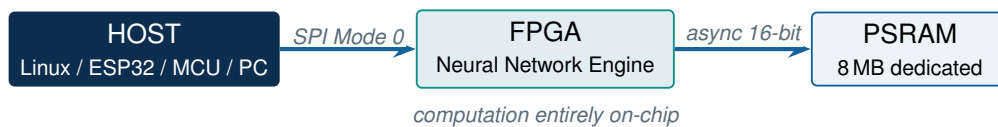


# FPGA – Neural

INT8 Neural Network Engine for FPGA

Parametric hardware accelerator – Datasheet and reference manual

Rev. A1 • September 2026



**Reference target device:** Lattice ECP5 LFE5U-45F-8BG381C (speed grade -8, CABGA381, 72×MULT18X18D, ≈44k LUT).

**Baseline configuration:** INT8/INT32, N\_INPUTS=256, N\_NEURONS=4, parametric PARALLEL, PSRAM working memory ISSI IS66WVE4M16EBLL-70BLI.

**Status:** RTL verified in simulation (Icarus) and real synthesis (Yosys + nextpnr-ecp5). Document describing the project as of September 2026.

Project author: Michele Bigi • MIKILAB / manvalan.

This datasheet documents the RTL code, documentation and benchmarks present in the repository [github.com/manvalan/FPGA-Neural](https://github.com/manvalan/FPGA-Neural).

## FPGA-Neural — General description and features

FPGA-Neural is a **parametric hardware accelerator for feed-forward neural networks** contained entirely within the FPGA. Computation (multiplication, accumulation, bias, activation, saturation) takes place entirely on-chip in INT8/INT32 integer arithmetic; the host system only provides configuration, weights, input data and control through a simple SPI interface, without ever being part of the computational datapath. A single bitstream serves any topology up to the build maximum.

### Features

- **INT8 × INT8 → INT16 → INT32** datapath, 32-bit accumulation with sign extension.
- **Balanced binary adder tree** ( $O(\log_2 \text{PARALLEL})$ ) instead of linear reduction.
- Configurable parallel MAC: `PARALLEL` simultaneous hardware MACs per neuron, mapped onto `MULT18X18D` DSPs.
- Fully **parametric** architecture: `N_INPUTS`, `N_NEURONS`, `PARALLEL`, `DATA_WIDTH`, `ACC_WIDTH`, `N_LAYERS`.
- **Runtime network width**: per-layer `n_inputs_real/n_neurons_real`, a single bitstream for every topology up to the maximum.
- Configurable activations: `ACT_RELU` (default) and `ACT_NONE` (linear with bilateral saturation), with INT8 saturation.
- **Two network types**: classic multi-layer dense (`layer_sequencer`, ping-pong buffers) and **arbitrary sparse graph** (`graph_engine` + activation buffer in `DP16KD` block RAM), selectable at runtime.
- **Dedicated memory subsystem**: byte↔word interface, asynchronous parallel PSRAM controller with **page mode** (70 ns random access, 20 ns page burst), 8 MB addressable (23 bit).
- **SPI Mode 0** MSB-first host interface, `SET_NET_TYPE+dispatch`, `STATUS.done` sticky/clear-on-read, `runtime READ_CONFIG`.
- **Flash subsystem** for boot/persistence: FPGA-exclusive access to a `W25Q128JV` SPI NOR (16 MB) via a dedicated SPI master, a flash↔PSRAM copy engine, and a 16-slot catalog with CRC32, 8 host opcodes.
- Verified in **simulation** (Icarus Verilog) and **real synthesis** (Yosys + nextpnr-ecp5 + ecppack).

### Applications

- Deterministic low-latency inference as a peripheral of a Linux SoC, Raspberry-Pi-like board, ESP32, microcontrollers.
- Reusable hardware block integrable into heterogeneous projects (a platform, not a single network).
- Edge AI on compact dense INT8-quantized networks.
- Off-loading the neural workload from the host CPU to dedicated hardware with predictable throughput.

### Target & toolchain

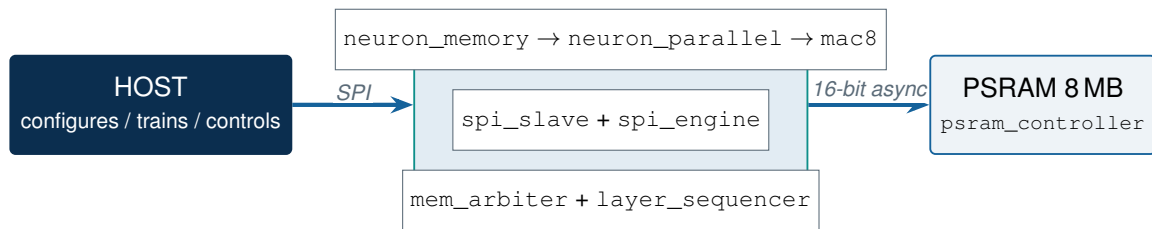
- **FPGA**: Lattice ECP5 LFE5U-45F-8BG381C (–8, CABGA381).
- **Synthesis**: Yosys; **place&route**: nextpnr-ecp5; **bitstream**: Project Trellis (ecppack).
- **Simulation**: Icarus Verilog (–g2012).
- **PSRAM**: ISSI IS66WVE4M16EBLL-70BLI (64 Mb, 4M×16).

### Key parameters (characterized baseline configuration)

| Quantity             | Value          | Notes                                         |
|----------------------|----------------|-----------------------------------------------|
| Data precision       | INT8 (signed)  | <code>DATA_WIDTH=8</code>                     |
| Accumulator          | INT32 (signed) | <code>ACC_WIDTH=32</code>                     |
| Inputs / neurons     | 256 / 4        | datapath benchmark baseline                   |
| Simultaneous MACs    | 2...64         | <code>=PARALLEL×N_NEURONS</code>              |
| Activations          | ReLU, linear   | <code>ACT_RELU</code> / <code>ACT_NONE</code> |
| Fmax (P=2, datapath) | 87.88 MHz      | isolated datapath benchmark                   |

|                               |               |                                                     |
|-------------------------------|---------------|-----------------------------------------------------|
| Fmax (P=2, integrated system) | 67.91 MHz     | full system incl. flash subsystem, real place&route |
| MAC throughput (P=16)         | ≈3.34 G MAC/s | theoretical, datapath only                          |
| Working memory                | 8 MB PSRAM    | 16-bit parallel bus, 70 ns / 20 ns page mode        |
| Address space                 | 23 bit (byte) | ADDR_WIDTH=23                                       |

### System block diagram



*The neural datapath is entirely inside the FPGA; the host does not take part in the individual MAC operations.*

## Pinout summary — pin-by-pin connection

Quick-reference table: the **57 real signals** of the top-level `spi_neuron_top`, each with its own individual CABGA381 ball (**not** a bus range) — real data from Project Trellis's device database (`iodb.json`), **verified by a complete nextpnr-ecp5 place&route run at 0 errors** (not a planned pinout). Full description, per-bank placement rationale and the pin-by-pin connection to the ISSI PSRAM: [ch. 12](#).

| Signal                                                                             | Ball | Bank | Dir | Corresponding pin / function                                                                 |
|------------------------------------------------------------------------------------|------|------|-----|----------------------------------------------------------------------------------------------|
| <i>Clock and reset</i>                                                             |      |      |     |                                                                                              |
| clk                                                                                | H5   | 7    | IN  | System clock, pad GR_PCLK7_0 (dedicated global clock).                                       |
| rst                                                                                | B4   | 7    | IN  | Global synchronous reset, active high.                                                       |
| <i>Application SPI (host ↔ FPGA, Mode 0)</i>                                       |      |      |     |                                                                                              |
| sclk                                                                               | B5   | 7    | IN  | SPI clock (CPOL=0, CPHA=0).                                                                  |
| mosi                                                                               | C5   | 7    | IN  | Master-Out Slave-In.                                                                         |
| miso                                                                               | A3   | 7    | OUT | Master-In Slave-Out.                                                                         |
| cs_n                                                                               | B3   | 7    | IN  | Chip-select, active low.                                                                     |
| <i>Host attention (active-low, level)</i>                                          |      |      |     |                                                                                              |
| data_ready_n                                                                       | C3   | 7    | OUT | Low while a result is waiting to be read.                                                    |
| irq_n                                                                              | C4   | 7    | OUT | Low while the graph engine's load-time guard has tripped.                                    |
| <i>Flash subsystem — SPI toward W25Q128JV (boot/persistence)</i>                   |      |      |     |                                                                                              |
| flash_sclk                                                                         | E3   | 7    | OUT | SPI clock toward the flash — ordinary GPIO, independent (Phase F7, <a href="#">ch. 12</a> ). |
| flash_mosi                                                                         | D3   | 7    | OUT | Master-Out Slave-In toward the onboard flash.                                                |
| flash_miso                                                                         | D5   | 7    | IN  | Master-In Slave-Out from the flash.                                                          |
| flash_cs_n                                                                         | E4   | 7    | OUT | Flash chip-select, active low.                                                               |
| <i>PSRAM address bus <code>psram_a[21:0]</code> — 22 individual balls (bank 2)</i> |      |      |     |                                                                                              |
| psram_a[0]                                                                         | E16  | 2    | OUT | PSRAM A0                                                                                     |
| psram_a[1]                                                                         | F16  | 2    | OUT | PSRAM A1                                                                                     |
| psram_a[2]                                                                         | D18  | 2    | OUT | PSRAM A2                                                                                     |
| psram_a[3]                                                                         | E17  | 2    | OUT | PSRAM A3                                                                                     |
| psram_a[4]                                                                         | E18  | 2    | OUT | PSRAM A4                                                                                     |
| psram_a[5]                                                                         | F18  | 2    | OUT | PSRAM A5                                                                                     |
| psram_a[6]                                                                         | F17  | 2    | OUT | PSRAM A6                                                                                     |
| psram_a[7]                                                                         | G16  | 2    | OUT | PSRAM A7                                                                                     |
| psram_a[8]                                                                         | G18  | 2    | OUT | PSRAM A8                                                                                     |
| psram_a[9]                                                                         | H16  | 2    | OUT | PSRAM A9                                                                                     |
| psram_a[10]                                                                        | H17  | 2    | OUT | PSRAM A10                                                                                    |
| psram_a[11]                                                                        | H18  | 2    | OUT | PSRAM A11                                                                                    |
| psram_a[12]                                                                        | J16  | 2    | OUT | PSRAM A12                                                                                    |
| psram_a[13]                                                                        | J17  | 2    | OUT | PSRAM A13                                                                                    |
| psram_a[14]                                                                        | C20  | 2    | OUT | PSRAM A14                                                                                    |
| psram_a[15]                                                                        | D19  | 2    | OUT | PSRAM A15                                                                                    |
| psram_a[16]                                                                        | E19  | 2    | OUT | PSRAM A16                                                                                    |
| psram_a[17]                                                                        | E20  | 2    | OUT | PSRAM A17                                                                                    |
| psram_a[18]                                                                        | F19  | 2    | OUT | PSRAM A18                                                                                    |
| psram_a[19]                                                                        | F20  | 2    | OUT | PSRAM A19                                                                                    |

|                                                                                   |     |   |     |                                                          |
|-----------------------------------------------------------------------------------|-----|---|-----|----------------------------------------------------------|
| psram_a[20]                                                                       | G20 | 2 | OUT | PSRAM A20                                                |
| psram_a[21]                                                                       | H20 | 2 | OUT | PSRAM A21                                                |
| psram_a[22]                                                                       | P18 | 3 | OUT | Always 0 (byte→word shift): NC on the board.             |
| <b>PSRAM data bus</b> <i>psram_dq[15:0]</i> — 16 individual balls (banks 2 and 3) |     |   |     |                                                          |
| psram_dq[0]                                                                       | K18 | 2 | IO  | PSRAM DQ0                                                |
| psram_dq[1]                                                                       | C18 | 2 | IO  | PSRAM DQ1 (dual-function ball, used as ordinary GPIO).   |
| psram_dq[2]                                                                       | D17 | 2 | IO  | PSRAM DQ2                                                |
| psram_dq[3]                                                                       | D20 | 2 | IO  | PSRAM DQ3                                                |
| psram_dq[4]                                                                       | G19 | 2 | IO  | PSRAM DQ4                                                |
| psram_dq[5]                                                                       | J18 | 2 | IO  | PSRAM DQ5                                                |
| psram_dq[6]                                                                       | J19 | 2 | IO  | PSRAM DQ6                                                |
| psram_dq[7]                                                                       | J20 | 2 | IO  | PSRAM DQ7                                                |
| psram_dq[8]                                                                       | K19 | 2 | IO  | PSRAM DQ8                                                |
| psram_dq[9]                                                                       | K20 | 2 | IO  | PSRAM DQ9                                                |
| psram_dq[10]                                                                      | L17 | 3 | IO  | PSRAM DQ10                                               |
| psram_dq[11]                                                                      | M18 | 3 | IO  | PSRAM DQ11                                               |
| psram_dq[12]                                                                      | M17 | 3 | IO  | PSRAM DQ12                                               |
| psram_dq[13]                                                                      | N16 | 3 | IO  | PSRAM DQ13                                               |
| psram_dq[14]                                                                      | N18 | 3 | IO  | PSRAM DQ14                                               |
| psram_dq[15]                                                                      | P17 | 3 | IO  | PSRAM DQ15                                               |
| <b>PSRAM control</b>                                                              |     |   |     |                                                          |
| psram_ce_n                                                                        | N17 | 3 | OUT | PSRAM CE# — chip enable, active low.                     |
| psram_oe_n                                                                        | R16 | 3 | OUT | PSRAM OE# — output enable (read).                        |
| psram_we_n                                                                        | R17 | 3 | OUT | PSRAM WE# — write enable.                                |
| psram_lb_n                                                                        | T16 | 3 | OUT | PSRAM LB# — lower-byte enable (DQ[7:0]).                 |
| psram_ub_n                                                                        | N19 | 3 | OUT | PSRAM UB# — upper-byte enable (DQ[15:8]).                |
| psram_zz_n                                                                        | N20 | 3 | OUT | PSRAM ZZ# — sleep/snooze (high during normal operation). |

Standard I/O: LVCMOS33 on all 57 signals. Boot config-SPI and JTAG balls (fixed-function dedicated pins, no RTL port) do not appear in this table — see ch. 12 §“Configuration and programming”. Source: `synth/ecp5/spi_neuron_top.lpf`, generated by `tools/pinout/gen_lpf.py` against Project Trellis’s `iodb.json`.

# Contents

---

|          |                                                               |           |
|----------|---------------------------------------------------------------|-----------|
| <b>1</b> | <b>System overview</b>                                        | <b>1</b>  |
| 1.1      | Project goal . . . . .                                        | 1         |
| 1.2      | Hardware configuration versus network configuration . . . . . | 1         |
| 1.3      | Boot and initialization . . . . .                             | 2         |
| 1.4      | Training and inference . . . . .                              | 2         |
| 1.5      | Design philosophy and reuse . . . . .                         | 2         |
| <b>2</b> | <b>RTL architecture</b>                                       | <b>3</b>  |
| 2.1      | Hierarchical organization . . . . .                           | 3         |
| 2.2      | Role of each module . . . . .                                 | 3         |
| 2.3      | Two execution paths . . . . .                                 | 4         |
| <b>3</b> | <b>Compute datapath</b>                                       | <b>6</b>  |
| 3.1      | INT8/INT32 arithmetic chain . . . . .                         | 6         |
| 3.2      | mac_unit — multiply-accumulator . . . . .                     | 6         |
| 3.3      | mac8 — parallel MAC and balanced adder tree . . . . .         | 6         |
| 3.4      | neuron_parallel — neuron FSM . . . . .                        | 7         |
| 3.4.1    | Parameter guard (elaboration-time) . . . . .                  | 7         |
| 3.5      | Activation functions . . . . .                                | 8         |
| 3.6      | INT8 saturation . . . . .                                     | 8         |
| 3.7      | layer — neurons in parallel . . . . .                         | 8         |
| <b>4</b> | <b>Parameters and configurability</b>                         | <b>9</b>  |
| 4.1      | Build parameters (synthesis-time) . . . . .                   | 9         |
| 4.2      | Runtime network width . . . . .                               | 9         |
| 4.2.1    | Measured savings . . . . .                                    | 10        |
| 4.3      | Characterized configurations . . . . .                        | 10        |
| 4.4      | Build versus runtime summary . . . . .                        | 10        |
| <b>5</b> | <b>Memory subsystem</b>                                       | <b>11</b> |
| 5.1      | Memory chain . . . . .                                        | 11        |
| 5.2      | int8_memory_access — byte/word conversion . . . . .           | 11        |
| 5.3      | memory_interface — handshake . . . . .                        | 11        |
| 5.4      | psram_controller — physical bus . . . . .                     | 11        |
| 5.4.1    | Timing . . . . .                                              | 12        |
| 5.5      | Read page mode . . . . .                                      | 12        |
| 5.6      | Address map and conventions . . . . .                         | 13        |
| 5.6.1    | PSRAM physical addressing . . . . .                           | 13        |
| 5.7      | Bandwidth . . . . .                                           | 13        |
| <b>6</b> | <b>Memory, multi-neuron and multi-layer</b>                   | <b>15</b> |
| 6.1      | neuron_memory — memory/neuron bridge . . . . .                | 15        |
| 6.2      | layer_sequencer — multi-layer network . . . . .               | 15        |
| 6.2.1    | Ping-pong buffers . . . . .                                   | 16        |
| 6.2.2    | Descriptor table . . . . .                                    | 16        |
| 6.3      | Hierarchy of the busy/done signals . . . . .                  | 16        |

