	2405	8/72	0	PASS
2	87.72 MHz	4359	3924	16/72	0	PASS(recommended)
4	65.01 MHz	9158	7986	32/72	0	FAIL

9.3.1 Fmax versus N_SLOTS



The activation cache's own real Fmax cost grows much faster with N_SLOTS than the arbiter-widening cost alone — a single shared resource with N_SLOTS request ports and an unpipelined, broadcast-capable hit-check.

9.4 Real parallel scaling

Not assumed — computed from real cycle counts, largest workload (256 independent neurons sharing one input vector).

N_SLOTS	Speedup(N)	Efficiency	PSRAM utilization	Real wall-clock speedup vs. N=1
1	1.00×	100%	55.5–71.8%	1.00×
2	1.06×	53%	≈90%	0.99× (a wash)
4	1.06×	27%	≈90%	0.79× (slower)
8	1.06×	13%	≈90%	—

Pre-optimization figures, isolating the real scaling behavior from the two memory optimizations' own effect (§9.5–9.6).

The central, measured finding

Real cycle-count speedup from N_SLOTS=1 to N_SLOTS=8 is essentially flat (1.05–1.06×) for sustained, memory-bound workloads — the single shared PSRAM port saturates at ≈90% utilization regardless of N_SLOTS≥2. Once real Fmax degradation is also folded in, N_SLOTS=4 measures *slower* in real wall-clock time than N_SLOTS=1. More hardware parallelism made this workload class worse, not better, because the bottleneck was never compute.

9.5 Memory optimization #1 — word-level burst reads

See ch. 5, §5.2, for the full rationale. Real, measured single-job cycle reduction: –49% (1 tile), –54% (3 tiles), –56% (5 tiles). Combined real wall-clock speedup on the 256-neuron sustained workload: 2.24–2.37× across every N_SLOTS tested, at negligible real Fmax cost.

9.6 Memory optimization #2 — shared activation cache

See ch. 5, §5.3. A further real 1.66–2.00× cycle reduction on top of optimization #1, at a real, steep Fmax cost that makes N_SLOTS=4 fail 80 MHz outright.

N_SLOTS	Wall-clock, baseline	Wall-clock, final	Total real speedup
1	5118.1 μs	1324.9 μs	3.86×
2	5169.5 μs	2113.9 μs	2.45× (recommended)
4	6498.7 μs	2842.6 μs (Fmax fails)	2.29× but a real regression vs. #1 alone

9.7 Bottleneck analysis

Candidate	Verdict, with real evidence
Memory (PSRAM port)	The real bottleneck. ≈90% utilization at N_SLOTS≥2; real compute-to-memory-wait ratio on the order of 1:170–1:220.

Compute (Neural Processor)	Not the bottleneck — the pipeline is idle most of the time waiting for data.
Arbiter overhead	Real but small: a mandatory 1-cycle pending-latch (correctness, not choice) plus modest Fmax cost (−6% at N_SLOTS=2, word-burst alone).
Director/depender logic	Not the bottleneck — zero lost/duplicated jobs, no queueing backlog observed; a real <i>fairness</i> issue exists (§6) but does not limit throughput.
DSP/LUT/FF availability	Not the bottleneck at N_SLOTS≤4 — all well under budget; DSP would eventually bind at N_SLOTS=9 (64/72), never reached in practice since the memory bottleneck dominates first.

9.8 Limitations, honestly stated

- V1’s own memory-utilization/stall figures were not re-measured this session (V1 is frozen); only its already-certified numbers are used for comparison.
- No clean per-cycle split between “processor computing” and “processor waiting for memory” exists in the current instrumentation — reported figures use tile-delivery-rate proxies, not an exact split.
- Power/energy: **NOT MEASURED** — no ECP5 power estimator (`ecppower`, `icepower`, or equivalent) is available in this project’s toolchain; no value is invented in its place.
- N_SLOTS=8 was not re-measured full-system (real PSRAM chain) after either memory optimization — only `dataflow_core.v` alone, pre-optimization (92.63 MHz).

CHAPTER 10

Hardware and board

10.1 Unchanged from V1

V2 targets the identical board and component set as V1: Lattice ECP5 LFE5U-45F-8BG381C (–8, CABGA381), ISSI IS66WVE4M16EBLL-70BLI PSRAM (64 Mb, 4M×16), same 16 MHz reference oscillator. The real PSRAM controller (`psram_controller.v`) and its byte↔word adapter (`memory_interface.v`) are reused byte-for-byte, unmodified, from `hardware/v1/` throughout every V2 milestone — their real, already-verified electrical/timing requirements and page-mode behavior are unchanged, because the controller itself was never touched.

Real ball assignment: defer to V1's own chapter

V1's own hardware chapter documents a real, `iodb.json`-verified, place&route-confirmed ball assignment for every PSRAM signal (`psram_a`, `psram_dq`, `psram_ce_n/oe_n/we_n/lb_n/ub_n/zz_n`). Since V2's own `neural_multiprocessor.v` drives these signals through the identical, unmodified controller, that same real ball assignment applies unchanged if V2 is deployed on the same physical board — it is not repeated here to avoid maintaining two copies of the same real data; see the V1 datasheet directly.