|           |                                                                        |           |
|-----------|------------------------------------------------------------------------|-----------|
| <b>7</b>  | <b>Graph network (Type #2)</b>                                         | <b>17</b> |
| 7.1       | Two network types                                                      | 17        |
| 7.2       | Global activation buffer                                               | 17        |
| 7.3       | Feed-forward DAG and the <code>src_id &lt; out_id</code> rule          | 17        |
| 7.4       | Data formats                                                           | 17        |
| 7.4.1     | Type #2 descriptor (graph)                                             | 18        |
| 7.4.2     | Graph edge (4 bytes, aligned)                                          | 18        |
| 7.5       | <code>graph_engine</code> — graph engine                               | 18        |
| 7.6       | Type #2 opcodes and registers                                          | 19        |
| 7.7       | Occupancy (Type #2 enabled)                                            | 19        |
| 7.8       | Gather bandwidth (measured)                                            | 20        |
| 7.9       | <code>netasm</code> host assembler                                     | 20        |
| <b>8</b>  | <b>SPI host interface</b>                                              | <b>21</b> |
| 8.1       | Physical layer                                                         | 21        |
| 8.2       | Framing and explicit length                                            | 21        |
| 8.3       | Opcode table                                                           | 21        |
| 8.4       | <code>SET_BASE</code> selectors                                        | 23        |
| 8.5       | <code>STATUS.done</code> sticky / clear-on-read                        | 24        |
| 8.6       | Host attention pins ( <code>data_ready_n</code> , <code>irq_n</code> ) | 24        |
| 8.7       | <code>READ_CONFIG</code>                                               | 25        |
| 8.8       | Flash subsystem (opcodes 0x40–0x47, completed 2026-09-04)              | 25        |
| 8.9       | Session sequences                                                      | 26        |
| 8.9.1     | Single-layer path                                                      | 26        |
| 8.9.2     | Multi-layer path ( <code>RUN_NETWORK</code> )                          | 26        |
| <b>9</b>  | <b>Network programming</b>                                             | <b>28</b> |
| 9.1       | General flow                                                           | 28        |
| 9.2       | Registers and opcodes involved                                         | 28        |
| 9.3       | Type #1 — dense network                                                | 29        |
| 9.3.1     | Memory layout                                                          | 29        |
| 9.3.2     | Worked example: a $4 \rightarrow 4 \rightarrow 2$ network              | 29        |
| 9.3.3     | Host pseudocode (dense)                                                | 29        |
| 9.4       | Type #2 — graph network                                                | 30        |
| 9.4.1     | Memory layout                                                          | 30        |
| 9.4.2     | Worked example                                                         | 30        |
| 9.4.3     | Host pseudocode (graph)                                                | 30        |
| 9.4.4     | <code>netasm</code> pseudo-assembly                                    | 31        |
| <b>10</b> | <b>Arbitration and top-level</b>                                       | <b>32</b> |
| 10.1      | <code>mem_arbiter</code> — three-port arbiter                          | 32        |
| 10.2      | <code>spi_neuron_top</code> — full integration                         | 32        |
| <b>11</b> | <b>ECP5 implementation</b>                                             | <b>34</b> |
| 11.1      | Flow and verification                                                  | 34        |
| 11.2      | Datapath benchmark ( $256 \times 4$ )                                  | 34        |
| 11.2.1    | Fmax and throughput versus parallelism                                 | 35        |
| 11.2.2    | Interpretation                                                         | 35        |
| 11.2.3    | Critical path and the 100 MHz limit                                    | 35        |
| 11.3      | Full integrated system                                                 | 35        |
| 11.3.1    | Cause: the saturation/ReLU carry chain                                 | 35        |
| 11.3.2    | Timing closure (2026-09-03)                                            | 36        |

|                                                                       |           |
|-----------------------------------------------------------------------|-----------|
| <b>12 Hardware design and pinout</b>                                  | <b>37</b> |
| 12.1 Target device                                                    | 37        |
| 12.2 Pin budget                                                       | 37        |
| 12.3 Signal map (top-level <code>spi_neuron_top</code> ) — real balls | 38        |
| 12.4 Per-bank allocation (real die geometry)                          | 40        |
| 12.5 PSRAM subsystem                                                  | 40        |
| 12.5.1 PSRAM connection (FPGA-exclusive)                              | 40        |
| 12.6 Clock                                                            | 41        |
| 12.7 Power                                                            | 41        |
| 12.8 Configuration and programming                                    | 41        |
| 12.8.1 JTAG (development / debug)                                     | 41        |
| 12.8.2 Config-SPI to boot flash                                       | 41        |
| 12.8.3 Configuration modes ( <code>CFGMDN</code> )                    | 42        |
| 12.9 Open tasks before schematic capture                              | 42        |
| <b>13 Quick reference</b>                                             | <b>44</b> |
| 13.1 SPI opcodes                                                      | 44        |
| 13.2 STATUS byte                                                      | 44        |
| 13.3 SET_BASE selectors                                               | 44        |
| 13.4 Descriptor table (11 bytes/layer, MSB-first)                     | 44        |
| 13.5 Build parameters                                                 | 44        |
| <b>14 Roadmap and development status</b>                              | <b>45</b> |
| 14.1 Development phases                                               | 45        |
| 14.2 Component status                                                 | 45        |
| 14.3 Architectural principle (summary)                                | 46        |
| 14.4 Long-term vision                                                 | 46        |
| <b>A Modules and toolchain</b>                                        | <b>47</b> |
| A.1 List of RTL modules                                               | 47        |
| A.2 Ports of the top-level <code>spi_neuron_top</code>                | 47        |
| A.3 Toolchain                                                         | 47        |
| A.3.1 Main nextpnr parameters                                         | 47        |
| A.3.2 Simulation example                                              | 48        |
| A.4 Main testbenches                                                  | 48        |

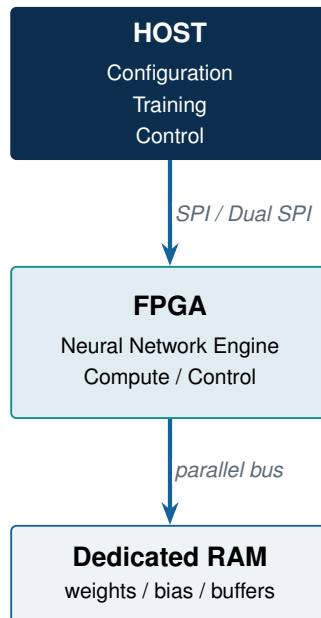
## CHAPTER 1

# System overview

### 1.1 Project goal

FPGA-Neural implements a **reusable Neural Network Engine in FPGA hardware**. The whole is made of three elements: the FPGA, which is the actual accelerator; a dedicated RAM physically associated with the FPGA and not shared with the host; and a host interface independent of the operating system, initially SPI (with possible future extension to Dual SPI).

The founding principle is the separation between who *executes* the computation and who *uses* it: the neural network computation happens entirely inside the FPGA, while the host system only provides configuration, network parameters, input data, control and result readback. The host is not part of the computational datapath. Possible host systems include Linux SoCs, Raspberry Pi-like systems, ESP32, microcontrollers and development PCs: the same engine architecture must be usable in completely different systems.



### 1.2 Hardware configuration versus network configuration

The project draws a precise distinction between the accelerator's **hardware architecture** and the **neural network parameters**.

The physical architecture of the engine is defined at FPGA synthesis and implementation time. Typical hardware parameters are `N_INPUTS`, `N_NEURONS`, `N_LAYERS`, `PARALLEL`, `DATA_WIDTH`, `ACC_WIDTH`: they are Verilog parameters resolved at synthesis and they determine the datapath contained in the bitstream. The network parameters — weights, bias, activation and quantization parameters, specific constants — are instead loaded at runtime through the host interface and stored in the RAM associated with the FPGA.

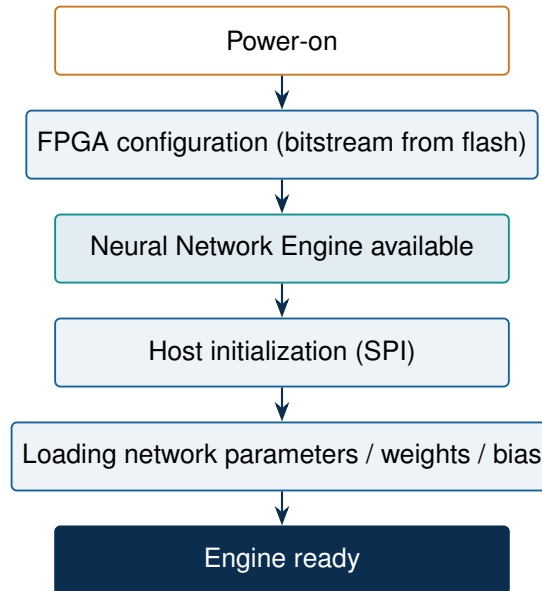
#### Central architectural principle

A build fixes the *ceiling* of the machine (maximum number of layers, maximum width, `PARALLEL`); the host configures the *actual* network — number of layers, per-layer input/output width, per-layer activation and trained parameters — entirely at runtime, over SPI, into the

FPGA's local memory. A single bitstream serves any topology up to that ceiling.

### 1.3 Boot and initialization

The FPGA is configured at power-on through the usual configuration mechanism (bitstream loading from SPI flash). The bitstream defines the hardware architecture of the engine; the host does not dynamically build the datapath during normal operation, but rather configures the network data on which the already existing datapath operates.



### 1.4 Training and inference

Training and inference are conceptually separate. The first implementation does not require the FPGA to perform training: weights can be computed externally (PC/Linux/other host) and transferred over SPI into the FPGA's RAM, which then performs inference. This drastically reduces the complexity of the initial hardware, without precluding a future implementation of assisted or fully hardware training (roadmap Phase 8, ch. 14). During inference the host only provides the input data and retrieves the result, obtaining deterministic computation, reduced host load, hardware parallelism, predictable latency and independence from the host CPU architecture.

### 1.5 Design philosophy and reuse

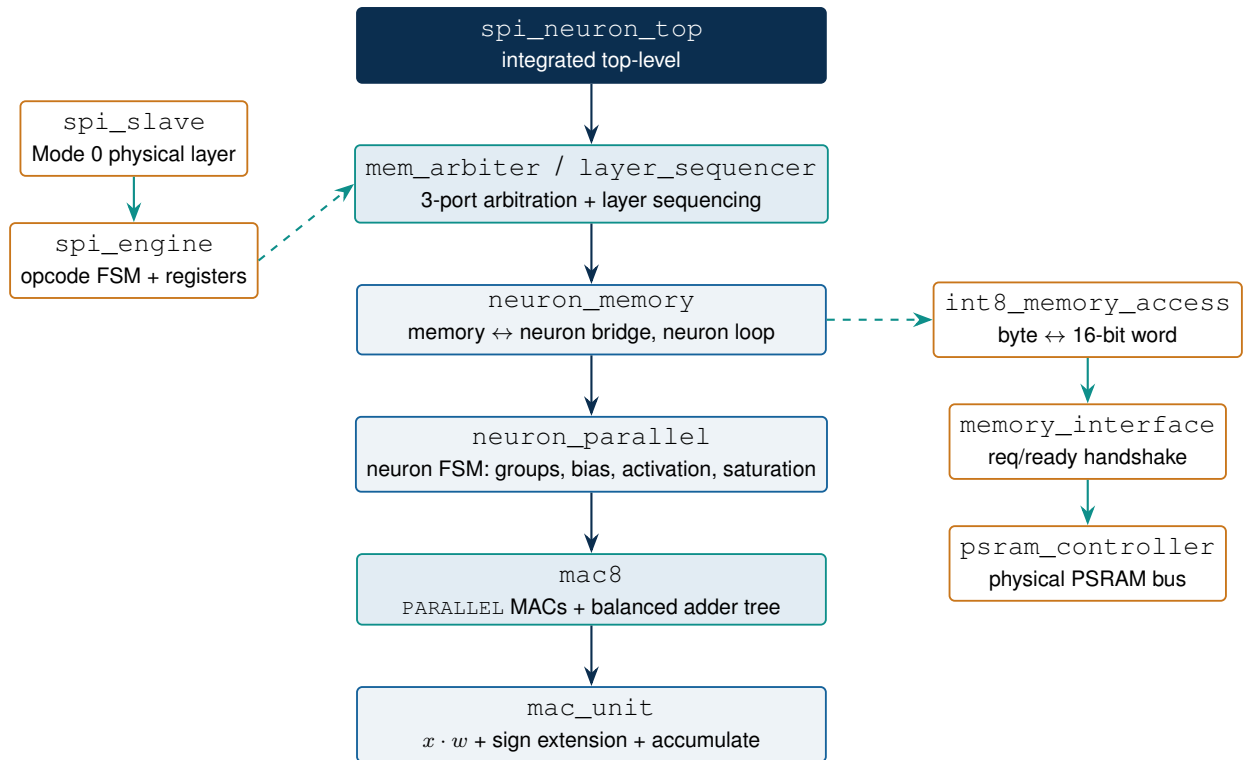
The project should be understood as a *reusable FPGA neural acceleration platform* rather than a single network. The application determines input size, topology, number of layers and neurons, parallelism, numeric precision, activation functions, memory and performance requirements; the hardware generation process produces the corresponding FPGA implementation. The same HDL architecture remains conceptually unchanged while the synthesis parameters generate implementations appropriate to the different application targets.

## CHAPTER 2

# RTL architecture and module hierarchy

### 2.1 Hierarchical organization

The design is organized in layers, from the elementary multiply-accumulator up to the integrated top-level with SPI interface and PSRAM. Each layer encapsulates the previous one and abstracts away its details: the validated datapath (`mac_unit`, `mac8`, `neuron_parallel`) is never modified by the higher orchestration layers.



### 2.2 Role of each module

| Module                       | Function                                                                                                                                                                                                                                                                                                         |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>mac_unit</code>        | Single multiply-accumulate: $\text{acc\_out} = \text{acc\_in} + (x \cdot w)$ , with sign extension of the product to <code>ACC_WIDTH</code> . Parametric on <code>DATA_WIDTH/ACC_WIDTH</code> .                                                                                                                  |
| <code>mac8</code>            | PARALLEL instances of <code>mac_unit</code> whose products are summed by a <i>balanced binary adder tree</i> of depth $\log_2(\text{PARALLEL})$ ; the result is added to the input accumulator.                                                                                                                  |
| <code>neuron_parallel</code> | FSM of a single neuron: processes <code>N_INPUTS</code> inputs in groups of <code>PARALLEL</code> , accumulates across groups, adds the bias, applies the activation and saturates to INT8. Includes the processing guard on <code>N_INPUTS % PARALLEL</code> and the runtime width <code>n_inputs_real</code> . |

|                    |                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| layer              | Instantiates <code>N_NEURONS</code> neurons <i>in parallel</i> on the same input vector; <code>busy=OR</code> , <code>done=AND</code> of the neurons. A purely data-combinational path used in the datapath benchmarks.                                                                                                                                                   |
| neuron_memory      | Integrates computation with memory: reads $X$ (shared) once, then for each neuron re-reads $W$ and bias from RAM and reuses a single <code>neuron_parallel</code> instance (memory-bound, one neuron at a time). Output <code>y_bus</code> packed neuron-major.                                                                                                           |
| layer_sequencer    | Chains up to <code>N_LAYERS</code> executions of <code>neuron_memory</code> by reading a descriptor table written by the host and alternating the ping-pong buffers in RAM (Phase 5).                                                                                                                                                                                     |
| act_buffer         | Global activation buffer in DP16KD block RAM, indexed by signal id (Type #2).                                                                                                                                                                                                                                                                                             |
| graph_engine       | Graph-network engine (Type #2): gather from <code>act_buffer</code> , reuses <code>neuron_parallel</code> , writes outputs by id (ch. 7).                                                                                                                                                                                                                                 |
| int8_memory_access | Converts the byte/INT8 interface (byte address) into the 16-bit word interface, selecting the low/high byte via <code>lb_n/ub_n</code> and <code>addr»1</code> .                                                                                                                                                                                                          |
| memory_interface   | 2-state handshake FSM (IDLE/WAIT) that serializes the single transaction toward the controller.                                                                                                                                                                                                                                                                           |
| psram_controller   | Asynchronous parallel PSRAM bus controller with read <b>page mode</b> : 70 ns random access ( $t_{AA}$ ), 20 ns same-page bursts ( $t_{APA}$ ) with <code>CE#</code> / <code>OE#</code> held asserted; enables page mode on the chip at boot via the configuration register (ch. 5, § 5.5). Drives <code>ce_n/oe_n/we_n/lb_n/ub_n/zz_n</code> and the tri-state data bus. |
| mem_arbiter        | Fixed-priority arbiter ( $B > C > A$ ) among three byte-level masters: <code>spi_engine</code> (A), <code>neuron_memory</code> (B), <code>layer_sequencer</code> (C).                                                                                                                                                                                                     |
| spi_slave          | SPI Mode 0 physical layer, MSB-first, 3-stage CDC synchronizer on <code>SCLK</code> / <code>MOSI</code> / <code>CS_N</code> , shift register and CS framing.                                                                                                                                                                                                              |
| spi_engine         | Protocol/opcode FSM and register bank ( <code>x_base</code> , <code>w_base</code> , <code>bias_addr</code> , ping-pong base, activation, runtime widths...), with sticky/clear-on-read <code>STATUS.done</code> .                                                                                                                                                         |
| spi_neuron_top     | Top-level: connects SPI, arbiter, sequencer, <code>neuron_memory</code> and the PSRAM chain; multiplexes control of <code>neuron_memory</code> between the sequencer and the direct single-layer path.                                                                                                                                                                    |

### Simulation models

`psram_model.v` (in `sim/`) and `memory_model.v` are behavioral memory models used in the testbenches; they are not part of the synthesizable design but they reproduce the real latency for end-to-end verification.

## 2.3 Two execution paths

The top-level exposes two mutually exclusive modes toward the same `neuron_memory` compute engine:

- **Single-layer / manual path:** the host sets the bases with `SET_BASE`, starts with `START` and reads with `READ_OUTPUT`. `spi_engine` drives `neuron_memory` directly.
- **Multi-layer path:** the host writes the descriptor table and starts with `RUN_NETWORK`; `layer_sequencer` takes over control of `neuron_memory` (while `seq_busy` is high) and chains the layers.

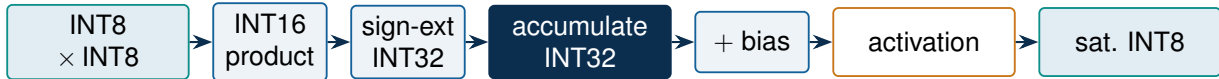
The top-level multiplexer switches the control lines of `neuron_memory` based on `seq_busy`, returning the engine to the direct path at the end of the sequence.

## CHAPTER 3

# Compute datapath

### 3.1 INT8/INT32 arithmetic chain

The elementary datapath implements the typical sequence of a quantized neuron:



Each  $\text{INT8} \times \text{INT8}$  product fits in 16 bits; it is sign-extended to 32 bits before accumulation, so the accumulator does not overflow on long vectors. Bias and activation operate at 32 bits; only the final output is saturated to INT8.

### 3.2 mac\_unit — multiply-accumulator

The `mac_unit` module is purely combinational and parametric on `DATA_WIDTH` and `ACC_WIDTH`. It computes:

$$\text{acc\_out} = \text{acc\_in} + \text{signext}_{ACC}(x \cdot w)$$

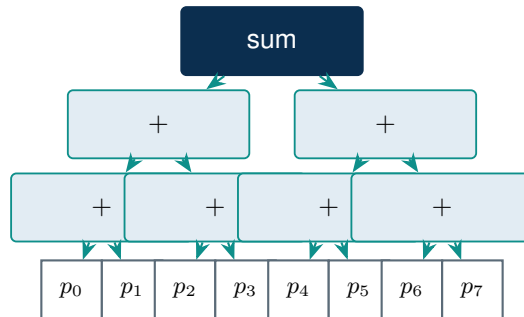
The product has width  $2 \times \text{DATA\_WIDTH}$  and is sign-extended by replicating the most significant bit. On ECP5 the multiplication maps onto a `MULT18X18D` DSP block.

**Listing 3.1.** `rtl/mac_unit.v` — arithmetic core

```
1 localparam PROD_WIDTH = 2 * DATA_WIDTH;
2 wire signed [PROD_WIDTH-1:0] product = x * w;
3 wire signed [ACC_WIDTH-1:0] product_ext =
4   {{(ACC_WIDTH-PROD_WIDTH){product[PROD_WIDTH-1]}}, product};
5 assign acc_out = acc_in + product_ext;
```

### 3.3 mac8 — parallel MAC and balanced adder tree

`mac8` instantiates `PARALLEL` `mac_unit` units that generate `PARALLEL` independent products, then sums them with a *balanced binary adder tree*. Compared to the linear reduction  $((((p_0 + p_1) + p_2) + p_3) + \dots)$ , of depth  $O(\text{PARALLEL})$ , the tree has depth  $O(\log_2 \text{PARALLEL})$ , drastically reducing the combinational path.



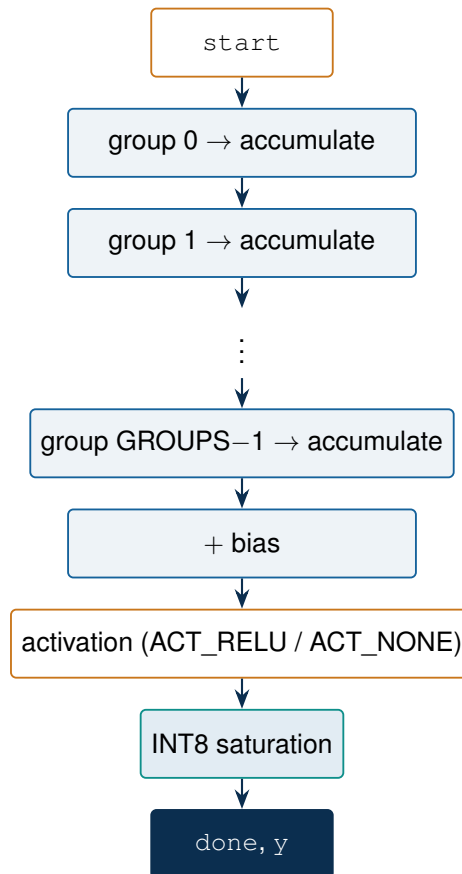
Example with  $\text{PARALLEL}=8$ : 3 levels.  $\text{PARALLEL}=16 \rightarrow 4$  levels;  $\text{PARALLEL}=32 \rightarrow 5$  levels.

### PARALLEL as a power of two

The tree is designed for `PARALLEL` as a power of two (8, 16, 32...). This is also the value used in all project configurations.

## 3.4 neuron\_parallel — neuron FSM

`neuron_parallel` processes `N_INPUTS` inputs in groups of `PARALLEL`, maintaining the accumulator from one group to the next. At the end it adds the bias, applies the activation and saturates to `INT8`. The number of groups is `GROUPS = N_INPUTS / PARALLEL`.



### 3.4.1 Parameter guard (elaboration-time)

If `PARALLEL` does not exactly divide `N_INPUTS` two failures occur, both confirmed empirically in `sim/parameter_sweep_tb.v`:

- integer division truncates `GROUPS` and the excess inputs are never read → **wrong** result, with no error and no warning;
- if `PARALLEL > N_INPUTS`, `GROUPS=0` and the terminal condition is never satisfied → the neuron **hangs** (busy high, done never asserted).

The solution does not modify the validated datapath: a `generate` block instantiates a deliberately undefined module when `N_INPUTS mod PARALLEL ≠ 0`, forcing an error at *elaboration* both in simulation and in synthesis. For valid configurations the branch is never elaborated.

**Listing 3.2.** `rtl/neuron_parallel.v` — parameter guard

```

1 generate
2   if (N_INPUTS == 0 || N_INPUTS % PARALLEL != 0) begin : PARAMETER_ERROR
3     neuron_parallel_requires_N_INPUTS_multiple_of_PARALLEL
4     invalid_parameter_combination();
5   end
6 endgenerate

```

**Edge case `N_INPUTS=0` (fixed 2026-09-04)**

The original condition (`N_INPUTS % PARALLEL != 0`) does not catch `N_INPUTS=0`, since  $0 \bmod \text{PARALLEL} = 0$  for any `PARALLEL`: the module elaborated successfully (both in simulation and in real Yosys synthesis) while leaving `x_bus/w_bus` undriven and `start` silently ineffective. Found during the re-certification campaign (`docs/validation/bugs.md`, BUG-002) and fixed by extending the guard as above — `N_INPUTS=0` now fails elaboration exactly like the other degenerate cases.

**3.5 Activation functions**

`neuron_parallel` accepts a 2-bit `activation` port. The default is `ACT_RELU`, the only behavior that existed before the port was introduced, so every pre-existing caller remains unchanged.

| Encoding              | Value             | Behavior                                                                                               |
|-----------------------|-------------------|--------------------------------------------------------------------------------------------------------|
| <code>ACT_NONE</code> | <code>2'd0</code> | Linear: no clamp to zero, bilateral saturation to the INT8 range $[-128, +127]$ .                      |
| <code>ACT_RELU</code> | <code>2'd1</code> | $\max(0, x)$ , then positive saturation to $+127$ (default; also the fallback for reserved encodings). |

**3.6 INT8 saturation**

After bias and activation, the 32-bit accumulator is reduced to INT8:

$$y = \begin{cases} +127 & \text{if } \text{final\_acc} > 127 \\ -128 & \text{if } \text{final\_acc} < -128 \text{ (ACT\_NONE only)} \\ 0 & \text{if } \text{final\_acc} \leq 0 \text{ (ACT\_RELU only)} \\ \text{final\_acc}[7:0] & \text{otherwise} \end{cases}$$

**3.7 `layer` — neurons in parallel**

`layer` instantiates `N_NEURONS` neurons that share the input vector `x_bus` but have distinct weights and bias; `busy` is the OR and `done` the AND of the neurons' signals. It is the module used in the datapath benchmarks (ch. 11), where all neurons work simultaneously. The addressing convention is neuron-major: the weights of neuron  $n$  occupy `weights_bus[n*N_INPUTS*DATA_WIDTH + : N_INPUTS*DATA_WIDTH]`.

## CHAPTER 4

# Parameters and configurability

### 4.1 Build parameters (synthesis-time)

The hardware architecture is fixed at synthesis through the following Verilog parameters. They determine the datapath contained in the bitstream and its capacity *ceiling*.

| Parameter      | Default  | Meaning                                                                                   |
|----------------|----------|-------------------------------------------------------------------------------------------|
| DATA_WIDTH     | 8        | Data width (INT8).                                                                        |
| ACC_WIDTH      | 32       | Accumulator width (INT32).                                                                |
| N_INPUTS       | 32 / 256 | Maximum number of inputs per neuron (benchmark baseline: 256).                            |
| N_NEURONS      | 1 / 4    | Maximum number of neurons per layer.                                                      |
| PARALLEL       | 8        | Simultaneous hardware MACs per neuron; must divide N_INPUTS and should be a power of two. |
| N_LAYERS       | 4        | Maximum number of layers chainable by <code>layer_sequencer</code> .                      |
| ADDR_WIDTH     | 23       | Byte-address width (8 MB).                                                                |
| MEM_DATA_WIDTH | 16       | Width of the physical PSRAM data bus.                                                     |
| CLK_FREQ_MHZ   | 80       | Frequency used in the PSRAM timing formulas (must be aligned to the real oscillator).     |

#### N\_INPUTS % PARALLEL constraint

PARALLEL must divide N\_INPUTS exactly, otherwise the elaboration guard fires (§3). The same constraint applies at runtime to `n_inputs_real`.

### 4.2 Runtime network width

A single bitstream serves any topology *up to* the build maximum. The actual width of each execution is a separate value, set by the host:

- `n_inputs_real` — inputs actually used in this execution (must be a multiple of PARALLEL);
- `n_neurons_real` — neurons actually computed in this execution.

Both default to the build maximum, so any caller that leaves them unconnected processes the full width as before the ports were introduced.

#### Real early termination

This is not mere address bookkeeping: the two values directly bound the hardware loops (X/W reads of `neuron_memory`, MAC group count of `neuron_parallel` and the length of the ping-pong copy for `RUN_NETWORK`). A narrower layer actually *computes* and *copies* faster and does not require zero-padding of the RAM for the unused tail: data beyond `n_inputs_real/n_neurons_real` is never read.

This lets a network taper within a single chained execution, for example  $256 \rightarrow 64 \rightarrow 16 \rightarrow 4$ , with each layer declaring its own actual width in the descriptor table (ch. 6).

### 4.2.1 Measured savings

Early termination was measured end-to-end:

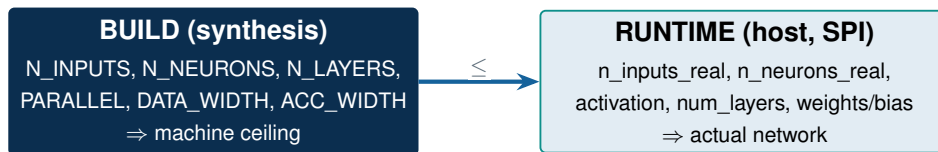
| Test                      | Cycles     | Comparison                                                                                |
|---------------------------|------------|-------------------------------------------------------------------------------------------|
| neuron_parallel_tb.v (T7) | 3 vs 6     | reduced vs full, with “garbage” data in the skipped lanes (proof that they are not read). |
| neuron_memory_tb.v (T5)   | 209 vs 788 | 8-of-32 vs full 32, through the real PSRAM stack.                                         |

### 4.3 Characterized configurations

Some combinations validated in simulation and/or synthesis:

| N_INPUTS | N_NEURONS | PARALLEL | Notes                                                           |
|----------|-----------|----------|-----------------------------------------------------------------|
| 32       | 4         | 8        | First functional parametric test (Phase 1).                     |
| 256      | 4         | 2/4/8/16 | Datapath benchmark sweep (Phase 7).                             |
| 32       | 1..3      | 8        | Single/multi-neuron memory integration (Phase 3).               |
| 4        | 4         | 2        | End-to-end 2-layer <code>RUN_NETWORK</code> test over real SPI. |

### 4.4 Build versus runtime summary

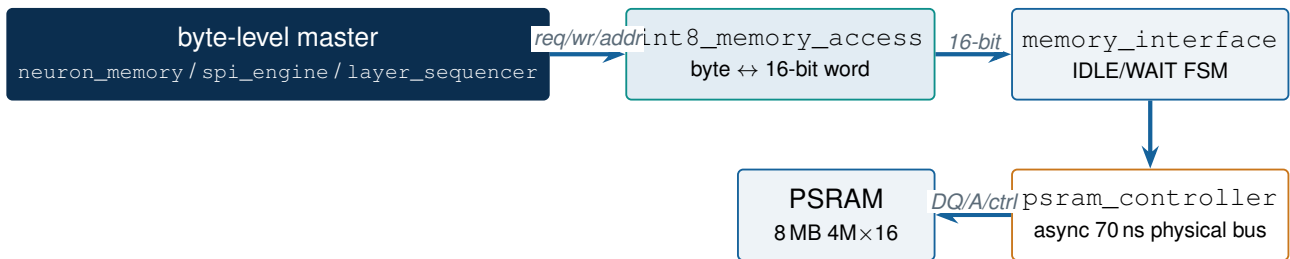


## CHAPTER 5

# Memory subsystem

### 5.1 Memory chain

The compute engine works with addresses and data at the *byte* level (INT8), while the PSRAM is a 16-bit word device. Three cascaded modules realize the conversion and the physical access:



### 5.2 int8\_memory\_access — byte/word conversion

Converts the INT8 interface (byte address) into the word interface. The byte address is divided by two ( $\text{addr} \gg 1$ ) to obtain the word address; the least significant bit selects the byte:

- $\text{addr}[0]=0 \rightarrow$  low byte:  $\text{lb\_n}=0$ ,  $\text{ub\_n}=1$ , data on  $\text{DQ}[7:0]$ ;
- $\text{addr}[0]=1 \rightarrow$  high byte:  $\text{lb\_n}=1$ ,  $\text{ub\_n}=0$ , data on  $\text{DQ}[15:8]$ .

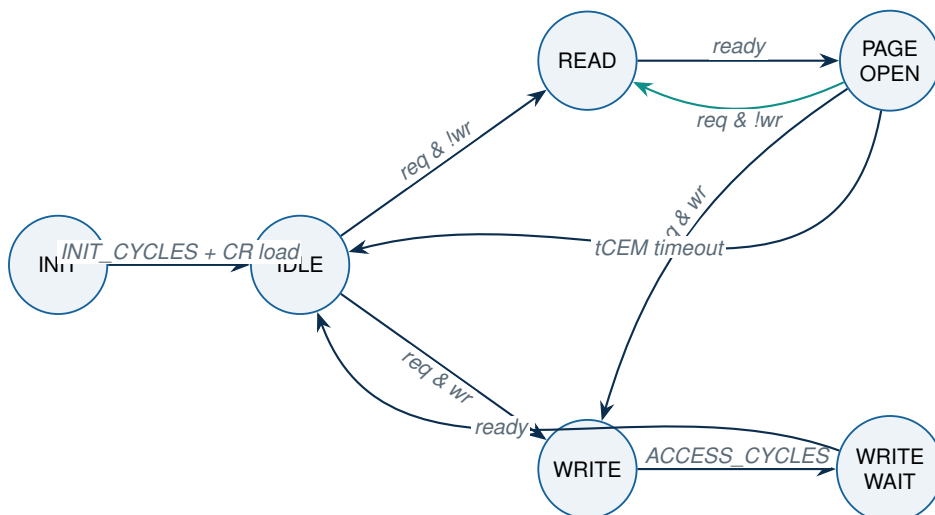
On read it extracts the correct byte from `mem_rdata`. The FSM has two states (IDLE, WAIT) and returns `ready` as a one-cycle pulse.

### 5.3 memory\_interface — handshake

Two-state FSM that serializes a single transaction: in IDLE, on the `req` request, it latches `wr/addr/wdata/lb_n/ub_n` and emits a one-cycle `mem_req` pulse toward the controller; in WAIT it waits for `mem_ready`, captures `rdata` on read and asserts `ready`. It guarantees the “one transaction at a time” contract.

### 5.4 psram\_controller — physical bus

Asynchronous parallel PSRAM bus controller, with support for the chip’s read **page mode** (§ 5.5). The main state machine is:



From INIT the controller automatically goes through a configuration-register load sub-sequence (STATE\_CR\_INIT, 4 steps) before reaching IDLE for the first time — see § 5.5. The PAGE\_OPEN → WRITE transition (bottom-right arrow) internally passes through two transit micro-states, STATE\_PAGE\_CLOSE and STATE\_PAGE\_REOPEN (one cycle each): the first forces CE#/OE# high for at least one cycle before the controller starts driving the data bus, avoiding contention with the PSRAM's still-active output ( $\geq t_{HZ}$ ); the second restarts the already-latched transaction exactly as IDLE would. They are not drawn as separate nodes to keep the figure readable.

### 5.4.1 Timing

#### Timing formulas

ACCESS\_CYCLES =  $\lceil (70 \times \text{CLK\_FREQ\_MHZ}) / 1000 \rceil$  (random-access latency,  $t_{AA}/t_{RC} = 70$  ns)

PAGE\_CYCLES =  $\lceil (20 \times \text{CLK\_FREQ\_MHZ}) / 1000 \rceil$  (same-page continuation,  $t_{APA}/t_{PC} = 20$  ns)

INIT\_CYCLES =  $150 \times \text{CLK\_FREQ\_MHZ}$  (power-up initialization,  $t_{PU} = 150$   $\mu$ s)

PAGE\_TIMEOUT\_CYCLES =  $\lceil (6000 \times \text{CLK\_FREQ\_MHZ}) / 1000 \rceil$  (automatic page close, safety margin under  $t_{CEM} = 8$   $\mu$ s)

The data bus is tri-state driven:  $\text{psram\_dq} = \text{dq\_oe} ? \text{dq\_out} : Z$ . On read  $\text{dq\_oe}=0$ ; on write  $\text{dq\_oe}=1$  during the  $\text{we\_n}$  pulse. A WRITE\_WAIT state keeps  $\text{ce\_n}/\text{lb\_n}/\text{ub\_n}$  active for the final hold before release.

#### This is not QSPI

This is a classic asynchronous-SRAM interface, **not** QSPI: most commercial serial/QSPI “PSRAM” parts are not compatible with this controller without a rewrite. See ch. 12 for the recommended part (parallel ISSI).

## 5.5 Read page mode

The recommended chip (ch. 12) is “asynchronous/**page mode**”: once an initial random access at  $t_{AA} = 70$  ns has been done, further reads inside the same 16-word page (address bits above A[3] unchanged) only cost  $t_{APA}/t_{PC} = 20$  ns, because CE#/OE# stay asserted and only the address bus changes. Page mode is **disabled by default** at power-up (bit 7 of the configuration register, CR = 0x0070 by default) and must be explicitly enabled.

- **Enable at boot:** right after INIT, the controller runs the datasheet’s “software-access sequence” (2 dummy reads + 2 writes, 0x0000 unlock then real CR 0x00F0 = default with the Page bit set) at the chip’s highest address — it reuses exactly the same READ/WRITE logic as every other transaction, so it goes through the same timing checks.
- **Page bursts:** after a READ the controller no longer closes CE#/OE# (PAGE\_OPEN state). A following read in the same page only waits PAGE\_CYCLES; a read crossing into a different page still avoids a CE# toggle but pays a full ACCESS\_CYCLES for that one word (any change at A[4] or above requires a new  $t_{AA}$ ). A counter closes the page before the  $t_{CEM}$  limit with a safety margin.
- **Only a WRITE closes the page.** Changes to  $\text{lb\_n}/\text{ub\_n}$  do *not* close it: `int8_memory_access` alternates these signals on nearly every access (byte-granular access over the 16-bit bus), so treating them as a close condition — the first implementation attempt — made the real workload *slower*, not faster (measured: 53.25→61.25 cycles/edge on `graph_engine`’s gather); removed, corrected to 53.25→37.53 cycles/edge (bandwidth +42%, § 5.7).