10.2 What V2 has not yet placed on real hardware

As stated in ch. 7, V2's own node-registration bus has no physical pin assignment in this revision — every V2 characterization to date used either a Verilator testbench or an unconstrained (`-lpf-allow-unconstrained`) synthesis top-level. A real deployment would need:

- A physical host transport for the registration bus (ch. 7).
- A real, constrained `nextpnr-ecp5` place&route run producing a genuine `.lpf/ball` assignment for `neural_multiprocessor.v`'s own top-level pins, analogous to V1's own `tools/pinout/gen_lpf.py` flow.
- Re-verification that the real Fmax numbers in ch. 9 (obtained unconstrained) hold once real pin locations are fixed — pin placement can itself affect routing and therefore Fmax.

10.3 Power supply, oscillator, configuration

Unchanged from V1: same board-level power sequencing, same oscillator, same JTAG/config-SPI boot path (fixed-function dedicated pins, outside RTL scope). No V2-specific hardware change was made or is required beyond the (not yet placed) registration-bus transport above.

10.4 SDRAM upgrade addendum (2026-09-07) — current, authoritative board state

This section supersedes the PSRAM description above for the current hardware baseline

The sections above describe an earlier V2 milestone that still reused V1's own PSRAM chain unconstrained. The project has since made a closed architectural decision (real `decisions.log` DEC-0034) to replace external memory with a single SDR SDRAM device, and has since upgraded that device's capacity and re-verified real, constrained place&route timing. This section is the current, real, measured state — see `hardware/v2/docs/MEMORY_UPGRADE_64MB_N8.md` in the repository for the full investigation.

10.4.1 Memory device

Alliance Memory AS4C32M16SB-7BIN — 512 Mbit (64 MByte) SDR SDRAM, organized 4 banks $\times$ 8M words $\times$ 16 bits, 54-ball FBGA package ($8 \times 8 \times 1.2$ mm max), -40 to 85°C industrial, -7 speed grade (143 MHz max). VDD/VDDQ 3.3 V $\pm$ 0.3 V. Single-ended CLK — **no CLK_N**, this is SDR, not DDR, SDRAM. Real distributor availability confirmed: DigiKey product 11613071, 568 units in stock, \$31.12/unit (qty 1), 16-week manufacturer lead time.

10.4.2 Complete AS4C32M16SB-7BIN ball assignment

From the manufacturer's own -7BIN -specific datasheet (Alliance Memory, Rev. 1.4, June 2024, Figure 1.1 — the real TFBGA ball diagram, not inferred from the TSOP-II -7TIN pinout).

Address / Bank

A0=H7, A1=H8, A2=J8, A3=J7, A4=J3, A5=J2, A6=H3, A7=H2, A8=H1, A9=G3, A10/AP=H9, A11=G2, A12=G1, BA0=G7, BA1=G8.

Data / Masks

DQ0=A8, DQ1=B9, DQ2=B8, DQ3=C9, DQ4=C8, DQ5=D9, DQ6=D8, DQ7=E9, DQ8=E1, DQ9=D2, DQ10=D1, DQ11=C2, DQ12=C1, DQ13=B2, DQ14=B1, DQ15=A2, LDQM=E8, UDQM=F1.

Control / Power

CLK=F2, CKE=F3, CS#=G9, RAS#=F8, CAS#=F7, WE#=F9. VDD={A9,E7,J9}, VSS={A1,E3,J1}, VDDQ={A7,B3,C7,D3}, VSSQ={A3,B7,C3,D7}, NC=E2.

10.4.3 FPGA ↔ SDRAM mapping (real, LPF-verified)

From hardware/v2/constraints/v2_board_top.lpf (45/45 unique FPGA balls, no duplicates, LFE5U-45F-8BG381 rev. 3.0 CSV-verified).

FPGA ball → SDRAM ball, by signal group

sdram_a[0..12]: D5,D3,F4,E5,E3,F5,A2,B1,C2,C1,D2,D1,F1 → A0..A12 (H7,H8,J8,J7,J3,J2,H3,H2,H1,G3,H9,G2,G1). sdram_ba[0:1]: E4,C3 → BA0,BA1 (G7,G8). sdram_dq[0..15]: E1,G5,H3,J5,K3,K2,H1,J1,K1,K4,L4,L5,M5,M4,N4,N5 → DQ0..DQ15. sdram_dqm[0:1]: P5,N3 → LDQM,UDQM. Control: sdram_cke/cs_n/ras_n/cas_n/we_n: B5,C5,C4,A3,B3 → CKE,CS#,RAS#,CAS#,WE#.

10.4.4 Real, measured clock closure (nextpnr-ecp5, 8 seeds/config)

Configuration	Pass	Worst Fmax	Notes
N_SLOTS=4 @ 64 MHz	8/8	66.58 MHz	Production baseline, GO
N_SLOTS=8 @ 64 MHz	5/8	60.12 MHz	Open, not production-frozen
N_SLOTS=4/8 @ 80 MHz	0/8	—	NO-GO, genuine ecpp11-regenerated PLL

Real, measured after the ERR-0029 weight-cache hit-index optimization (serial priority scan → flat one-hot compare); see hardware/v2/logs/errors.log and decisions.log DEC-0040.

10.5 Power supply design (2026-09-07) — verified against the real Lattice hardware checklist

Supersedes the generic §3 stub above

The “Power supply, oscillator, configuration” section earlier in this chapter only said “unchanged from V1” without real design data. This section replaces that stub with the actual rail topology, sized against the real, primary-source Lattice and TI documents below — not estimated.

10.5.1 Rail topology

Three rails, one simplification from the original V1 reference design: **no separate buck regulator for the 3.3 V I/O rail** — the board’s own external input is specified as **3.3 V**, so V_{CCIO} , the SDRAM (VDD/VDDQ, 3.3 V per its own datasheet), and the flash (3.3 V) are fed directly from the board input. A buck targeting 3.3 V output from a 3.3 V input would run at 100% duty cycle permanently — zero regulation margin, no benefit over a direct connection.