### No benefit without a sequential pattern

Page mode only speeds up accesses that stay in the same page (or nearly) while the controller is waiting for a new request with the page still open. Isolated, scattered accesses (a random address every time) still pay a full ACCESS\_CYCLES, plus a small close/reopen overhead if preceded by a WRITE or a  $t_{CEM}$  timeout: it is not a universal win, it depends on the caller's access pattern.

Real Fmax (nextpnr-ecp5, ch. 11) on the integrated `spi_neuron_top` system with Type #2 enabled: **75.73 MHz** at PARALLEL=2 (was 55.59 MHz before page mode was added) and **65.13 MHz** at PARALLEL=8, both still FAIL against the 80 MHz target but not regressed. The critical path stays, in both cases, entirely inside `u_graph_engine.u_neuron` (the `mac8/neuron_parallel` accumulate chain, ch. 11) — `psram_controller` never appears in the critical path despite page mode's resource growth.

## 5.6 Address map and conventions

The addressing space is ADDR\_WIDTH=23 bits (*byte* address), for a full 8 MB. The regions do not have hardwired addresses: their bases are registers set by the host via `SET_BASE` (single-layer path) or read from the descriptor table (multi-layer path).

| Region                   | Base                                                 | Content / convention                                                           |
|--------------------------|------------------------------------------------------|--------------------------------------------------------------------------------|
| Input $X$                | <code>x_base</code>                                  | Shared input vector, read once per invocation.                                 |
| Weights $W$              | <code>w_base</code>                                  | Neuron-major: weights of neuron $n$ at <code>w_base + n*N_INPUTS</code> bytes. |
| Bias                     | <code>bias_addr</code>                               | One byte per neuron: bias of neuron $n$ at <code>bias_addr + n</code> .        |
| Descriptor table         | <code>table_base</code>                              | <code>N_LAYERS</code> 11-byte entries (ch. 6).                                 |
| Ping-pong buffers<br>A/B | <code>buf_a_base</code> /<br><code>buf_b_base</code> | Intermediate outputs between layers.                                           |

### 5.6.1 PSRAM physical addressing

The recommended PSRAM is 4M×16 (8 MB), which requires a 22-bit word address (A0–A21). `int8_memory_access` computes `addr>1`, turning the 23-bit byte address into a 22-bit word address that maps exactly onto A0–A21; bit 22 of `psram_a` is therefore always 0 and 22 real address lines remain on the PCB.

## 5.7 Bandwidth

Measured on `graph_engine`'s edge-list gather (ch. 7), by difference between two graph sizes to isolate the per-edge cost from the fixed per-neuron overhead (`sim/graph_engine_bandwidth_tb.v`):

|                    | Before (no page mode) | After (page mode) | $\Delta$ |
|--------------------|-----------------------|-------------------|----------|
| Cycles/edge        | 53.25                 | 37.53             | –29.5%   |
| Bandwidth @80 MHz  | 6.01 MB/s             | 8.53 MB/s         | +41.9%   |
| Bandwidth @16 MHz* | 1.20 MB/s             | 1.71 MB/s         | +41.9%   |

\*recommended real oscillator (ch. 12).

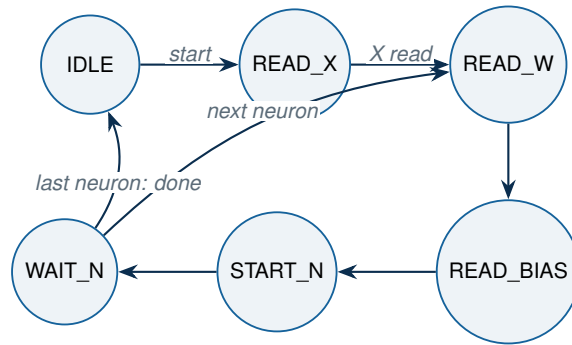
The model still remains memory-bound by construction: `neuron_memory` reads  $X$  once and re-reads  $W$ /bias for each neuron (ch. 6), one neuron at a time; page mode reduces the per-byte cost of a sequential access, it does not eliminate the access pattern itself.

## CHAPTER 6

# Memory integration, multi-neuron and multi-layer

### 6.1 neuron\_memory — memory/neuron bridge

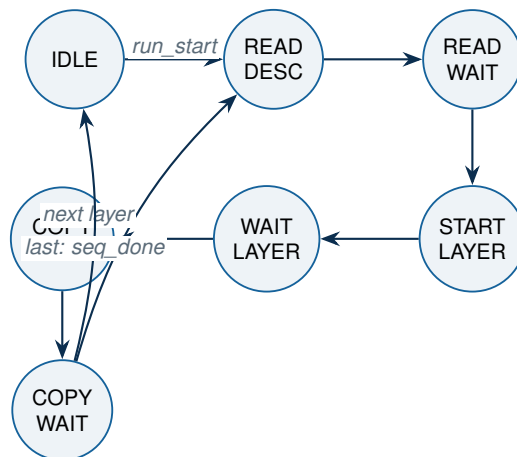
`neuron_memory` connects the compute datapath to memory and manages the loop over the neurons. It reads the  $X$  vector only once (shared input), then for each neuron re-reads  $W$  and bias from RAM and feeds them to a single reused instance of `neuron_parallel`: the design is memory-bound, one neuron computed at a time, without duplicating the datapath. The output is `y_bus`, packed neuron-major ( $\text{DATA\_WIDTH} \times \text{N\_NEURONS}$  bits).



The states are `IDLE`, `READ_X`, `READ_W`, `READ_BIAS`, `START_N`, `WAIT_N`. After the last neuron the FSM returns to `IDLE` and asserts `done`. The count of neurons and inputs actually processed is given by `n_neurons_real/n_inputs_real` (ch. 4).

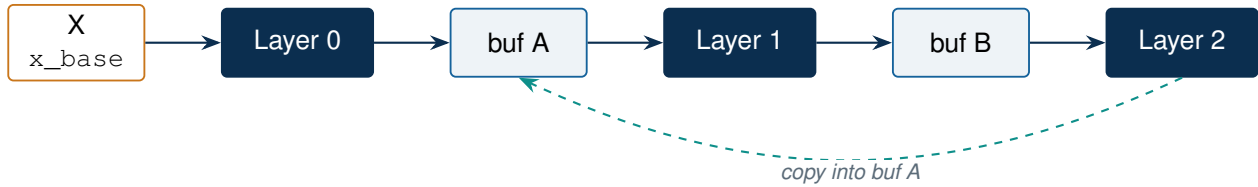
### 6.2 layer\_sequencer — multi-layer network

`layer_sequencer` chains up to `N_LAYERS` executions of the same `neuron_memory` instance, realizing a dense feed-forward network *without* touching the validated compute core. It reads a descriptor table written by the host and alternates the two output buffers in RAM (ping-pong).



### 6.2.1 Ping-pong buffers

Layer 0 reads the external input `x_base`. Layer  $k > 0$  reads from the buffer written by layer  $k - 1$ ; the output of each layer is copied into the other buffer, alternating A and B. The final output remains both in `y_bus` (readable with `READ_OUTPUT`) and in the ping-pong buffer into which it was copied.



### 6.2.2 Descriptor table

Written by the host into RAM at `table_base` with `WRITE_RAM`; `N_LAYERS` entries of 11 bytes each, MSB-first:

| Field                       | Bytes     | Meaning                                                          |
|-----------------------------|-----------|------------------------------------------------------------------|
| <code>w_base</code>         | 3         | Weight base of the layer.                                        |
| <code>bias_addr</code>      | 3         | Bias base of the layer.                                          |
| <code>activation</code>     | 1         | Layer activation (low 2 bits, cf. <code>ACT_*</code> ).          |
| <code>n_inputs_real</code>  | 2         | Actual inputs of the layer (multiple of <code>PARALLEL</code> ). |
| <code>n_neurons_real</code> | 2         | Actual neurons of the layer.                                     |
| <b>Total</b>                | <b>11</b> | <b>per entry/layer</b>                                           |

#### Copy proportional to the actual width

The sequencer copies exactly `n_neurons_real` bytes of `y_bus` into the ping-pong buffer (not the full build width): a narrower layer is copied faster, without zero-padding in RAM. Each activation is read per-layer from the table, independent of the `activation` register of the single-layer path.

### 6.3 Hierarchy of the busy/done signals

In the multi-layer path, `STATUS.busy` is the OR of the single-layer and sequencer busy signals, while `STATUS.done` latches only at completion of the *last* layer, not at each intermediate layer (ch. 8). The top-level returns control of `neuron_memory` to the direct `START` path at the end of the sequence.

## CHAPTER 7

# Two-level configuration: graph network (Type #2)

### 7.1 Two network types

The engine exposes two *network types* selectable by the host, with the same start command dispatching to the correct engine:

- **Type #1 — classic network (dense).** Layers with neurons per layer, fully connected between consecutive layers. It is the `layer_sequencer` path (ch. 6), started by `RUN_NETWORK`. Connections are *implicit by position*: nothing is enumerated, only the weights are defined, addressed as `w_base + k*n_inputs + j`.
- **Type #2 — arbitrary graph (sparse).** Starting from the input neuron ids, each neuron's connections up to the output are defined through a per-neuron *sparse edge-list*. Connections are *explicit by enumeration*: each connection is an edge `(src_id, weight)`; if it is not in the list, it does not exist.

#### The difference in one line

Dense: you define the *weights* by position in a matrix. Graph: you define each *connection* as an edge `(src_id, weight)` in a per-neuron list. The two descriptor tables share the same 11-byte format but different fields; the `net_type` register tells the engine which interpretation to use.

### 7.2 Global activation buffer

Type #2 introduces an **activation buffer** indexed by *signal id*, one INT8 byte per id, implemented in **on-chip DP16KD block RAM** (`rtl/act_buffer.v`). Ids `0..N_in-1` are the inputs; each neuron writes its own output into its own id. The source gather reads from here with *single-cycle random access*: this is what makes the graph cheap, because it is the access that PSRAM (70 ns, sequential) could not accelerate.

#### V1 sizing

`N_TOTAL=4096` signals, 16-bit id (room to 65536 without changing the format). Buffer = 4 KB, i.e. 2 DP16KD blocks out of 108. The real constraint becomes the PSRAM edge capacity ( $\approx 2$  M edges at 4 B), not block RAM.

### 7.3 Feed-forward DAG and the `src_id < out_id` rule

The graph is a feed-forward DAG: every connection points to an **already-computed** id (`src_id < out_id`). Neurons are processed in ascending id order, so that when a neuron is computed all its sources are ready in the buffer. Cycles and recurrence are out of scope for V1. The rule is checked at two levels: by the host assembler (compile time) and by a runtime guard in `graph_engine` (`STATUS.err`), in the same philosophy as the elaboration guard on `N_INPUTS % PARALLEL`.

### 7.4 Data formats

Both descriptors are 11 bytes/entry, MSB-first, at `table_base`.

### 7.4.1 Type #2 descriptor (graph)

| Field        | Bytes     | Meaning                                           |
|--------------|-----------|---------------------------------------------------|
| conn_ptr     | 3         | Byte address in PSRAM of the neuron's edge block. |
| n_conn       | 2         | Real connections (pre-padding).                   |
| out_id       | 2         | Id into which the neuron's output is written.     |
| activation   | 1         | ACT_RELU / ACT_NONE (low 2 bits).                 |
| bias         | 1         | Neuron bias (INT8).                               |
| reserved     | 2         | 0.                                                |
| <b>Total</b> | <b>11</b> | entries in ascending <code>out_id</code> order    |

### 7.4.2 Graph edge (4 bytes, aligned)

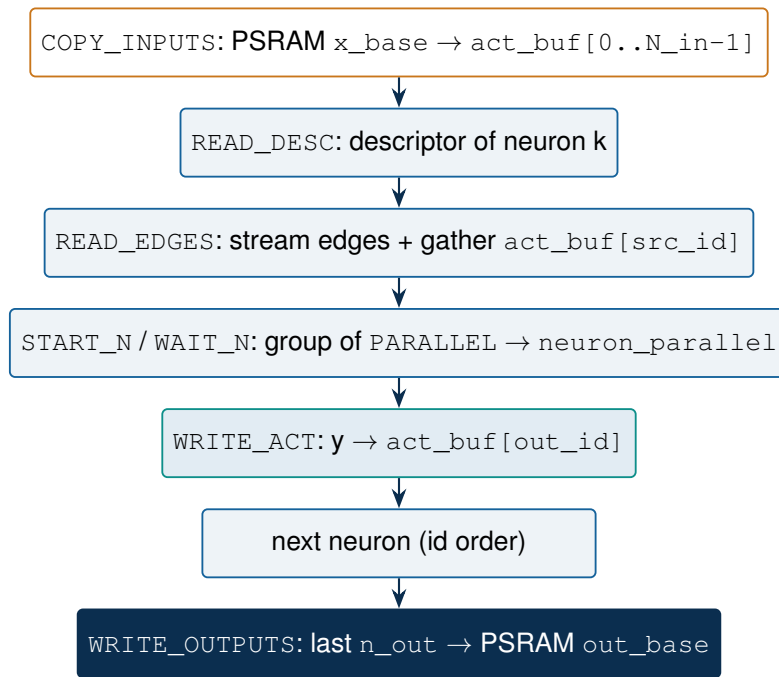
| Field    | Bytes | Meaning                |
|----------|-------|------------------------|
| src_id   | 2     | Source id (uint16 BE). |
| weight   | 1     | Weight (INT8).         |
| reserved | 1     | 0 (4-byte alignment).  |

#### Padding to PARALLEL

An arbitrary `n_conn` is not a multiple of `PARALLEL`: the neuron's edge-list is padded up to the multiple with **zero-weight** edges (waste  $\leq \text{PARALLEL}-1$  per neuron). This keeps the datapath and its guard intact.

## 7.5 graph\_engine — graph engine

`rtl/graph_engine.v` orchestrates Type #2 **reusing neuron\_parallel unmodified**, as `neuron_memory` does for the dense case. Key difference: between the two modes only the *X addressing* changes. In Type #1 the input is contiguous (`x_base + i`); in Type #2 it is a gather (`act_buf[src_id]`). The arithmetic core is untouched.



The outputs are the **last n\_out** ids: in a DAG with the  $src\_id < out\_id$  ordering the output neurons (sinks, not reused as sources) naturally end up with the highest ids. At the end `graph_engine` copies these `n_out` bytes into a PSRAM region at `out_base`, which the host reads back with `READ_RAM`.

## 7.6 Type #2 opcodes and registers

The type is selected with a new opcode; `RUN_NETWORK` dispatches on the `net_type` register (details in ch. 8).

| Opcode / sel       | Name              | Function                                                                                             |
|--------------------|-------------------|------------------------------------------------------------------------------------------------------|
| 0x11               | SET_NET_TYPE      | <code>type(1B): 0x01=dense (#1), 0x02=graph (#2).</code><br>Default after <code>RESET</code> =dense. |
| SET_BASE<br>sel 9  | num_neurons_graph | Number of graph neurons (uint16).                                                                    |
| SET_BASE<br>sel 10 | n_out             | Number of output ids (uint16).                                                                       |

### Zero regression on Type #1

With `net_type=dense` (the default value after `RESET`) the #1 path is bit-identical to before: `RUN_NETWORK` keeps its `num_layers(1B)` payload and the framing of the existing opcodes does not change.

## 7.7 Occupancy (Type #2 enabled)

Yosys synthesis of the full `spi_neuron_top` system with Type #2 enabled (`PARALLEL=2`):

| Resource           | Use                                  |
|--------------------|--------------------------------------|
| DP16KD (block RAM) | 2 (activation buffer)                |
| MULT18X18D (DSP)   | 4 (2 neuron_memory + 2 graph_engine) |

|                      |      |
|----------------------|------|
| LUT4                 | 2619 |
| TRELLIS_FF           | 2467 |
| \$_TBUF_ (PSRAM bus) | 16   |

The device (108 DP16KD, 72 DSP,  $\approx 44\text{k}$  LUT/FF) stays well below saturation: Type #2 adds a complete mode at a contained resource cost. LUT4/TRELLIS\_FF grew from an earlier measurement (2367/2406) because of the PSRAM page mode added to the controller (ch. 5, § 5.5) — under 6% utilization, no practical impact.

## 7.8 Gather bandwidth (measured)

The per-edge gather cost was **isolated** by building two structurally-identical graphs with different edge counts and differencing the cycles: the subtraction cancels the fixed per-neuron overhead and leaves the edge cost alone.

### Per-edge cost

**37.53 cycles/edge** with PSRAM page mode enabled (ch. 5, § 5.5) — **53.25 cycles/edge** without it (pre-page-mode baseline, consistent with theory: 4 bytes/edge  $\times \approx 13$  cycles/byte over async PSRAM  $\approx 52$ ). At 80 MHz:  $\approx 2.13$  M edges/s ( $\approx 8.5$  MB/s, +42% vs. baseline); at the real 16 MHz clock:  $\approx 426$  k edges/s ( $\approx 1.71$  MB/s).

Page-mode read (roadmap G7, ch. 14) has been implemented and measured: the gather's sequential access benefits directly, cutting the per-edge cost by 29.5% (53.25 $\rightarrow$ 37.53 cycles/edge). Each edge still pays `int8_memory_access`'s byte-granular access (4 bytes/edge); page mode reduces the cost of each sequential byte, not the number of accesses.

## 7.9 netasm host assembler

Readable network configuration needs no dedicated FPGA logic: a pseudo-assembly is compiled *on the host* (`tools/netasm/`) into the exact bytes of the tables and edges, then loaded with `WRITE_RAM`. The assembler validates at compile time (`src_id < out_id`, `N_TOTAL` bounds, padding to `PARALLEL`), complementing the runtime guard.

**Listing 7.1.** Pseudo-assembly example (graph)

```

1 NET graph
2 INPUTS 4                ; ids 0..3
3 NEURON n4 relu bias=2
4   CONN 0 w=5
5   CONN 1 w=-3
6 NEURON n5 none bias=0
7   CONN n4 w=2            ; symbolic reference to n4's output
8   CONN 2 w=7
9 OUTPUT n5
10 END

```

## CHAPTER 8

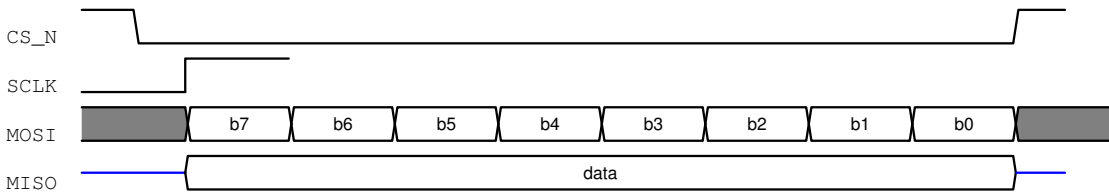
# SPI host interface

### 8.1 Physical layer

The FPGA is always an SPI **slave**. The v1 protocol uses SPI **Mode 0** (CPOL=0, CPHA=0), MSB-first, single-SPI. One command per low-CS period; byte 0 of each transaction is the opcode. Multi-byte fields are big-endian.

#### Mode 0 sampling

`mosi` is sampled on the **rising** edge of `sclk`; `miso` is driven on the **falling** edge (stable before the master's next sampling). `spi_slave` synchronizes `sclk/mosi/cs_n` with a double flip-flop (3-stage CDC) before every edge detection.



*Framing of one byte: CS falls, 8 SCLK pulses, MSB first; MISO in tri-state outside a transaction.*

#### tx\_byte\_req contract

`tx_byte_req` is a *prefetch hint*, not a “byte consumed” event: a consumer must advance its pointers (RAM address, response byte index) on `rx_valid`, which pulses exactly once per real byte transferred.

### 8.2 Framing and explicit length

The length of RAM transfers is **explicit**, not delimited by the CS edge: `WRITE_RAM/READ_RAM` carry a 2-byte length field, so the SPI controller only needs a byte counter. Byte addresses are 23-bit, carried in a 3-byte field with the most significant bit reserved to 0.

### 8.3 Opcode table

| Op   | Name      | Payload<br>(host→FPGA) | Response  | Function                                                        |
|------|-----------|------------------------|-----------|-----------------------------------------------------------------|
| 0x00 | NOP       | —                      | —         | No operation<br>(idle/dummy clocking).                          |
| 0x01 | WRITE_RAM | addr(3B)+len(2B)+data  | —         | Writes a block into<br>PSRAM (X, weights,<br>bias, parameters). |
| 0x02 | READ_RAM  | addr(3B)+len(2B)       | len bytes | Reads a block back<br>from PSRAM.                               |

| Op   | Name             | Payload                               | Response        | Function                                                                                                                                                           |
|------|------------------|---------------------------------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0F | RESET            | —                                     | —               | Synchronous reset of the engine and clearing of the STATUS latch; does not erase PSRAM.                                                                            |
| 0x10 | SET_BASE         | sel(1B)+addr(3B)                      | —               | Sets the bases/registers (see §8.4).                                                                                                                               |
| 0x11 | SET_NET_TYP      | type(1B)                              | —               | Network type:<br>0x01=dense (#1),<br>0x02=graph (#2).<br>Default after RESET=dense.                                                                                |
| 0x20 | START            | —                                     | —               | Starts <code>neuron_memory</code> (single-layer path); ignored if busy.                                                                                            |
| 0x21 | STATUS           | —                                     | 1 byte          | bit0=busy (live),<br>bit1=done (sticky, clear-on-read),<br>bit2=err (graph guard),<br>bit3=flash_err (sticky, clear-on-read),<br>bit4=flash_busy (live); bit7:5=0. |
| 0x22 | READ_OUTPUT      | —                                     | N_NEURONS bytes | <code>y_bus</code> neuron-major (byte 0 = neuron 0); dense path only (Type #1).                                                                                    |
| 0x23 | RUN_NETWORK      | num_layers(1B)                        | —               | Starts execution: dispatches on <code>net_type</code> to <code>layer_sequencer</code> (#1) or <code>graph_engine</code> (#2); ignored if busy.                     |
| 0x30 | READ_CONFIG      | —                                     | 11 bytes        | Hardware configuration record (§8.7).                                                                                                                              |
| 0x40 | FLASH_READ_BLOCK | flash_addr(3B)+psram_addr(3B)+len(3B) | —               | Raw flash→PSRAM read, bypasses the catalog.                                                                                                                        |
| 0x41 | FLASH_WRITE      | psram_addr(3B)+flash_addr(3B)+len(3B) | —               | Raw PSRAM→flash write (internal erase-before-write + ≤256B Page Program loop + WIP poll, transparent to the host), bypasses the catalog.                           |

| Op   | Name           | Payload                                 | Response | Function                                                                                                                                                                        |
|------|----------------|-----------------------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x42 | FLASH_ERASE    | sector_addr(3B)                         | —        | Standalone 4 KB sector erase (must be sector-aligned), bypasses the catalog.                                                                                                    |
| 0x43 | CAT_READ       | —                                       | —        | Reloads the 16-slot catalog (on-chip registers) from the flash's reserved sector.                                                                                               |
| 0x44 | CAT_WRITE_SLOT | slot_id(1B)+offset(3B)+len(3B)+type(1B) | —        | Registers/updates the slot's (offset, length, type) in the on-chip catalog and persists it to flash; marks the slot <i>invalid</i> until <a href="#">SAVE_SLOT</a> confirms it. |
| 0x45 | LOAD_SLOT      | slot_id(1B)+psram_addr(3B)              | —        | Flash→PSRAM for the slot (offset/length from the catalog), verifies the CRC32 live; <code>STATUS.flash_err</code> if the slot is invalid or the CRC does not match.             |
| 0x46 | SAVE_SLOT      | slot_id(1B)+psram_addr(3B)+len(3B)      | —        | PSRAM→flash at the slot's already-registered offset, computes the CRC32 live; on success updates and persists the catalog entry (length, CRC, valid=1).                         |
| 0x47 | CAT_INSPECT    | slot_id(1B)                             | 16 bytes | Synchronous read of an already-loaded catalog entry: off-set[3]+len[3]+type[1]+valid[1]+CRC32[4]-MSB-first.                                                                     |

All flash opcodes are *fire-and-forget*: the host polls [STATUS](#) (bit4= `flash_busy`, bit3=`flash_err`) or the `irq_n/data_ready_n` pins for the outcome, except [CAT\\_INSPECT](#), which responds synchronously.

The 8 flash opcodes (0x40–0x47) are described in full, with design rationale and measured real latencies, in §8.8 below.

## 8.4 SET\_BASE selectors

| sel | Register  | Use                   |
|-----|-----------|-----------------------|
| 0   | x_base    | Input base <i>X</i> . |
| 1   | w_base    | Weight base.          |
| 2   | bias_addr | Bias base.            |

|    |                 |                                                   |
|----|-----------------|---------------------------------------------------|
| 3  | table_base      | Descriptor table base (multi-layer).              |
| 4  | buf_a_base      | Ping-pong buffer A.                               |
| 5  | buf_b_base      | Ping-pong buffer B.                               |
| 6  | activation      | Activation (low 2 bits) — single-layer path only. |
| 7  | n_inputs_real   | Runtime input width (16-bit BE) — single-layer.   |
| 8  | n_neurons_real  | Runtime neuron width (16-bit BE) — single-layer.  |
| 9  | num_neurons_gra | Number of graph neurons (16-bit BE) — Type #2.    |
| 10 | n_out           | Number of output ids (16-bit BE) — Type #2.       |

Selectors 6–8 concern only the single-layer/manual path; with `RUN_NETWORK` the equivalent values are read per-layer from the descriptor table.

#### “real=0” edge cases fixed (2026-09-04)

The re-certification campaign ([docs/validation/bugs.md](#)) found that several runtime values equal to zero were unguarded, with outcomes ranging from a silently ignored limit to a hang or arbitrary-address PSRAM writes. All five cases below are now safe no-ops, independently verified:

- `n_inputs_real=0` (selector 7): completes in 1 cycle with  $y = \text{activation}(\text{bias})$  (BUG-003).
- `n_neurons_real=0` (selector 8): completes without performing any per-neuron computation, far faster than a full-width run (BUG-004).
- `num_neurons_graph=0` (selector 9): completes immediately after the input copy, without ever entering the descriptor loop (BUG-006).
- `RUN_NETWORK` with `num_layers=0` (dense path): an immediate no-op — **before the fix it executed 256 fabricated layers, reading arbitrary PSRAM data as descriptors** (BUG-005, CRITICAL, see §8.9.2 below).
- `SET_NET_TYPE` received while a run is in progress: now silently rejected (no effect, no SPI error) instead of remapping the arbiter’s multiplexer mid-execution — **before the fix it caused a permanent hang of the in-progress engine** (BUG-007, CRITICAL).

Details, evidence, and per-fix verification are in [docs/validation/bugs.md](#).

## 8.5 STATUS.done sticky / clear-on-read

In `neuron_memory` the `done` signal is a single-cycle pulse. A host polling over SPI (much slower than the FPGA clock) would almost certainly miss a raw one-cycle pulse. The SPI register bank therefore latches `done` into a sticky bit on the pulse and clears it when the host reads `STATUS` (or `RESET`). The `busy` bit is instead held at level for the whole computation and is read live.

#### Race corrected (2026-09-02)

A real race in the sticky mechanism (present since Phase 4) was corrected by latching a `status_snapshot` on acceptance of the `STATUS` opcode and conditioning the clearing of the sticky bit on `status_snapshot[1]` (it clears only if the byte actually transmitted showed `done=1`). A `done` that arrives too late for a snapshot is reported on the next poll instead of being lost.

## 8.6 Host attention pins (data\_ready\_n, irq\_n)

Besides `STATUS` polling, the top-level exposes two active-low physical pins (bank 7, ch. 12) that mirror the sticky bits without an SPI transaction, handy for driving a host GPIO/IRQ:

- `data_ready_n = ~STATUS.done` (sticky): low when a result is ready to read, returns high on the `STATUS` read (clear-on-read).
- `irq_n = ~STATUS.err` (graph guard): low when `graph_engine`’s load-time guard has tripped.

It is **not** clear-on-read: it clears only on `RESET` or a fresh graph start, so an error is not missed between polls.

These are additive ports: they touch neither the existing opcodes nor the registers.

#### flash\_err has no dedicated pin

`STATUS.flash_err` (bit3) is reported **only** in the `STATUS` byte, by design: reusing `irq_n` would have conflated it with graph-guard errors (two independent error domains on one pin), while a flash operation is always host-initiated with an opcode just issued, so polling `STATUS` right after — already implicit in the “fire-and-forget, then poll `STATUS/data_ready_n`” convention — is already a natural fit, no extra async pin needed. `data_ready_n`, on the other hand, *also clears at the end of a flash operation*: it mirrors `STATUS.done` (bit1), which now latches on a completed flash op too, not only on `RUN_NETWORK/START`.

## 8.7 READ\_CONFIG

Fixed **11-byte** payload: it lets a single host firmware work with different bitstreams without recompiling. The `N_INPUTS/N_NEURONS` values report the build *maximum* (the ceiling), not necessarily the currently loaded network.

| Byte | Field                             | Source                                |
|------|-----------------------------------|---------------------------------------|
| 0    | <code>ADDR_WIDTH</code> (bit)     | <code>neuron_memory.ADDR_WIDTH</code> |
| 1–2  | <code>N_INPUTS</code> (16-bit BE) | build maximum                         |
| 3    | <code>N_NEURONS</code>            | build maximum                         |
| 4    | <code>PARALLEL</code>             | build parameter                       |
| 5    | <code>DATA_WIDTH</code> (bit)     | build parameter                       |
| 6–7  | protocol version (BE)             | 0x0001                                |
| 8–9  | <code>N_TOTAL</code> (16-bit BE)  | max graph signals (Type #2)           |
| 10   | capability flag                   | bit0=GRAPH_SUPPORTED=1                |

## 8.8 Flash subsystem (opcodes 0x40–0x47, completed 2026-09-04)

The FPGA has **exclusive** access to the onboard boot/persistence flash (Winbond W25Q128JV, 16 MB SPI NOR, ch. 12 §6/§7) through a dedicated, physically separate SPI master (`rtl/spi_flash_master.v`), never through direct host access to the flash pins. This is **not** a filesystem: a fixed-size catalog (16 slots, `rtl/flash_slot_manager.v`) maps `slot_id` → (offset, length, type, valid, CRC32) in a reserved flash sector (sector 0) — no dynamic allocation, no garbage collection.

#### Layering (each level independently testable)

- `rtl/spi_flash_master.v` — raw SPI master toward the flash chip (RDID/READ/WREN/PP/SE/RDSR-1). Fully independent 4-wire bus (`sclk/mosi/miso/cs_n`, all ordinary GPIO — Phase F7, 2026-09-04): an earlier version reused the boot `CCLK` pad via the ECP5 `USRMCLK` primitive to save one pin, dropped because it made the “exclusive flash bus” claim electrically misleading (`SCLK` still depended on the same pad as the config engine) and carried an unresolved verification gap (`USRMCLKTS` timing never checked against the primary Lattice `sysCONFIG` Usage Guide).
- `rtl/flash_copy_engine.v` — block-streaming engine on top: flash → PSRAM (`DIR_LOAD`), PSRAM → flash with internal erase-before-write + ≤256B Page Program loop + WIP polling (`DIR_SAVE`), standalone sector erase (`DIR_ERASE`). A low-priority master (Port

D) on `rtl/mem_arbiter.v`: flash operations are ms-scale and never block inference.

- `rtl/flash_slot_manager.v` — the slot catalog on top of that, plus a CRC32 (`rtl/crc32.v`, IEEE 802.3/zlib) computed live over the real byte stream during `LOAD_SLOT/SAVE_SLOT`, so a corrupted or partially-written slot (e.g. power lost mid-erase) is detected even when the underlying flash operation itself reported success.

### Sector alignment is mandatory

`SAVE_SLOT` (and the raw `FLASH_WRITE_BLOCK/FLASH_ERASE`) require the target flash address to be 4 KB-sector-aligned — rejected as an error otherwise, rather than a silent partial-sector read-modify-erase-write (no scratch buffer large enough exists for that, and every real `SAVE_SLOT` already writes a whole, sector-aligned slot by construction).

Full rationale, every datasheet citation, every adversarial test (CRC mismatch, never-saved slot, page-boundary crossing, simulated power loss, arbiter contention), and the two real bugs found and fixed during bring-up (one pre-existing in `psram_controller.v`, one in the new arbiter request handshake) are in `WORKLOG.md` (Phases F1-F6 entries) and `docs/FPGA-Neural-Flash-Subsystem-Verification.md` (per-module coverage summary, not repeated here).

| Operation                              | Measured real latency                                                                   |
|----------------------------------------|-----------------------------------------------------------------------------------------|
| ERASE (4 KB sector)                    | ≈400 ms (dominated by the flash chip's own internal tSE, independent of the host clock) |
| SAVE (256 B page, incl. its own erase) | ≈403 ms (same, tSE+tPP)                                                                 |
| LOAD (4096 B)                          | 1.74 ms (2.35 MB/s) @80 MHz; 8.71 ms (0.47 MB/s) @16 MHz (purely SPI-clock-bound)       |

Full measurement methodology in `docs/FPGA-Neural-Flash-Subsystem-Verification.md`.

## 8.9 Session sequences

### 8.9.1 Single-layer path

Listing 8.1. Single-layer session

```

1 RESET          -> 0x0F
2 READ_CONFIG    -> 0x30      (host learns N_INPUTS/N_NEURONS/...)
3 WRITE_RAM (weights) -> 0x01 ...
4 WRITE_RAM (bias)   -> 0x01 ...
5 SET_BASE (X/W/BIAS) -> 0x10 x3
6 WRITE_RAM (input X) -> 0x01 ...
7 START          -> 0x20
8 poll STATUS     -> 0x21      (until done=1; cleared by this read)
9 READ_OUTPUT     -> 0x22
```

### 8.9.2 Multi-layer path (RUN\_NETWORK)

Listing 8.2. Multi-layer session

```

1 WRITE_RAM (descriptor table) -> 0x01 ...
2 WRITE_RAM (weights/bias per layer, X L0) -> 0x01 ...
3 SET_BASE (X/TABLE/BUF_A/BUF_B) -> 0x10 x4
4 RUN_NETWORK(num_layers) -> 0x23 <num_layers>
5 poll STATUS -> 0x21 (until done=1)
6 READ_OUTPUT -> 0x22 (y_bus of the final layer)
```

**Out of scope for v1**

Dual SPI and CRC/checksum on host transfers (SPI assumed reliable on a board trace — not to be confused with the flash catalog's CRC32, §8.8, which protects a different domain: flash↔PSRAM persistence, not the host SPI link).

**WRITE\_RAM/READ\_RAM have no backpressure to the host — a real risk, not a theoretical one**

Every received/produced byte must be fully processed by `spi_engine` before the next SCLK-driven byte boundary arrives — reasonable for the initial bulk-loading of weights/inputs, not a real-time path. The concrete risk: if a host issues `WRITE_RAM/ READ_RAM` before `psram_controller.v`'s power-up sequence has completed ( $\sim 150\ \mu\text{s}$  after reset, `STATE_INIT+STATE_CR_INIT`), `spi_engine` stalls waiting for the very first PSRAM access to complete, while the host — not slowed by any handshake — keeps clocking bytes. Bytes received during that stall are **silently dropped**, with no error and no hang: just wrong data in PSRAM. Found during the flash-subsystem work (`WORKLOG.md`, Phase F5) via a minimal `WRITE_RAM`-only reproduction with no flash opcodes involved at all: it is a general hazard for any host, not specific to the flash opcodes. **Current mitigation: a host must wait for PSRAM power-up (or otherwise ensure the FPGA has been out of reset for  $>150\ \mu\text{s}$ ) before its first `WRITE_RAM/READ_RAM`.** Not fixed at the protocol level (would need real backpressure, a larger change) — declared here as an open risk, not silently worked around.

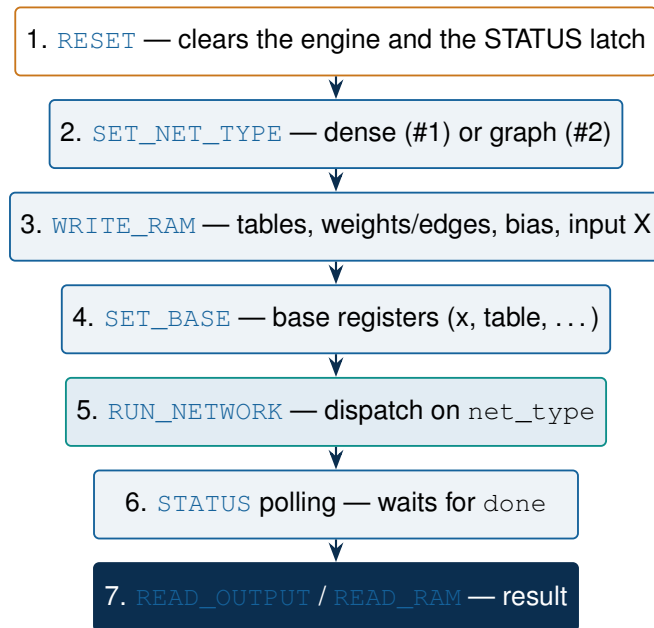
## CHAPTER 9

# Neural network programming

This chapter is the practical guide to encoding a network for FPGA-Neural: how it is laid out in memory, which registers are set and how it is started, for both topologies. It assumes the SPI opcodes (ch. 8) and the descriptor formats (ch. 6, 7).