Rail	Value	Source	Feeds
I/O	3.3 V	Direct board input	FPGA $V_{CCIO0-8}$, SDRAM VDD/VDDQ, SPI flash, PMOD
Core	1.1 V	TLV62568 (buck), from the 3.3 V rail	FPGA V_{CC}
Auxiliary	2.5 V	TLV73325 (LDO), from the 3.3 V rail	FPGA V_{CCAUX}

10.5.2 Power-up sequencing — real Lattice requirement, verified compliant

Per Lattice’s own *ECP5 and ECP5-5G Hardware Checklist* (FPGA-TN-02038-2.0, July 2024), §4: “ V_{CCIO} supplies should be powered up before or together with the V_{CC} and V_{CCAUX} supplies.” The same document’s §2 adds: all three monitored rails must rise **monotonically**, and the on-chip Power-On-Reset de-asserts only once $V_{CC} \geq 0.9$ V, $V_{CCAUX} \geq 2.0$ V, and $V_{CCIO8} \geq 0.95$ V are all simultaneously satisfied — device initialization waits for whichever of the three is slowest.

This board’s topology satisfies the requirement **by construction**, with no sequencer IC needed: V_{CCIO} (3.3 V) is a direct, unregulated connection to the board input, so it rises first/fastest, strictly before the two regulated rails (Core, Aux) can even begin their own soft-start ramps — “before or together with” is met on every possible power-up transient, not just the typical case.

10.5.3 Decoupling — real Lattice-recommended values (not a generic “one cap per pin” guess)

Per FPGA-TN-02038-2.0 Table 3.1 (§3.1), applied per-rail:

Rail	Filter	Notes
V_{CC}	$10\ \mu\text{F} \times 3$ (bulk) + 100 nF per pin	Core, 1.1 V
V_{CCAUX}	120 Ω ferrite bead + 10 μF + 100 nF per pin	2.5 V; new part not in the earlier power tree draft — a ferrite bead in series was missing before this verification pass
$V_{CCIO}[0-8]$	10 μF + 100 nF per pin (per bank in use)	1 μF acceptable on unused banks; 22 μF (or a second 10 μF) on banks with heavy output loading

Capacitor selection, also per the same document: X5R/X7R dielectric (avoid Y5V/Z5U), voltage rating $\geq 80\%$ above the rail's maximum — for the 3.3 V rail this means a **6.3 V minimum** rating, not the bare 3.3 V-rated parts sometimes used to save cost. All ground pins tie to the board's ground plane (no star grounding on this family).

10.5.4 Regulator component values (real, computed from datasheet constants)

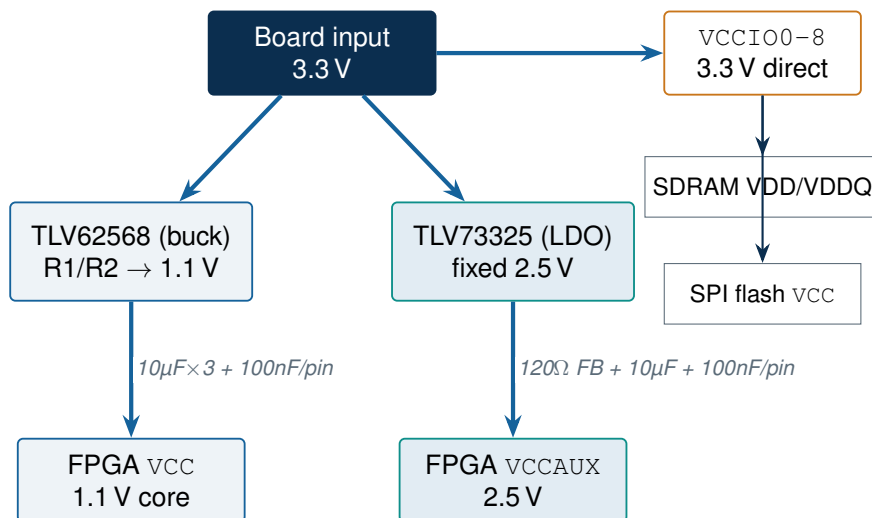
TLV62568 (core, 1.1 V): input range 2.5–5.5 V (3.3 V input has full margin); feedback reference $V_{FB} = 0.6$ V (typical, per TI SLVSD89B). Output set via $V_{OUT} = V_{FB} (1 + \frac{R1}{R2})$: choosing **R1=100 k Ω** , **R2=120 k Ω** gives $0.6 \times (1 + 100/120) = 1.1$ V exactly. Per TI's own typical application circuit: C1=4.7 μ F on V_{IN} , L1=2.2 μ H inductor, C2=10 μ F on V_{OUT} .

TLV73325 (auxiliary, 2.5 V fixed-output LDO): input range 1.4–5.5 V (per TI SBVS221, real datasheet), dropout 125 mV at 300 mA — far above this rail's ~ 10 mA real load, so dropout is not a concern at 3.3 V input. Capacitor-free architecture (stable without external caps at the regulator itself); the 10 μ F + 100 nF on V_{CCAUX} above are the FPGA-side filter from FPGA-TN-02038, not regulator-stability caps, and are still required.

Open item carried from §3 above

The 16 MHz reference oscillator's exact manufacturer part number is not yet specified in this document (only "16 MHz" as a frequency requirement) — flagged, not invented, pending the schematic capture the user is preparing separately.