### 9.1 General flow

Whatever the type, the cycle is the same: the host *builds the data structures in RAM*, sets the *base registers*, declares the *network type*, *starts* and *reads back* the result.



### 9.2 Registers and opcodes involved

All base values are set with `SET_BASE sel(1B)+addr(3B)`. Selectors:

| sel | Register          | Type #1 | Type #2 | Use                                     |
|-----|-------------------|---------|---------|-----------------------------------------|
| 0   | x_base            | ✓       | ✓       | Input base X.                           |
| 3   | table_base        | ✓       | ✓       | Descriptor table.                       |
| 4   | buf_a_base        | ✓       | ✓*      | Ping-pong A (#1) / out_base reuse (#2). |
| 5   | buf_b_base        | ✓       | —       | Ping-pong B (#1).                       |
| 9   | num_neurons_graph | —       | ✓       | Number of graph neurons.                |
| 10  | n_out             | —       | ✓       | Number of output ids.                   |

\* In Type #2 the ping-pong buffers are unused: selector 4 is reused as `out_base` (region into which outputs are copied). Selectors 1/2/6/7/8 concern only the manual single-layer path (`START`), not `RUN_NETWORK`.

For Type #1, the *per-layer* `w_base/bias_addr` are **not** set with `SET_BASE`: they are fields of the descriptor table. `SET_NET_TYPE` defaults to *dense* after `RESET`, so a #1 network works even without issuing it.

## 9.3 Type #1 — dense network

### 9.3.1 Memory layout

| Structure           | Format                                                                                    |
|---------------------|-------------------------------------------------------------------------------------------|
| Input $X$           | $n\_inputs\_real$ INT8 bytes at $x\_base$ .                                               |
| Weights (per layer) | Neuron-major: neuron $k$ at $w\_base + k*n\_inputs\_real$ , $n\_neurons*n\_inputs$ bytes. |
| Bias (per layer)    | One INT8 byte per neuron at $bias\_addr$ .                                                |
| Descriptor table    | $num\_layers$ 11-byte entries at $table\_base$ .                                          |
| Buffers A/B         | Ping-pong intermediate outputs.                                                           |

Descriptor (11 bytes, MSB-first):  $w\_base(3) \mid bias\_addr(3) \mid activation(1) \mid n\_inputs\_real(2) \mid n\_neurons\_real(2)$ .

### 9.3.2 Worked example: a $4 \rightarrow 4 \rightarrow 2$ network

Layer 0: 4 inputs, 4 neurons, ReLU. Layer 1: 4 inputs, 2 neurons, linear (PARALLEL=2, so each  $n\_inputs\_real$  is a multiple of 2). Chosen addresses:  $table\_base=0x000000$ ,  $x\_base=0x001000$ , L0 weights/bias at  $0x002000/0x002100$ , L1 at  $0x002200/0x002300$ , buffers at  $0x003000/0x003100$ .

**Listing 9.1.** Dense descriptor table (22 bytes)

```

1 Layer 0:  00 20 00 | 00 21 00 | 01 | 00 04 | 00 04
2           w_base  bias_addr ReLU  n_in=4  n_neu=4
3 Layer 1:  00 22 00 | 00 23 00 | 00 | 00 04 | 00 02
4           w_base  bias_addr NONE  n_in=4  n_neu=2

```

**Listing 9.2.** SPI session (dense)

```

1 0x0F                                RESET
2 0x11 01                             SET_NET_TYPE = dense
3 0x01 000000 0016 <22-byte table>    WRITE_RAM table
4 0x01 002000 0010 <16-byte L0 wts>    WRITE_RAM L0 weights (neuron-major)
5 0x01 002100 0004 <4-byte L0 bias>
6 0x01 002200 0008 <8-byte L1 wts>
7 0x01 002300 0002 <2-byte L1 bias>
8 0x01 001000 0004 <x0 x1 x2 x3>       WRITE_RAM input X
9 0x10 00 001000                      SET_BASE x_base
10 0x10 03 000000                     SET_BASE table_base
11 0x10 04 003000                     SET_BASE buf_a
12 0x10 05 003100                     SET_BASE buf_b
13 0x23 02                            RUN_NETWORK num_layers=2
14 0x21 ...                           poll STATUS until done=1
15 0x22                               READ_OUTPUT -> 2 bytes (final layer)

```

### 9.3.3 Host pseudocode (dense)

**Listing 9.3.** Encoding and loading a dense network

```

1 def load_dense(layers, X):           # layers in execution order
2     spi(RESET); spi(SET_NET_TYPE, DENSE)
3     table = b""
4     for L in layers:                  # L: weights[n][k], bias[n], act, n_in, n_out
5         assert L.n_in % PARALLEL == 0
6         w = alloc(L.weights_neuron_major) # k slow, input fast
7         b = alloc(L.bias)
8         table += u24(w)+u24(b)+u8(L.act)+u16(L.n_in)+u16(L.n_out)
9     write_ram(TABLE_BASE, table)
10    write_ram(X_BASE, X)
11    set_base(0, X_BASE); set_base(3, TABLE_BASE)

```

```

12     set_base(4, BUF_A); set_base(5, BUF_B)
13     spi(RUN_NETWORK, len(layers))
14     wait_status_done()
15     return read_output(layers[-1].n_out)

```

## 9.4 Type #2 — graph network

### 9.4.1 Memory layout

| Structure        | Format                                                                                                    |
|------------------|-----------------------------------------------------------------------------------------------------------|
| Input $X$        | $N_{in}$ bytes at $x\_base$ ; copied into $act\_buf[0..N_{in}-1]$ at start.                               |
| Descriptor table | $num\_neurons\_graph$ 11-byte entries at $table\_base$ , in ascending $out\_id$ order.                    |
| Edge blocks      | Per neuron: $n\_conn$ 4-byte edges at $conn\_ptr$ , padded to a multiple of PARALLEL (zero-weight edges). |
| Outputs          | $n\_out$ bytes written to $out\_base$ (=selector 4).                                                      |

Graph descriptor (11 bytes):  $conn\_ptr(3) | n\_conn(2) | out\_id(2) | activation(1) | bias(1) | reserved(2)$ . Edge (4 bytes):  $src\_id(2) | weight(1) | reserved(1)$ . Rule:  $src\_id < out\_id$  (feed-forward DAG).

### 9.4.2 Worked example

4 inputs (ids 0–3). Neuron  $n4$  ( $out\_id=4$ , ReLU,  $bias=2$ ) connected to ids 0 and 1; neuron  $n5$  ( $out\_id=5$ , linear,  $bias=0$ ) connected to  $n4$  (id 4) and id 2; output =  $n5$  ( $n\_out=1$ ). PARALLEL=2, both have 2 connections (no padding). Addresses:  $table\_base=0x000000$ , edges at  $0x000100$ ,  $x\_base=0x001000$ ,  $out\_base=0x002000$ .

**Listing 9.4.** Graph descriptors + edges

```

1 Descriptors (at 0x000000, 22 bytes):
2   n4: 00 01 00 | 00 02 | 00 04 | 01 | 02 | 00 00
3       conn_ptr n_conn out_id ReLU bias rsv
4   n5: 00 01 08 | 00 02 | 00 05 | 00 | 00 | 00 00
5       conn_ptr n_conn out_id NONE bias rsv
6
7 Edge blocks (at 0x000100, 4 bytes/edge: src_id, weight, rsv):
8   n4 @0x000100: 00 00 05 00 (src=0, w=+5)
9                  00 01 FD 00 (src=1, w=-3) ; -3 = 0xFD
10  n5 @0x000108: 00 04 02 00 (src=4, w=+2) ; id4 = n4's output
11                  00 02 07 00 (src=2, w=+7)

```

**Listing 9.5.** SPI session (graph)

```

1 0x0F RESET
2 0x11 02 SET_NET_TYPE = graph
3 0x01 000000 0016 <22-byte table> WRITE_RAM descriptors
4 0x01 000100 0010 <16-byte edges> WRITE_RAM edge blocks
5 0x01 001000 0004 <x0 x1 x2 x3> WRITE_RAM input X
6 0x10 00 001000 SET_BASE x_base
7 0x10 03 000000 SET_BASE table_base
8 0x10 04 002000 SET_BASE out_base (sel 4 reuse)
9 0x10 09 000002 SET_BASE num_neurons_graph = 2
10 0x10 0A 000001 SET_BASE n_out = 1
11 0x23 00 RUN_NETWORK (dispatch to graph_engine)
12 0x21 ... poll STATUS (bit2=err if src_id>=out_id)
13 0x02 002000 0001 READ_RAM out_base -> 1 byte (n5 output)

```

### 9.4.3 Host pseudocode (graph)

**Listing 9.6.** Encoding and loading a graph

```

1 def load_graph(neurons, X, n_out): # neurons sorted by ascending out_id
2     spi(RESET); spi(SET_NET_TYPE, GRAPH)
3     edges = b""; table = b""
4     for N in neurons:
5         for (src,_) in N.conns:
6             assert src < N.out_id and src < N_TOTAL # DAG rule
7             conn_ptr = EDGE_BASE + len(edges)
8             padded = pad(N.conns, PARALLEL, fill=(0,0)) # zero-weight edges
9             for (src,w) in padded:
10                 edges += u16(src)+i8(w)+u8(0)
11                 table += u24(conn_ptr)+u16(len(N.conns))+u16(N.out_id) \
12                     + u8(N.act)+i8(N.bias)+u16(0)
13     write_ram(TABLE_BASE, table); write_ram(EDGE_BASE, edges)
14     write_ram(X_BASE, X)
15     set_base(0, X_BASE); set_base(3, TABLE_BASE); set_base(4, OUT_BASE)
16     set_base(9, len(neurons)); set_base(10, n_out)
17     spi(RUN_NETWORK, 0) # payload ignored in graph
18     wait_status_done()
19     return read_ram(OUT_BASE, n_out)

```

**9.4.4 netasm pseudo-assembly**

The readable description is compiled by the host assembler (`tools/netasm/`) into exactly the table and edge bytes above. Example equivalent to the worked graph:

**Listing 9.7.** netasm: source and generated bytes

```

1 ; --- source ---
2 NET graph
3 INPUTS 4 ; ids 0..3
4 NEURON n4 relu bias=2
5     CONN 0 w=5
6     CONN 1 w=-3
7 NEURON n5 none bias=0
8     CONN n4 w=2 ; symbolic reference -> id 4
9     CONN 2 w=7
10 OUTPUT n5
11 END
12
13 ; --- the assembler emits ---
14 ; assigned ids: n4=4, n5=5 (guarantees src_id < out_id)
15 ; descriptors: 00 01 00 00 02 00 04 01 02 00 00
16 ;               00 01 08 00 02 00 05 00 00 00 00
17 ; edges:       00 00 05 00 00 01 FD 00 (n4)
18 ;               00 04 02 00 00 02 07 00 (n5)
19 ; registers:   table_base, x_base, out_base, num_neurons=2, n_out=1
20 ; compile-time checks: src_id<out_id, N_TOTAL, padding to PARALLEL

```

**Why two encoding levels**

The host pseudocode and `netasm` produce the *same bytes*. The former is useful when the network is generated at runtime (e.g. trained weights); the latter when the topology is hand-written or version-controlled as source. In both cases the FPGA receives only tables and data via `WRITE_RAM`: no on-board interpreter.

## CHAPTER 10

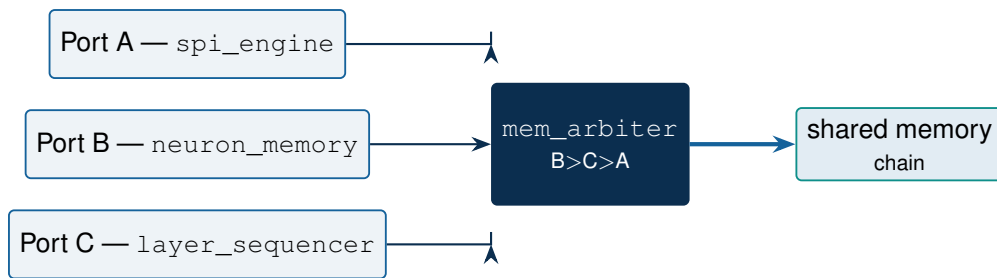
# Arbitration and top-level integration

### 10.1 mem\_arbiter — three-port arbiter

A single byte-level memory master (which feeds the shared chain `int8_memory_access` → `memory_interface` → `psram_controller`) is arbitrated among three requesters:

| Port | Master                       | Accesses                                         |
|------|------------------------------|--------------------------------------------------|
| A    | <code>spi_engine</code>      | <code>WRITE_RAM</code> / <code>READ_RAM</code> . |
| B    | <code>neuron_memory</code>   | X/W/bias reads during an execution.              |
| C    | <code>layer_sequencer</code> | Descriptor reads + buffer writes between layers. |

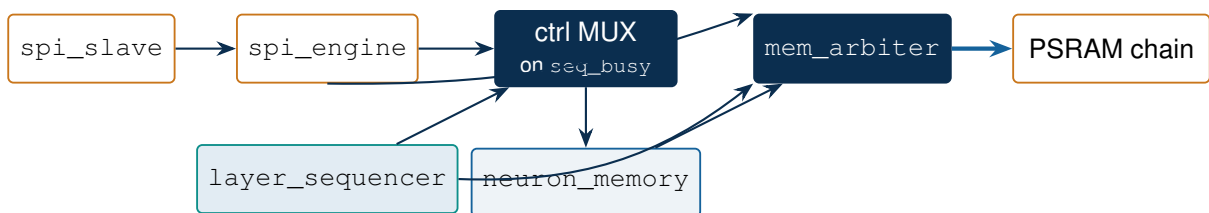
Fixed priority **B > C > A**: an inference in progress is more critical than the sequencer's book-keeping, which in turn is more critical than a manual SPI access that has just arrived. In normal operation B and C are anyway temporally disjoint (`neuron_memory` requests only during an execution, `layer_sequencer` only in the pauses between layers), so the priority matters mostly for the corner case of a manual `WRITE_RAM/READ_RAM` arriving during a multi-layer execution.



Once access is granted, the arbiter retains ownership until the single transaction's `m_ready` pulse, then releases: all three masters emit `req` as a clean one-cycle pulse, so a queue-less grant-and-forward design suffices.

### 10.2 spi\_neuron\_top — full integration

The top-level connects SPI (`spi_slave`+`spi_engine`), the arbiter, the sequencer, `neuron_memory` and the PSRAM chain. The reset of `neuron_memory` is the OR of the global reset with the soft-reset pulse of the `RESET` opcode, so the host can recover the engine over SPI without a physical reset (the RAM stays intact).



The multiplexer switches the control lines of `neuron_memory` between the sequencer (while `seq_busy` is high) and the direct path of `spi_engine` (legacy single-layer mode), returning the engine to the direct path at the end of the sequence.

**End-to-end verification**

`spi_neuron_top` is verified in simulation with real PSRAM (`psram_model.v`, no mock): RESET/READ\_CONFIG/WRITE\_RAM/READ\_RAM/SET\_BASE/ START/STATUS/READ\_OUTPUT and `RUN_NETWORK` are exercised purely over simulated SPI (ch. 11).

## CHAPTER 11

# ECP5 implementation and characterization

### 11.1 Flow and verification

The project is verified on two complementary planes: functional **simulation** with Icarus Verilog (signed algebra, products, accumulation, groups, bias, ReLU, saturation, busy/done signals) and real **implementation** with Yosys (synthesis) + nextpnr-ecp5 (place&route, timing) + Project Trellis (ecppack).

| Verification stage    | Outcome | Covers                          |
|-----------------------|---------|---------------------------------|
| Functional RTL        | PASS    | datapath correctness            |
| Parametric simulation | PASS    | configuration sweep             |
| ECP5 synthesis        | PASS    | synthesizability, mapping       |
| Placement / Routing   | PASS    | LUT/FF/DSP, timing              |
| Bitstream (ecppack)   | PASS    | full flow, 0 errors (P2 and P8) |

#### End-to-end toolchain through the bitstream

The full flow RTL → Yosys → nextpnr-ecp5 → ecppack produces a valid bitstream for P2 and P8, **0 errors at every stage**. Header verified byte-by-byte: Part : LFE5U-45F-8CABGA381, the target's real part number, not a placeholder. Only *generation* is verified: no physical-hardware test in this session.

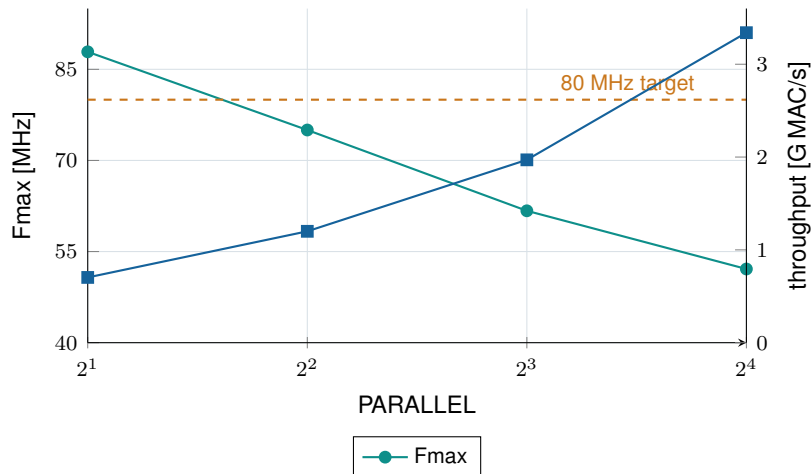
### 11.2 Datapath benchmark (256×4)

Configuration: INT8/INT32, N\_INPUTS=256, N\_NEURONS=4, variable PARALLEL, 80 MHz target, device LFE5U-45F-8BG381C (-8). The test buses are generated *inside* the benchmark wrapper so as not to expose thousands of I/Os; the top-level exposes only clk/rst/start/y\_bus/busy/done.

| PAR | tot MAC | DSP   | Fmax  | Tcrit | 80 MHz | LUT4  |
|-----|---------|-------|-------|-------|--------|-------|
| 16  | 64      | 64/72 | 52.13 | 19.18 | FAIL   | ≈2531 |
| 8   | 32      | 32/72 | 61.71 | 16.20 | FAIL   | —     |
| 4   | 16      | 16/72 | 75.01 | 13.33 | FAIL   | 804   |
| 2   | 8       | 8/72  | 87.88 | 11.38 | PASS   | 481   |

*Fmax and Tcrit in MHz and ns. Total MACs = PARALLEL×4 neurons.*

### 11.2.1 Fmax and throughput versus parallelism



Fundamental trade-off: as `PARALLEL` grows, `Fmax` drops (deeper routing/tree) but the theoretical throughput rises. The blue line (squares) is the throughput  $\approx \text{MAC/cycle} \times F_{\text{max}}$ .