10.5.5 Power tree



Power tree, direct 3.3 V I/O rail (no redundant buck), verified against FPGA-TN-02038-2.0 §3–4. Full schematic capture (BOM, connectors, FPGA–RAM/FLASH and PROG sections) pending separately.

10.6 Programming architecture (2026-09-07) — two independent flash devices, ESP32 over JTAG only

Real, closed design – not a placeholder

Converged after evaluating and rejecting a single-shared-flash and an SSPI-every-boot alternative (see `decisions.log` DEC-0041 for the full comparison). This is the current, real architecture.

10.6.1 Two physically separate flash chips

- **Flash #1** — neural-network weights/graph data. V1's own real subsystem (`flash_copy_engine.v/ flash_slot_manager.v`), 4 ordinary GPIO balls. V1's own real balls (`flash_sclk=E3`, `flash_mosi=D3`, `flash_miso=D5`, `flash_cs_n=E4`) are **not** reusable in V2 — confirmed conflict, all four already carry V2's own SDRAM bus. New balls reserved (bank 7, 3.3 V): `flash_sclk=B2`, `flash_mosi=E2`, `flash_miso=F2`, `flash_cs_n=F3`. **Not yet in the LPF** — the RTL port does not exist in `fpga_neural_v2_top.v` yet (a real, separate, open integration task).
- **Flash #2** — boot bitstream only. Connects exclusively to the ECP5's own dedicated sysCONFIG pins, Master SPI mode, auto-boots every power-up, zero ESP32 involvement in normal operation.

10.6.2 ESP32 ↔ ECP5: JTAG only

Neither ESP32-S3 nor ESP32-C6 has a hardware JTAG *master* peripheral (verified against Espressif's own documentation): their native "USB Serial/JTAG Controller" lets an external host debug the ESP32 itself — the wrong direction for driving the ECP5. TCK/TMS/ TDI/TDO are therefore bit-banged from ordinary ESP32 GPIO, standard practice. ESP32 updates flash #2 by commanding the ECP5's own internal sysCONFIG engine to bridge JTAG writes through to the external flash (real Lattice mechanism, FPGA-TN-02038-2.0 Figure 6.3, "Programming external Flash via JTAG") — ESP32 never drives flash #2's own SPI pins directly, zero bus contention by construction.

10.6.3 Real ball assignments (CABGA381)

From the official Lattice pinout CSV (`FPGA-SC-02034-3-0- ECP5U-45-Pinout.csv rev.3.0`) cross-checked against Project Trellis's `iodb.json`.

JTAG (bank 40/TAP) — to ESP32

TCK=T5, TMS=U5, TDI=R5, TDO=V4.

Dedicated config (bank 8) — to ESP32

PROGRAMN=W3, INITN=V3, DONE=Y3.

CFG[2:0] (bank 8) — board jumpers/0Ω, NOT to ESP32

For MSPI, `CFG[2:0]=[0,1,0]` read MSB-first: `CFG_2(R4)=GND`, `CFG_1(T4)=pull-up 1–10 kΩ to VCCIO8`, `CFG_0(U4)=GND`.

MSPI dedicated/dual-function pins to flash #2 (bank 8) — NOT to ESP32

MCLK/CCLK=U3, CSSPIN=R2 (dual w/ HOLDN/DI/BUSY/CEN), D0/MOSI=W2, D1/MISO=V2.

Confirmed real and safe (Lattice FPGA-TN-02039-2.3 sysCONFIG User Guide, §6.1.2): once User Mode is reached, the MSPI dedicated pins tristate with a weak pull-up, so they never contend with another driver on the same net — not load-bearing for this specific two-chip architecture (flash #1/#2 are physically separate), but confirms the mechanism is real should a future revision ever share a single chip.

CHAPTER 11

Register-level interface & internal state encodings

No SPI register map in this revision

V1's own quick-reference chapter documents a real SPI opcode/register map (`STATUS`, `SET_BASE`, `READ_CONFIG`, ...). V2 has no equivalent yet (ch. 7) — this chapter instead documents the **node-registration field layout** (repeated here for quick reference) and the **internal FSM state encodings** exposed by each module, useful for simulation-level debug and for a future host driver.

11.1 Node registration fields (quick reference)

See ch. 7 for the full field-level description. `reg_node_id`, `reg_required`, `reg_producer_ids`, `reg_x_base`, `reg_w_base`, `reg_n_tiles`, `reg_result_addr` — valid/ready handshake, `reg_ready` gated on the target node id's table slot being `EMPTY`.

11.2 Dependency Manager node state (`node_state`)

Value	Name	Meaning
2'd0	ST_EMPTY	Table slot free; <code>reg_ready</code> asserted for this node id.
2'd1	ST_WAITING	Registered, at least one producer not yet resolved.
2'd2	ST_READY	All producers resolved; eligible for dispatch.
2'd3	ST_DISPATCHED	Handed to the Director; terminal (§6).

11.3 Neural Director state (`dir_state`)

Value	Name	Meaning
4'd0	DIR_IDLE	Reset/startup.
4'd1	DIR_SCAN_READY	Checking whether a queued job and a free slot both exist.
4'd2	DIR_ALLOCATE	Dispatching the head-of-queue job to the first free slot.
4'd3	DIR_ERROR	Recoverable only via reset (an isolated fault never blocks other slots).

11.4 Memory Manager state (`state`)

Value	Name	Meaning
3'd0	MM_IDLE	Waiting for <code>job_start</code> .
3'd1	MM_PREFETCH_FIFO	Waiting for tile 0's activation <i>and</i> weight halves to both arrive.

3'd2	MM_STREAM	Presenting tiles to the Neural Processor, double-buffering the next one.
3'd3	MM_WAIT_RESULT	Last tile handed off; waiting for the Neural Processor's own result.
3'd4	MM_WRITE_RESULT	Issuing the real PSRAM word write for the INT8 result.
3'd5	MM_DONE	Waiting for the write's own <code>mem_ready</code> ; then pulses <code>job_done</code> .

11.5 Neural Processor state (`np_state`)

Value	Name	Meaning
4'd0	NP_IDLE	No job in flight.
4'd1	NP_LOAD_JOB	Latching <code>job_bias/job_activation</code> , clearing the accumulator.
4'd2	NP_WAIT_OPERAND	Consuming tiles as they arrive (absorbs the per-tile MAC/accumulate/next-tile sequence).
4'd3	NP_FINISH	Draining the pipeline after <code>tile_last</code> .
4'd4	NP_WRITE_RESULT	Result available for the Memory Manager to consume.
4'd5	NP_DONE	Job complete.
4'd6	NP_ERROR	Reachable only via an unreachable <code>default</code> case — isolated per-processor, never blocks other slots.

11.6 Slot Memory Arbiter owner encoding

`owner` is 0 for “no port granted”, or (`port index + 1`) for the currently-granted port — indices $0..N_SLOTS-1$ are the per-slot Memory Managers' own weight/write-back traffic; index `N_SLOTS` is the shared Activation Cache's own traffic.

CHAPTER 12

Roadmap and development status

12.1 Milestones M1–M10

M	Title	Status	Content
1	Neural Processor	OK	Bit-exact 8-stage pipeline vs. V1, 7/7 tests; 183.12 MHz isolated.
2	Processor Array	OK	1/2/4/8 processors, real concurrent-slot simulation; DSP (not LUT/FF) found to saturate first.
3	Buffers	OK	activation_buffer/weight_buffer/result_buffer, real DP16KD inference — superseded in the real datapath by the Activation Cache (§12.2).
4	Memory Manager	OK	Double-buffered prefetch, real V1 PSRAM chain, 3 real RTL bugs found/fixed.
5	Neural Director	OK	First-free dispatch, real backpressure, 4/4 tests.
6	Dependency Manager	OK	Multi-dependency/shared-producer wake-up, 4/4 tests.
7	Dataflow Core	OK	Full M1–M6 integration, wake-up loop closed end-to-end.
8	PSRAM integration	OK	Real, shared PSRAM across concurrent slots; 1 real arbiter bug found/fixed (dropped request under contention).
9	Full benchmark	OK	V1 vs. V2 comparison, every number classified.
10	Optimization	OK	N_SLOTS ceiling (DSP), ACC_WIDTH 6-seed sweep, real stall/utilization instrumentation.

12.2 Post-campaign: user-requested optimizations

Following M9/M10's own final benchmark campaign ([hardware/v2/docs/benchmarks/final-benchmark](#)), two concrete optimizations were implemented and measured against the real toolchain:

1. **Word-level burst reads** (ch. 5, §5.2): real $2.24\text{--}2.37\times$ wall-clock speedup, negligible Fmax cost.
2. **Shared activation cache** (ch. 5, §5.3): a further real $1.66\text{--}2.00\times$ cycle reduction, at a real, steep Fmax cost that makes $N_SLOTS=4$ fail 80 MHz outright.

Combined: $2.45\times$ real wall-clock speedup at $N_SLOTS=2$ (recommended) over the pre-optimization baseline, which was itself already $2.6\times$ faster than V1.

12.3 Open work items (real, not hidden)

Item	Why it is open
------	----------------

Activation cache pipelining	The concrete fix for <code>N_SLOTS=4</code> 's Fmax failure: register the hit-detection/broadcast logic to break its single-cycle combinational path. Not attempted this round — <code>N_SLOTS=4</code> delivers no real throughput benefit anyway (memory-bound), so this protects <code>N_SLOTS=2</code> 's own margin rather than making 4 useful.
Dependency Manager node-slot reclamation	<code>ST_DISPATCHED</code> is terminal; a real long-running system will eventually exhaust <code>N_NODES</code> .
Scheduler fairness	Fixed lowest-index priority shows real, measured per-slot imbalance under sustained contention (ch. 6); no fairness-aware alternative has been measured yet.
Second physical PSRAM bank	The only real way to raise the memory-bandwidth ceiling itself, rather than use existing bandwidth more efficiently — a board-level change, not attempted this round.
Real host driver & pinout	No physical transport or placed pin assignment exists for the node-registration bus (ch. 7, ch. 10).
Per-node bias/activation	Every job currently hardcodes <code>bias=0/ACT_RELU</code> ; not yet exposed by the Dependency Manager's own job descriptor.
Power/energy characterization	No ECP5 power estimator available in this toolchain; honestly reported as NOT MEASURED, not invented.

Every claim in this datasheet traces to a log entry

hardware/v2/logs/: development.log, simulation.log, synthesis.log, timing.log, benchmark.log, decisions.log (DEC-NNNN), experiments.log (EXP-NNNN), errors.log (ERR-NNNN). IDs are never reused, past results are never overwritten, even failed ones — the same discipline V1's own docs/validation/ campaign followed.