### 11.2.2 Interpretation

Reducing `PARALLEL` lowers simultaneous MACs, DSPs, adder-tree depth and routing congestion, so `Fmax` rises; but the number of groups increases and hence the latency. Frequency alone is not enough to choose: what matters is the overall throughput  $\approx \text{MAC/cycle} \times \text{frequency}$ .

#### Architectural choices

`PARALLEL=8` is the candidate for the throughput-oriented V1: exactly 32 simultaneous MACs with 4 neurons, DSP at  $\approx 44\%$ , leaving resources for controller, buffers, SPI and future pipelines. `PARALLEL=2` is the frequency-oriented reference: 87.88 MHz, the only one to exceed the 80 MHz target, but it requires 128 groups for a 256-input neuron.

### 11.2.3 Critical path and the 100 MHz limit

The 100 MHz target is not met (best result 87.88 MHz with P2). The limit is *temporal*, not one of occupancy: with P2 the FPGA is barely used (DSP  $\approx 11\%$ , LUT  $\approx 1\%$ ). The critical path runs through weight FF  $\rightarrow$  MULT18X18D  $\rightarrow$  products  $\rightarrow$  adder/carry  $\rightarrow$  acc\_next  $\rightarrow$  ReLU/saturation  $\rightarrow$  output FF. Exceeding 100 MHz will require one or more internal pipelines, not yet necessary to proceed.

## 11.3 Full integrated system

Real synthesis of `spi_neuron_top` (SPI + arbiter + neuron\_memory + graph\_engine + PSRAM chain), speed grade -8. Before timing closure the integrated system missed the 80 MHz target (P2  $\approx 55$  MHz, P8  $\approx 45$  MHz), with a critical path entirely inside `neuron_parallel`.

### 11.3.1 Cause: the saturation/ReLU carry chain

Resource usage is not the cause (device below 10% everywhere). The integrated system's critical path is the **ccu2c carry chain of the saturation/ReLU comparator** in `neuron_parallel.v` — not SPI, arbiter, PSRAM, nor the Type #2 modules. The saturation was written as an arithmetic comparison (`acc > 127, acc < -128`), mapped by the synthesizer onto a 32-bit subtractor with a long carry chain.

### 11.3.2 Timing closure (2026-09-03)

An explicit waiver of the “datapath untouchable” rule for a separate timing-closure task, with the single constraint of **bit-exact equivalence** across the whole regression. Two steps:

- **Step 1 — saturation/ReLU as bit-test.** A signed 32-bit value fits INT8 iff `acc[31:7]` are all equal: an AND/OR reduction over a bit slice instead of a 32-bit carry chain. A correct, bit-exact-verified simplification; real logic gain, but on its own submerged by placement noise.
- **Step 2 — pipeline register** between accumulate and activation (+1 cycle of latency per neuron, absorbed by the `start/done` handshake, transparent to callers). This is the decisive step.

| Config            | Before      | After        | $\Delta$        |
|-------------------|-------------|--------------|-----------------|
| P2, real .lpf     | 54.58       | <b>75.30</b> | +38%            |
| P2, 5-seed sweep  | 55.59       | 73.38–75.55  | robust          |
| P8, unconstrained | 45.47       | <b>60.26</b> | +33%            |
| P8, 5-seed sweep  | 43.15–50.48 | 60.26–68.87  | non-overlapping |

*Fmax in MHz, real place&route (nextpnr-ecp5). Robust across 5 seeds, not attributable to placement luck.*

#### Stop criterion and real margin

80 MHz is not reached (75.30 MHz at P2, 94% of target) but the gain is large and real (+38%/+33%). The next step (the `MULT18X18D` output register, which would touch `mac_unit.v`) was left out: 80 MHz is *headroom* toward the real .lpf, not an operating requirement. With the planned 16 MHz oscillator, even the worst measured number ( $\approx 45$  MHz at P8) has  $2.8\times$  of margin. **Superseded 2026-09-04:** after adding the flash subsystem (ch. 8 §8.8, ch. 14), Fmax for the full system (P2, same real pinout + 3 new flash signals) was 66.68 MHz, critical path still on the same `neuron_parallel` accumulator chain identified above — not a new bottleneck, the difference from 75.30 MHz was placement/routing noise from the added pins/logic. **Updated again the same day (Phase F7):** made the flash SPI bus genuinely independent (dropped the `CCLK/USRMCLK` reuse, added a 4th ordinary `flash_sclk` pin), Fmax re-measured **67.91 MHz** (slight improvement, critical path confirmed still identical). Margin on the 16 MHz oscillator:  $4.2\times$ .

#### Separate future optimization

Independent of timing closure: the `x_mem/w_mem` arrays of `neuron_memory` are still inferred as distributed RAM on LUTs instead of `DP16KD`. Moving them to block RAM would free LUTs and is a Phase 7 candidate — but it was not on the critical path resolved here.

## CHAPTER 12

# Hardware design and signal map

### Pinout status — assigned and verified

A real `.lpf` now exists (`synth/ecp5/spi_neuron_top.lpf`) with the top-level's **57 signals** assigned to concrete CABGA381 balls, **verified by a full 0-error `nextpnr-ecp5 place&route`** (no longer `--lpf-allow-unconstrained`). The balls come from Project Trellis's device database (`iodb.json`, the same `nextpnr` uses) and were independently validated against §4.3.2 of the official Lattice datasheet (per-bank GPIO counts: exact match on 6 of 7 banks, off by 1 ball on bank 3, immaterial since no assigned signal uses it). `TRELLIS_IO`: 57/245 (23%). Current-build Fmax (full system incl. flash subsystem with an independent SPI bus, Phase F7, 2026-09-04) **67.91 MHz**, critical path confirmed still on `neuron_parallel`'s accumulator chain, unchanged from earlier builds (ch. 11). Pin-by-pin summary at the front of the document (pp. 2–3). The boot config-SPI and JTAG balls do not appear here: they are dedicated fixed-function pins with no corresponding RTL port, `nextpnr` never requires them (0 errors), they matter only for the PCB schematic.

### 12.1 Target device

| Parameter   | Value                                         |
|-------------|-----------------------------------------------|
| Device      | Lattice ECP5 LFE5U-45F-8BG381C                |
| Package     | CABGA381 (381 balls)                          |
| Speed grade | –8 (the fastest of the ECP5 family)           |
| Resources   | ≈44k LUT/FF, 72×MULT18X18D, DP16KD block RAM  |
| Usable I/O  | ≈232 balls out of 381 (rest: power/ground/NC) |

### 12.2 Pin budget

The project requires about 60 signals out of ≈232 usable I/Os: ample margin (>170 free pins), so the board is not pin-constrained.

| Function                                                                                                                               | Pins       |
|----------------------------------------------------------------------------------------------------------------------------------------|------------|
| PSRAM (22 address, 16 data, 6 control)                                                                                                 | up to 44   |
| Application SPI ( <code>sclk/mosi/miso/cs_n</code> )                                                                                   | 4          |
| Clock, reset                                                                                                                           | 2          |
| Host attention pins ( <code>irq_n, data_ready_n</code> )                                                                               | 2          |
| Flash runtime SPI bus<br>( <code>flash_sclk/flash_mosi/flash_miso/flash_cs_n</code> , ordinary GPIO, fully independent bus — Phase F7) | 4          |
| JTAG (bring-up / debug, recommended)                                                                                                   | 4          |
| <b>Total</b>                                                                                                                           | <b>≈60</b> |

## 12.3 Signal map (top-level spi\_neuron\_top) — real balls

Real assignment of the top-level's 57 signals, verified by place&route, **an individual ball for every bit** (never a bus range). I/O standard: LVCMOS33 (3.3 V I/O supply). The balls come from the real place&route-verified .lpf. A compact summary of the same table also appears at the front of the document (pp. 2–3).

| Signal                                                                                           | Dir | Ball | Bank | Function                                                                                                                                   |
|--------------------------------------------------------------------------------------------------|-----|------|------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Clock and reset (bank 7, left edge)</i>                                                       |     |      |      |                                                                                                                                            |
| clk                                                                                              | IN  | H5   | 7    | System clock on pad GR_PCLK7_0 (dedicated global clock).                                                                                   |
| rst                                                                                              | IN  | B4   | 7    | Global synchronous reset, active high.                                                                                                     |
| <i>Application SPI (bank 7, opposite the PSRAM bus)</i>                                          |     |      |      |                                                                                                                                            |
| sclk                                                                                             | IN  | B5   | 7    | SPI clock (CPOL=0, CPHA=0).                                                                                                                |
| mosi                                                                                             | IN  | C5   | 7    | Master-Out Slave-In.                                                                                                                       |
| miso                                                                                             | OUT | A3   | 7    | Master-In Slave-Out (driven on the falling edge).                                                                                          |
| cs_n                                                                                             | IN  | B3   | 7    | Active-low chip-select.                                                                                                                    |
| <i>Host attention pins (bank 7, active-low, level)</i>                                           |     |      |      |                                                                                                                                            |
| data_ready_n                                                                                     | OUT | C3   | 7    | Low while a result awaits reading (mirrors STATUS.done, clear on STATUS read).                                                             |
| irq_n                                                                                            | OUT | C4   | 7    | Low if the graph load-time guard has tripped (mirrors STATUS.err); clears only on RESET or a fresh run_start, <i>not</i> on a STATUS read. |
| <i>Flash subsystem — independent SPI bus toward the onboard W25Q128JV (bank 7, Phases F1-F7)</i> |     |      |      |                                                                                                                                            |
| flash_sclk                                                                                       | OUT | E3   | 7    | SPI clock toward the flash — ordinary GPIO, no config primitive involved (Phase F7).                                                       |
| flash_mosi                                                                                       | OUT | D3   | 7    | Master-Out Slave-In toward the flash.                                                                                                      |
| flash_miso                                                                                       | IN  | D5   | 7    | Master-In Slave-Out from the flash.                                                                                                        |
| flash_cs_n                                                                                       | OUT | E4   | 7    | Flash chip-select, active low.                                                                                                             |
| <i>PSRAM address bus psram_a[21:0] — 22 individual balls (bank 2)</i>                            |     |      |      |                                                                                                                                            |
| psram_a[0]                                                                                       | OUT | E16  | 2    | PSRAM A0                                                                                                                                   |
| psram_a[1]                                                                                       | OUT | F16  | 2    | PSRAM A1                                                                                                                                   |
| psram_a[2]                                                                                       | OUT | D18  | 2    | PSRAM A2                                                                                                                                   |
| psram_a[3]                                                                                       | OUT | E17  | 2    | PSRAM A3                                                                                                                                   |
| psram_a[4]                                                                                       | OUT | E18  | 2    | PSRAM A4                                                                                                                                   |
| psram_a[5]                                                                                       | OUT | F18  | 2    | PSRAM A5                                                                                                                                   |
| psram_a[6]                                                                                       | OUT | F17  | 2    | PSRAM A6                                                                                                                                   |
| psram_a[7]                                                                                       | OUT | G16  | 2    | PSRAM A7                                                                                                                                   |
| psram_a[8]                                                                                       | OUT | G18  | 2    | PSRAM A8                                                                                                                                   |
| psram_a[9]                                                                                       | OUT | H16  | 2    | PSRAM A9                                                                                                                                   |
| psram_a[10]                                                                                      | OUT | H17  | 2    | PSRAM A10                                                                                                                                  |
| psram_a[11]                                                                                      | OUT | H18  | 2    | PSRAM A11                                                                                                                                  |
| psram_a[12]                                                                                      | OUT | J16  | 2    | PSRAM A12                                                                                                                                  |
| psram_a[13]                                                                                      | OUT | J17  | 2    | PSRAM A13                                                                                                                                  |
| psram_a[14]                                                                                      | OUT | C20  | 2    | PSRAM A14                                                                                                                                  |
| psram_a[15]                                                                                      | OUT | D19  | 2    | PSRAM A15                                                                                                                                  |
| psram_a[16]                                                                                      | OUT | E19  | 2    | PSRAM A16                                                                                                                                  |

|             |     |     |   |                                              |
|-------------|-----|-----|---|----------------------------------------------|
| psram_a[17] | OUT | E20 | 2 | PSRAM A17                                    |
| psram_a[18] | OUT | F19 | 2 | PSRAM A18                                    |
| psram_a[19] | OUT | F20 | 2 | PSRAM A19                                    |
| psram_a[20] | OUT | G20 | 2 | PSRAM A20                                    |
| psram_a[21] | OUT | H20 | 2 | PSRAM A21                                    |
| psram_a[22] | OUT | P18 | 3 | Always 0 (byte→word shift): NC on the board. |

*PSRAM data bus psram\_dq[15:0] — 16 individual balls (banks 2 and 3)*

|              |    |     |   |                                                                   |
|--------------|----|-----|---|-------------------------------------------------------------------|
| psram_dq[0]  | IO | K18 | 2 | PSRAM DQ0                                                         |
| psram_dq[1]  | IO | C18 | 2 | PSRAM DQ1 (dual-function ball, used as ordinary GPIO).            |
| psram_dq[2]  | IO | D17 | 2 | PSRAM DQ2                                                         |
| psram_dq[3]  | IO | D20 | 2 | PSRAM DQ3                                                         |
| psram_dq[4]  | IO | G19 | 2 | PSRAM DQ4                                                         |
| psram_dq[5]  | IO | J18 | 2 | PSRAM DQ5                                                         |
| psram_dq[6]  | IO | J19 | 2 | PSRAM DQ6                                                         |
| psram_dq[7]  | IO | J20 | 2 | PSRAM DQ7                                                         |
| psram_dq[8]  | IO | K19 | 2 | PSRAM DQ8                                                         |
| psram_dq[9]  | IO | K20 | 2 | PSRAM DQ9                                                         |
| psram_dq[10] | IO | L17 | 3 | PSRAM DQ10                                                        |
| psram_dq[11] | IO | M18 | 3 | PSRAM DQ11                                                        |
| psram_dq[12] | IO | M17 | 3 | PSRAM DQ12                                                        |
| psram_dq[13] | IO | N16 | 3 | PSRAM DQ13                                                        |
| psram_dq[14] | IO | N18 | 3 | PSRAM DQ14                                                        |
| psram_dq[15] | IO | P17 | 3 | PSRAM DQ15 (bidirectional tri-state data bus, dq_oe = direction). |

*PSRAM control (bank 3)*

|            |     |     |   |                                            |
|------------|-----|-----|---|--------------------------------------------|
| psram_ce_n | OUT | N17 | 3 | Chip enable, active low.                   |
| psram_oe_n | OUT | R16 | 3 | Output enable (read).                      |
| psram_we_n | OUT | R17 | 3 | Write enable (write).                      |
| psram_lb_n | OUT | T16 | 3 | Lower-byte enable (DQ[7:0]).               |
| psram_ub_n | OUT | N19 | 3 | Upper-byte enable (DQ[15:8]).              |
| psram_zz_n | OUT | N20 | 3 | Sleep/snooze (inactive=high in operation). |

**Board signals not exposed as RTL ports**

Not ports of `spi_neuron_top` but required at board level: the **configuration SPI** lines to the onboard NOR flash (`PROGRAMN/INITN/DONE/CCLK...`, the datasheet's "Miscellaneous Dedicated Pins") and the 4 **JTAG** lines (`TCK/TMS/TDI/TDO`), the **oscillator** on the `PCLK` pad, the **power supplies**. Their ball numbers are not in the Lattice datasheet (separate file) but are not needed here: dedicated pins with no RTL port, `nextpnr` never requires them (0 errors), they matter only for the PCB schematic.

**Application SPI separate from configuration SPI**

The application SPI (`sclk/mosi/miso/cs_n`) must land on ordinary I/Os, **never** on the configuration-SPI pins: the config-SPI clock pin is not reusable as a general-purpose input after configuration without a board-level workaround. Keeping them physically separate avoids that problem.

## 12.4 Per-bank allocation (real die geometry)

The placement follows the die-edge geometry (from Trellis's `globals.json`, `ball` → `(col,row)` → `bank`): banks **2 and 3** sit contiguously along the chip's **right** edge and together hold the entire PSRAM bus (44+1 signals) — exactly the “one or two adjacent banks” recommended. Bank **7** (**left** edge, physically opposite the PSRAM bus) holds the application SPI and clock/reset, deliberately on the far side so the two buses do not cross. `clk` is on the dedicated pad `H5` (`GR_PCLK7_0`). Where a bank ran out of plain balls (part of `psram_dq`), the next dual-function ball was used as ordinary GPIO, confirmed usable by the real place&route.

| Signal group                                                                            | # pins | Bank (real)                                            |
|-----------------------------------------------------------------------------------------|--------|--------------------------------------------------------|
| PSRAM addresses <code>psram_a[21:0]</code>                                              | 22     | bank 2 (right edge)                                    |
| PSRAM data <code>psram_dq[15:0]</code>                                                  | 16     | banks 2 + 3 (adjacent)                                 |
| PSRAM control ( <code>ce/oe/we/lb/ub/zz</code> )                                        | 6      | bank 3                                                 |
| Application SPI                                                                         | 4      | bank 7 (left edge)                                     |
| Host attention pins ( <code>irq_n, data_ready_n</code> )                                | 2      | bank 7                                                 |
| Independent flash SPI bus<br>( <code>flash_sclk/flash_mosi/flash_miso/flash_cs</code> ) | 4      | bank 7                                                 |
| Clock / reset                                                                           | 2      | bank 7, <code>clk</code> on<br><code>GR_PCLK7_0</code> |
| Boot config SPI / JTAG                                                                  | —      | dedicated pins (outside<br>RTL, PCB only)              |

## 12.5 PSRAM subsystem

The `psram_controller.v` controller implements an **asynchronous parallel** interface (address bus, 16-bit data, `ce_n/oe_n/we_n` and byte-lanes `lb_n/ub_n`, plus `zz_n`) with an access latency of **70 ns** wired as  $\lceil 70 \text{ ns} \times f_{clk} \rceil$ . It is an asynchronous-SRAM-style bus, not QSPI.

| Role               | Component                                                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Working memory     | ISSI <code>IS66WVE4M16EBLL-70BLI</code> — 64 Mbit parallel PSRAM (4M×16, 8 MB), async, 70 ns, an exact match to the controller timing.                             |
| Fallback           | ISSI <code>IS61WV6416DBLL</code> / <code>IS61WV102416BLL</code> (true async SRAM, drop-in on the same signals, <code>zz_n</code> inactive, ~10 ns, lower density). |
| Persistent storage | Winbond <code>W25Q128JV</code> — 16 MB SPI NOR flash for bitstream, weights, bias, network metadata.                                                               |

### 12.5.1 PSRAM connection (FPGA-exclusive)

The PSRAM is driven **exclusively by the FPGA** through `psram_controller.v`: no external master touches the bus. The external host (RPi/ESP32/MCU) only speaks SPI to the FPGA and never touches these lines. Pin-by-pin connection FPGA ↔ ISSI `IS66WVE4M16EBLL-70BLI`:

| FPGA signal                 | PSRAM pin | Function                                                           |
|-----------------------------|-----------|--------------------------------------------------------------------|
| <code>psram_a[21:0]</code>  | A0–A21    | Address bus (22 lines, 8 MB word address).                         |
| <code>psram_dq[15:0]</code> | DQ0–DQ15  | Bidirectional data bus (tri-state, <code>dq_oe=direction</code> ). |
| <code>psram_ce_n</code>     | CE#       | Chip enable (active low).                                          |

|            |     |                                        |
|------------|-----|----------------------------------------|
| psram_oe_n | OE# | Output enable (read).                  |
| psram_we_n | WE# | Write enable (write).                  |
| psram_lb_n | LB# | Lower-byte enable (DQ[7:0]).           |
| psram_ub_n | UB# | Upper-byte enable (DQ[15:8]).          |
| psram_zz_n | ZZ# | Sleep/snooze (held high in operation). |

PSRAM supply: **3.3 V** (BLL variant), on the same I/O rail as banks 2/3 to which it is wired (ch. 12, real balls). Decoupling per supply pin per the ISSI datasheet.