CHAPTER 13

The Neural Memory System (NMS)

Scope of this chapter

Chapters 2–9 document **Current V2** (`memory_manager.v` + `activation_cache.v`, DEC-0015/ DEC-0016) as a complete, frozen, real-measured system in its own right. This chapter documents a **parallel, later evolution** — the Neural Memory System (NMS) — built to directly address Current V2’s own central finding (§5.3’s own honest warning: real parallel scaling flat beyond `N_SLOTS=2`, a single shared PSRAM port saturating regardless of on-chip organization). Both systems are real, both are independently synthesizable and simulatable, and both remain available: **Current V2 is not being retired by this chapter** — §13.7’s own real data shows the choice between them is configuration-dependent, not a strict win for either.

13.1 Design goal

Current V2’s own memory path is fundamentally an on-demand, per-request architecture: every tile fetch is a fresh transaction, arbitrated one at a time onto the shared PSRAM port, with the activation cache’s own single shared instance introducing exactly the kind of centralized combinational hit-check that §5.3 already flagged as a real Fmax risk at higher `N_SLOTS`. The NMS instead asks: *what is the minimum on-chip organization that lets the Neural Processor array run at close to its own compute rate, treating PSRAM purely as backing storage?* Following the project’s own established discipline, this was answered with real, measured data at every step (a real bandwidth-requirement study, a real bank-contention sweep, real candidate synthesis) rather than assumed.

13.2 STEP1 — real bandwidth requirement study

An idealized backing-store model (runtime-configurable latency and bandwidth, simulation-only, never synthesized) drove the real, unmodified `neural_processor.v` directly, sweeping `N_SLOTS` × `PREFETCH_DEPTH` × latency × bandwidth (768 real Verilator data points). Three real bugs in the study harness itself were found and fixed first (a registered-grant race, a single-transfer-at-a-time serialization cap, and a stale-value issuance throttle) before any result was trusted.

Real result: a hard, linear bandwidth floor

Minimum aggregate bandwidth for ≥90/95/99% of compute-only throughput scales **exactly linearly** with `N_SLOTS` at **16 bytes/cycle/slot** ($= 2 \times P_IN$, the raw activation+weight demand of one `neural_processor.v` at its own maximum pipelined rate) — a hard floor, not a design margin. `PREFETCH_DEPTH` (tiles of lookahead) needed to actually reach that floor scales with round-trip latency, independent of bandwidth: ≈ 4 tiles hides 0–1 cycle latency; ≈ 16 tiles is *not yet enough* to hide 16 cycles (83.4% measured, not 90%+).

13.3 STEP2 — closed-form traffic model

Per slot at steady state: **weight** traffic is always $P_IN=8$ B/cycle (never shared, no amortization possible ever); **activation** traffic is 8 B/cycle worst case (no sharing) down to ≈ 0 amortized (full sharing across a layer); **result** traffic is negligible ($1/n_tiles$ B/cycle/slot). The 16 B/cycle/slot worst-case floor measured in STEP1 is exactly $8 + 8$ — a clean cross-validation of the simulated result against the analytical model, not a coincidence.

13.4 STEP3 — real bank-contention sweep

A second simulation harness measured whether banking the shared Activation SRAM (broadcast-on-same-address, round-robin arbitration on conflict) actually lets `N_SLOTS` scale under a *realistic* dispatch stagger (the Neural Director dispatches one job at a time, never simultaneously) — the exact mechanism behind Current V2’s own flat-scaling finding. Two real bugs (fixed-priority starvation causing an actual simulation hang; a testbench/DUT handshake mismatch) were found and fixed first.

Real result: banking recovers real parallel scaling

With `N_BANKS=N_SLOTS`, aggregate throughput scales **near-linearly** regardless of dispatch stagger (0–8 cycles tested): `N_SLOTS=1` → 0.990, 2 → 1.979 (1.999×), 4 → 3.950 (3.990×), 8 → 7.869 (7.949×) tiles/cycle. With `N_BANKS=1` (matching Current V2’s own single shared port), utilization collapses under any nonzero stagger exactly as Current V2’s own real benchmark showed (e.g. `N_SLOTS=2`, stagger=1: 49.8%) — the first real, simulated confirmation in this project that `N=2>N=1` and `N=4>N=2` are achievable without the shared memory nullifying parallelism.

13.5 STEP4–7 — real candidate synthesis and selection

Two real, synthesizable candidates were built and bit-exact verified for *each* SRAM, then compared on real Yosys+nextpnr-ecp5 data (never chosen a priori):

Activation SRAM. Candidate A (`N_SLOTS` private replicated copies, broadcast-write fill) vs. Candidate B (banked + round-robin arbiter + 2-stage registered crossbar, deliberately pipelined per §5.3’s own Fmax lesson). Candidate A won decisively: 2–4× higher real Fmax and ≈24× fewer LUTs than Candidate B at `N_SLOTS=8` (`MAX_TILES=16`), for a real BRAM cost that stays cheap even at a much deeper, more realistic vector length (8 DP16KD, 7% of the chip, at `MAX_TILES=256/N_SLOTS=8`) — confirming the M3-era warning against assuming “shallower depth = less BRAM”: at `MAX_TILES=16` *neither* candidate used any real BRAM at all (Yosys chose distributed LUT-RAM for both).

Weight SRAM. Candidate W1 (one native-width memory per slot, mirroring `weight_buffer.v`’s own M3-era structure) vs. Candidate W2 (per-MAC-lane packed narrow memories). At `MAX_TILES=256` both use *identical* real DP16KD count (one full block’s own native 16 Kbit capacity per slot, either way), but packed uses ≈2× fewer LUTs/FFs at `N_SLOTS=8` for the same BRAM cost — the wide single memory’s own byte-lane write-enable decode logic is exactly what per-lane packing avoids by construction.

Selected: replicated Activation SRAM + packed Weight SRAM. Combined real cost at `N_SLOTS=8/MAX_TILES=256`: 16 DP16KD (14.8% of the LFE5U-45F’s 108 total) — an honestly affordable real price for this project’s own realistic workload sizes.

13.6 STEP8 — full integration

`nms_dataflow_core.v` mirrors `dataflow_core.v`’s own scope exactly: the Dependency Manager and Neural Director are **reused verbatim**, unmodified — only the memory cluster changed. Each slot’s own `nms_memory_manager.v` is structurally simpler than `memory_manager.v`: since the on-chip SRAMs now hold the *entire* vector (not just 2 double-buffered banks), there is no more bank-swap logic — a slot simply reads sequentially once its own weight-fetch progress and the shared activation controller’s own resident count both exceed the tile index it needs.

Four real bugs found at full integration scale

All four are the same root cause: a counter that must represent the *value* `MAX_TILES` itself (e.g. a 16-tile job with `MAX_TILES=16`) needs one more bit than an address field indexing `0..MAX_TILES-1` — easy to miss because every test smaller than `MAX_TILES` passes regardless. Found only once a real `n_tiles=MAX_TILES` job (this project’s own realistic

16-tile neurons) was actually run: a truncated 16-bit compare that read 16 as 0 (hanging weight fetch entirely); an undersized counter wrapping 15→0 instead of reaching 16 (an infinite re-fetch loop); a logic error comparing the wrong two signals introduced while fixing the first bug (deadlocking exactly the last tile of every job); and a top-level connecting wire left at the narrower width after both endpoint modules were widened (silently truncating the real value 16 back to 0 one wire short of the fix). Each was isolated via real cycle-by-cycle signal tracing, the same discipline used throughout this project.

7/7 bit-exact tests pass at `N_SLOTS=2`, including the exact scenario STEP3 modeled (two slots dispatched together on the identical `x_base`, different never-shared weights) and a new multi-tile test that specifically catches bug class 2 above.

13.7 STEP9–10 — real end-to-end benchmark vs. Current V2

`nms_neural_multiprocessor.v` mirrors `neural_multiprocessor.v`'s own real hardware-facing scope exactly (same real `slot_mem_arbiter.v`, same real, unmodified V1 PSRAM chain). The **identical** D-Stress workload (256 neurons, 16 inputs×8 tiles, one shared input vector) used for every Current-V2 number in this datasheet was run through it, bit-exact against the same golden model.

Real, direct comparison — same workload, same toolchain

Metric (<code>N_SLOTS=2</code>)	Current V2	NMS	Δ
Fmax (real P&R)	87.72 MHz	93.10 MHz	+6.1%
LUT4	4359	1948	−55.3%
CCU2C	366	266	−27.3%
TRELLIS_FF	3924	3522	−10.2%
DSP / BRAM	16 / 0	16 / 0	=
D-Stress cycles	185428	185645	+0.1%
D-Stress wall-clock	2113.9 μs	1994.0 μs	+6.0% faster
Effective MAC/s	15.50 M	16.43 M	+6.0%

Metric (<code>N_SLOTS=4</code>)	Current V2	NMS	Δ
Fmax (real P&R)	65.01 MHz (FAIL)	56.62 MHz (FAIL)	−12.9pp
D-Stress cycles	184795	184764	−0.02%
D-Stress wall-clock	2842.6 μs	3263.2 μs	−12.9% (NMS slower)

Cycles are essentially flat between `N_SLOTS=2` and 4 for *both* systems (185645→184764 for NMS, −0.5%) — confirming STEP1's own analytical floor: a single real PSRAM port caps *aggregate* throughput regardless of on-chip organization; NMS's banking work makes the on-chip side efficient, it cannot and does not remove the external bandwidth ceiling.

Real critical path found at N_SLOTS=4/8 — not hidden

Real nextpnr-ecp5 critical-path tracing at N_SLOTS=4 shows the worst path running through `nms_activation_fill_ctrl.v`'s own combinational priority-scan/address logic (6.26 ns logic + 11.40 ns routing) — the *same class* of unpipelined, N_SLOTS-scaling combinational cost §5.3 already documented for `activation_cache.v`, reintroduced here in the module that decides *which* shared tag to chase (a genuinely different piece from the replicated SRAM itself, which has no such problem in isolation). N_SLOTS≤2 is unaffected and real, measured faster; N_SLOTS≥4 is a real, open regression, not recommended, until this scan is pipelined (§13.10).

13.8 Real per-metric detail, N_SLOTS=2 (D-Stress)

Metric	Value	Note
Processor utilization	1.10%	tiles(4096)/(2×185645 cycles) — consistent with the project's own 1:170–1:220 compute-to-memory-wait finding
Memory (PSRAM port) utilization	90.4%	167830/185645 busy cycles
Memory stall (per slot)	93.6%	92.5% waiting on weight + 1.1% waiting on activation, measured directly
Compute stall	≡ memory stall	the Neural Processor stalls <i>only</i> on a missing operand in this design — no separate compute-only stall source exists
Weight-buffer hit rate	0%	confirmed empirically (2048 real fetches = 2048 tiles/slot, zero reuse) — weights are never shared, by design
Activation-buffer hit rate	99.61%	only 16 real PSRAM fetches for 4096 tile-consumptions (256 neurons share one vector)
Prefetch effectiveness	low (≈0%)	a real, honest gap: this revision fetches weight “as fast as possible” but with no bounded lookahead buffer (<code>PREFETCH_DISTANCE</code>), so weight-fetch latency dominates stall almost entirely — see §13.10
Parallel efficiency (N=2 vs. N=1)	48.1%	real speedup = cycles(1)/cycles(2) = 178432/185645 = 0.961 × (N=2 needs <i>more</i> cycles than N=1) — the shared PSRAM port is still the bottleneck

13.9 Recommendation

Adopt NMS at N_SLOTS≤2 as a real, measured upgrade over Current V2 at its own already-recommended default: faster, smaller, higher Fmax margin, bit-exact, same workload. Do **not** adopt NMS at N_SLOTS=4/8 yet — Current V2 is really faster there until the fill-controller pipelining fix below is implemented and re-measured. Both systems remain in the repository; selecting between them is a real, configuration-dependent decision, not a blanket replacement.

13.10 Open work (real, not hidden)

- **Pipeline `nms_activation_fill_ctrl.v`'s own priority-scan/address logic** — the concrete, identified fix for the N_SLOTS=4/8 Fmax regression above.

- **Implement real bounded-lookahead weight prefetch** (`PREFETCH_DISTANCE`, per STEP1's own findings) — the current single-shot “fetch as fast as possible” weight path is why prefetch effectiveness measures low; STEP1's own data shows a real, achievable fix (depth scaled to real round-trip latency).
- Re-measure `N_SLOTS=1` and 8 D-Stress cycle counts for full parity with Current V2's own 4-point table (only 2 and 4 measured this round, time-bounded).
- A fixed, smaller-`N_BANKS` Activation SRAM variant was never revisited after full replication was selected — BRAM cost was cheap enough at this project's real workload sizes that it was never worth reconsidering.

CHAPTER A

Module and file map

A.1 V2 RTL (`hardware/v2/rtl/`)

File	Role
<code>neural_processor.v</code>	8-stage INT8 pipeline (M1); bit-exact vs. V1.
<code>neural_processor_ar</code>	N-processor array used for the M2 concurrency sweep.
<code>activation_buffer.v</code> , <code>weight_buffer.v</code> , <code>result_buffer.v</code>	M3 BRAM-backed buffers; superseded in the real datapath by <code>activation_cache.v</code> .
<code>prefetch_engine.v</code>	Weight-only, word-level burst fetch engine (M4, rewritten DEC-0015/DEC-0016).
<code>memory_manager.v</code>	Double-buffered per-slot tile manager; coordinates the Activation Cache (X) and <code>prefetch_engine.v</code> (W).
<code>neural_director.v</code>	First-free job dispatch (M5).
<code>dependency_manager.v</code>	Node table, dependency counting, wake-up (M6).
<code>dataflow_core.v</code>	Full M1–M6 integration + Activation Cache (M7, extended DEC-0016).
<code>slot_mem_arbiter.v</code>	Generic N-port arbiter to the real PSRAM chain (M8).
<code>activation_cache.v</code>	Shared, single-tag activation cache (post-M10, DEC-0016).
<code>neural_multiprocessor</code>	Real hardware-facing top level (M8).

A.2 NMS RTL (`hardware/v2/nms/rtl/`, [ch. 13](#))

File	Role
<code>nms_activation_replica</code>	Selected Activation SRAM: N_{SLOTS} private full-vector copies, broadcast-write fill (DEC-0019).
<code>nms_activation_fill_c</code>	Shared dedup/fetch controller backing it – the real $N_{\text{SLOTS}}=4/8$ Fmax bottleneck identified in ch. 13 .
<code>nms_weight_packed.v</code>	Selected Weight SRAM: per-MAC-lane packed private copies (DEC-0020).
<code>nms_memory_manager.v</code>	Per-slot job FSM, drop-in replacement for <code>memory_manager.v</code> 's own external interface.
<code>nms_dataflow_core.v</code>	Full NMS integration, mirrors <code>dataflow_core.v</code> 's own scope (STEP8).
<code>nms_neural_multiproce</code>	Real hardware-facing top level, mirrors <code>neural_multiprocessor.v</code> 's own scope (STEP9).

Also reused verbatim, unmodified, in the NMS datapath: `neural_processor.v`, `prefetch_engine.v` (as a generic P_IN-byte-tile fetch engine, not weight-specific despite its name), `dependency_manager.v`, `neural_director.v`, `slot_mem_arbiter.v`.

A.3 Reused, unmodified V1 (`hardware/v1/rtl/`)

File	Role in V2
<code>memory_interface.v</code>	Word-level (16-bit) PSRAM backend port, now the direct target of both <code>prefetch_engine.v</code> and <code>activation_cache.v</code> .
<code>psram_controller.v</code>	Real PSRAM controller, page-mode support exploited more effectively by the word-burst rewrite.
<code>int8_memory_access.v</code>	No longer instantiated in V2's datapath post-DEC-0015 — file itself untouched.

A.4 Simulation (`hardware/v2/sim/`)

`tb_neural_processor.v`, `tb_dataflow_core.v`, `tb_memory_manager.v`,
`tb_neural_director.v`, `tb_dependency_manager.v`, `tb_neural_multiprocessor.v`,
`tb_benchmark_suite.v` (the final campaign's own testbench, parametric in `N_SLOTS_CFG` via Verilator's own `-G` override).

A.5 Documentation and logs (`hardware/v2/docs/`, `hardware/v2/logs/`)

`ROADMAP.md`; `docs/benchmarks/final-benchmark.md` (the 21-section pre-optimization campaign report); **append-only logs** (`development`, `simulation`, `synthesis`, `timing`, `benchmark`, `decisions`, `experiments`, `errors`) — the primary source of every number in this datasheet.