## 12.6 Clock

There is no PLL in the RTL yet: `CLK_FREQ_MHZ` is a *timing parameter* (it feeds the PSRAM access formulas), not a clock generator. The mounted oscillator drives `clk` directly. Recommendation: a 16 MHz MEMS oscillator (SiTime SiT2001B family), well below the 67.91 MHz Fmax of the full integrated system (incl. flash subsystem, ch. 11). `CLK_FREQ_MHZ` must be set to the real value of the mounted oscillator, otherwise the PSRAM timing comes out wrong.

## 12.7 Power

A **three-rail** tree (the Lattice eval board's SERDES section is not needed and is omitted: no 1.2 V `VCCA/VCCHTX`):

| Rail           | Voltage | Feeds / regulator                                  |
|----------------|---------|----------------------------------------------------|
| VCC (core)     | 1.1 V   | FPGA core logic. Buck TLV62568, $\geq 600$ mA.     |
| VCCIO0/2/3/6/7 | 3.3 V   | I/O of all used banks + PSRAM. Buck TLV62568, 1 A. |
| VCCAUX         | 2.5 V   | FPGA auxiliary. LDO TLV73325, 10 mA.               |

Decoupling: at least one capacitor per supply pin + bulk per rail, per the Lattice ECP5 hardware checklist. Input: external 12 V (or match the bucks to the source).

## 12.8 Configuration and programming

Writing the FPGA “map” (bitstream) happens through dedicated silicon pins, **not** RTL top-level ports. Default mode: **MSPI** — automatic boot from the NOR flash at power-on (standalone product); JTAG available for development.

### 12.8.1 JTAG (development / debug)

| Signal | Ball <sup>†</sup> | Function          |
|--------|-------------------|-------------------|
| TCK    | T5                | Test clock.       |
| TDI    | R5                | Test data in.     |
| TDO    | V4                | Test data out.    |
| TMS    | U5                | Test mode select. |

### 12.8.2 Config-SPI to boot flash

The FPGA loads the bitstream from the **Winbond w25q128jv** (128 Mbit SPI NOR, Quad read) at power-on. The flash subsystem (`rtl/flash_slot_manager.v`, Phases F1-F7, ch. 11) uses the **same physical flash** for network weights/bias/ metadata at runtime, FPGA-exclusive access:

after configuration, the FPGA regains control of the chip through a fully independent 4-wire SPI bus, `flash_sclk/flash_mosi/flash_miso/flash_cs_n` (all ordinary GPIO, pp. 2–3 and §“Signal map” — no ECP5 config primitive involved, Phase F7) — this still implies a board-level dual connection (the flash’s DI/DO/CS/CLK pins wired both to the dedicated boot pins below and to these 4 ordinary balls, since it is the same physical chip serving both roles), not yet captured in a schematic (none exists yet, see the checklist below).

| Signal        | Ball <sup>†</sup> | Function                                    |
|---------------|-------------------|---------------------------------------------|
| CCLK/MCLK/SCK | U3                | Configuration clock.                        |
| DQ0_MOSI      | W2                | Config data (MOSI).                         |
| DQ1_MISO      | V2                | Config data (MISO).                         |
| BUSY_CSSPIN   | R2                | Flash chip-select.                          |
| DQ2 / DQ3     | Y2 / W1           | Quad-read lines.                            |
| PROGRAMN      | W3                | Start reconfiguration (button, active low). |
| INITN         | V3                | Init / configuration error (LED).           |
| DONE          | Y3                | Configuration complete (LED).               |
| CFGMDN[2:0]   | R4/T4/U4          | Mode select (see below).                    |

### 12.8.3 Configuration modes (CFGMDN)

| Mode                   | CFGMDN[2:0] | Use                          |
|------------------------|-------------|------------------------------|
| MSPI (boot from flash) | 010         | <b>Default</b> — standalone. |
| SSPI (slave SPI)       | 001         | Config from external host.   |
| SCM (slave serial)     | 101         | Serial config.               |
| SPCM (slave parallel)  | 111         | 8-bit parallel config.       |

#### Configuration balls to verify on the 45F

<sup>†</sup>The JTAG and config-SPI balls listed here are the *reference* from the Lattice eval board (85F device). JTAG and config-SPI are dedicated, largely fixed pins in the ECP5 family, but the exact positions on the LFE5U-45F-8BG381C target must be confirmed against the Lattice 45F pinout file (Diamond/Radiant or the Trellis database) before committing them to the schematic, as already done for the application signals (ch. 12). **Distinct from this open item** (do not conflate the two): the flash subsystem’s own runtime SPI pins (`flash_sclk`, `flash_mosi`, `flash_miso`, `flash_cs_n` — Phases F1-F6, made fully independent in Phase F7) **are** real, pinned, place&route-verified ordinary GPIO on bank 7 — **no pin shared with any ECP5 config primitive**: an earlier version reused the boot CCLK pad for SCLK via `USRMCLK`, dropped in Phase F7 (`USRMCLK` utilisation in the current full-system synthesis is 0/1, confirming it is no longer used at all).

## 12.9 Open tasks before schematic capture

- OK** ADDR\_WIDTH=23 (full 8 MB) across all modules and testbenches.
- OK** Real .lpf with the CABGA381 ball assignment, place&route-verified with 0 errors (synth/ecp5/spi\_neuron\_top.lpf, 57 signals incl. flash subsystem).
- OK** Boot/persistence flash subsystem (Phases F1-F7): SPI master, copy engine, CRC32 slot catalog, fully independent 4-wire SPI bus, real synthesis at 0 errors, Fmax 67.91 MHz (WORKLOG.md).

- ☐ Confirm PSRAM/SPI signal integrity at the actually mounted clock.
- ☐ Board-level dual-wiring diagram for the flash's DI/DO/CS/CLK pins (dedicated boot pins + the flash subsystem's 4 ordinary balls) — not yet captured in a schematic.
- ☐ Choice of the JTAG connector footprint.
- ☐ Schematic capture (KiCad or other): no schematic exists yet for this device/package combination.

## CHAPTER 13

# Quick reference

### 13.1 SPI opcodes

| Value | Name        | Response    | Summary                       |
|-------|-------------|-------------|-------------------------------|
| 0x00  | NOP         | —           | idle                          |
| 0x01  | WRITE_RAM   | —           | PSRAM block write             |
| 0x02  | READ_RAM    | len B       | PSRAM block read              |
| 0x0F  | RESET       | —           | engine reset + STATUS latch   |
| 0x10  | SET_BASE    | —           | set base/register (sel 0..10) |
| 0x20  | START       | —           | single-layer start            |
| 0x21  | STATUS      | 1 B         | busy(live)/done(sticky)       |
| 0x22  | READ_OUTPUT | N_NEURONS B | y_bus                         |
| 0x23  | RUN_NETWORK | —           | multi-layer start             |
| 0x30  | READ_CONFIG | 11 B        | configuration record          |

### 13.2 STATUS byte

|       |       |       |       |       |       |       |              |
|-------|-------|-------|-------|-------|-------|-------|--------------|
| bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0        |
| 0     | 0     | 0     | 0     | 0     | 0     | done  | reserved = 0 |

done is sticky, clear-on-read; busy is live.

### 13.3 SET\_BASE selectors

- 0 — x\_base
- 1 — w\_base
- 2 — bias\_addr
- 3 — table\_base
- 4 — buf\_a\_base
- 5 — buf\_b\_base
- 6 — activation (single-layer)
- 7 — n\_inputs\_real (single-layer)
- 8 — n\_neurons\_real (single-layer)
- 9 — num\_neurons\_graph (Type #2)
- 10 — n\_out (Type #2)

### 13.4 Descriptor table (11 bytes/layer, MSB-first)

|        |           |     |               |                |
|--------|-----------|-----|---------------|----------------|
| w_base | bias_addr | act | n_inputs_real | n_neurons_real |
| 3B     | 3B        | 1B  | 2B            | 2B             |

### 13.5 Build parameters

- DATA\_WIDTH — 8 (INT8)
- ACC\_WIDTH — 32 (INT32)
- N\_INPUTS — max inputs
- N\_NEURONS — max neurons
- PARALLEL — simultaneous MACs
- N\_LAYERS — max layers
- ADDR\_WIDTH — 23 (8 MB)
- MEM\_DATA\_WIDTH — 16
- CLK\_FREQ\_MHZ — PSRAM timing

## CHAPTER 14

# Roadmap and development status

### 14.1 Development phases

| Phase | Title                    | Status          | Content                                                                                                                                                                                      |
|-------|--------------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | Parametric layer         | OK              | inputs/neurons/parallelism, accumulation, bias, ReLU; test $32 \times 4/P=8$ .                                                                                                               |
| 2     | Parameter sweep          | OK              | multiple configurations incl. non-multiple and degenerate; elaboration guard added.                                                                                                          |
| 3     | Memory architecture      | OK              | <code>neuron_memory</code> single/multi-neuron, real PSRAM tested; multi-layer buffers → Phase 5.                                                                                            |
| 4     | SPI interface            | OK              | <code>spi_slave+spi_engine</code> , 17 opcodes incl. flash subsystem, Fmax checked at full-system level.                                                                                     |
| 5     | Multi-layer network      | OK <sup>†</sup> | <code>layer_sequencer</code> , configurable activations, runtime width; real toolchain checked.                                                                                              |
| 6     | Host software            | planned         | Linux and ESP32 drivers on the same protocol.                                                                                                                                                |
| 7     | Optimization             | in progress     | timing closure done (55→75 MHz); PSRAM page-mode done (gather bandwidth +42%); <i>x/w</i> block RAM remains.                                                                                 |
| 8     | Hardware training (opt.) | future          | backprop, gradients, weight update.                                                                                                                                                          |
| 9     | Flash subsystem (F1-F7)  | OK              | dedicated SPI master, flash↔PSRAM copy engine, 16-slot catalog with CRC32, fully independent 4-wire SPI bus (F7), 8 opcodes ( <code>0x40–0x47</code> , ch. 8 §8.8); real synthesis 0 errors. |

<sup>†</sup> RTL, unit tests and end-to-end over simulated SPI complete; timing closure done: 75.30 MHz (P2) / 60.26 MHz (P8) at the time of Phase 5, bit-exact across the whole regression; Fmax of the full system after Phase 9 (incl. independent flash subsystem): **67.91 MHz** (ch. 11).

### 14.2 Component status

| Component                             | Status     |
|---------------------------------------|------------|
| Parametric neural layer               | OK working |
| Parametric inputs/neurons/parallelism | OK         |
| Accumulation, bias, ReLU              | OK         |
| $32 \times 4$ / $P=8$ validation      | OK         |

|                                                                              |                                                                |
|------------------------------------------------------------------------------|----------------------------------------------------------------|
| Dedicated RAM (interface + controller + INT8 access)                         | <b>OK</b> tested on real PSRAM                                 |
| SPI interface (17 opcodes incl. RUN_NETWORK + flash)                         | <b>OK</b> Fmax at full-system level                            |
| Dual SPI                                                                     | future                                                         |
| Multi-layer engine                                                           | <b>OK</b> timing closure 75.30 MHz (P2) at the time of Phase 5 |
| Configurable activations (ACT_NONE/ACT_RELU)                                 | <b>OK</b>                                                      |
| Runtime network width (one bitstream, any topology)                          | <b>OK</b> measured savings                                     |
| Type #2 graph network (act_buffer, graph_engine, netasm)                     | <b>OK</b> RTL + tests + synthesis                              |
| CABGA381 pinout (real .1pf, 57 signals incl. flash)                          | <b>OK</b> place&route-verified, 0 errors                       |
| PSRAM page-mode (G7)                                                         | <b>OK</b> done (37.53 cycles/edge, bandwidth +42%)             |
| Flash subsystem (SPI master, copy engine, CRC32 catalog, independent bus F7) | <b>OK</b> real synthesis 0 errors, Fmax 67.91 MHz              |
| Real bitstream (ecppack, P2/P8)                                              | <b>OK</b> 0 errors, part LFE5U-45F-8CABGA381                   |
| Linux / ESP32 host driver                                                    | planned                                                        |
| Hardware training                                                            | future                                                         |

### 14.3 Architectural principle (summary)

#### Foundation of the project

The FPGA implements the neural machine and owns its own RAM; the host configures and uses the machine. A build fixes the *ceiling* (max layers, max width, PARALLEL); the host configures the *actual* network — number of layers, per-layer width, per-layer activation, trained parameters — entirely at runtime, over SPI, into the FPGA's local memory. A single bitstream serves any topology up to that ceiling.

### 14.4 Long-term vision

The final goal is a reusable hardware block integrable into different future projects: the host platform can change (Linux, ESP32, MCU, PC) without changing the fundamental architecture of the engine. The FPGA becomes a dedicated neural computation peripheral, optimized for the topology required by each application.

## CHAPTER A

# Modules, ports and toolchain

### A.1 List of RTL modules

| File                   | Type          | Role                                         |
|------------------------|---------------|----------------------------------------------|
| rtl/mac_unit.v         | combinational | single multiply-accumulator                  |
| rtl/mac8.v             | combinational | parallel MAC + balanced adder tree           |
| rtl/neuron_parallel    | FSM           | neuron: groups, bias, activation, saturation |
| rtl/layer.v            | structural    | N_NEURONS neurons in parallel                |
| rtl/neuron_memory.v    | FSM           | memory/neuron bridge, neuron loop            |
| rtl/layer_sequer       | FSM           | multi-layer sequencing, ping-pong            |
| rtl/int8_memory_access | FSM           | byte ↔ word conversion                       |
| rtl/memory_inter       | FSM           | req/ready handshake                          |
| rtl/psram_controller   | FSM           | async 70 ns physical PSRAM bus               |
| rtl/mem_arbiter.       | arbiter       | 3 ports, priority B>C>A                      |
| rtl/spi_slave.v        | FSM           | SPI Mode 0 physical layer + CDC              |
| rtl/spi_engine.v       | FSM           | opcode + register bank                       |
| rtl/act_buffer.v       | block RAM     | DP16KD activation buffer (Type #2)           |
| rtl/graph_engine       | FSM           | graph-network engine (Type #2)               |
| rtl/spi_neuron_top.v   | top           | full integration                             |
| rtl/memory_model       | model         | behavioral RAM (sim)                         |

### A.2 Ports of the top-level spi\_neuron\_top

See the complete signal-by-signal table in ch. 12. In summary: clock and reset (`clk`, `rst`); application SPI (`sclk`, `mosi`, `miso`, `cs_n`); PSRAM bus (`psram_a[22:0]`, `psram_dq[15:0]`, `psram_ce_n/oe_n/we_n/lb_n/ub_n/zz_n`).

### A.3 Toolchain

| Tool            | Version            | Use                                        |
|-----------------|--------------------|--------------------------------------------|
| Yosys           | 0.68+post          | RTL synthesis → JSON netlist, ECP5 mapping |
| nextpnr-ecp5    | 0.11.1-19-g8dbcee5 | placement, routing, timing                 |
| Project Trellis | install            | ecppack/ecppll/ecpbram                     |
| Icarus Verilog  | -g2012             | functional simulation                      |

#### A.3.1 Main nextpnr parameters

```

1 --45k                selects LFE5U-45F
2 --package CABGA381   package
3 --speed 8            speed grade -8
4 --json <netlist>     netlist from Yosys
5 --lpf <constraints>  pin constraints (currently empty)
6 --lpf-allow-unconstrained allows unconstrained I/Os (benchmark)
7 --freq 80            80 MHz timing target

```

### A.3.2 Simulation example

```

1 iverilog -g2012 -Ptb.PARALLEL=16 -o sim/parametric_256x4_p16 \
2   sim/parametric_tb.v rtl/mac_unit.v rtl/mac8.v \
3   rtl/neuron_parallel.v rtl/layer.v
4 vvp sim/parametric_256x4_p16

```

## A.4 Main testbenches

| Testbench                                     | Coverage                                                                                   |
|-----------------------------------------------|--------------------------------------------------------------------------------------------|
| parametric_tb.v                               | 256×4 datapath, accumulate/bias/ReLU/saturation cases                                      |
| parameter_sweep_tb.v                          | sweep of valid configurations                                                              |
| neuron_parallel_tb.v                          | activations, runtime width (T7)                                                            |
| neuron_memory_tb.v /<br>_multi_tb.v           | single/multi-neuron memory integration, real PSRAM (T5)                                    |
| psram_controller_tb.v                         | PSRAM controller                                                                           |
| psram_page_mode_tb.v                          | page bursts, page crossing, close on WRITE/ $t_{CEM}$ timeout, byte-enable changes (§ 5.5) |
| spi_slave_tb.v                                | SPI physical layer (4 tests)                                                               |
| spi_engine_tb.v                               | opcodes, registers (10+ tests)                                                             |
| spi_neuron_top_tb.v                           | end-to-end, real PSRAM over simulated SPI                                                  |
| spi_neuron_top_runnetworklayer_sequencer_tb.v | RUN_NETWORK 2-layer end-to-end<br>2-layer sequence, ping-pong, byte-exact copy             |

This datasheet is generated from the RTL code, the documentation and the benchmarks present in the repository [github.com/manvalan/FPGA-Neural](https://github.com/manvalan/FPGA-Neural) as of September 2026. The Fmax, resource usage and throughput values are those reported in the repository measurements (real .lpf already assigned and place&route-verified, ch. 12) and must be re-verified on any substantial RTL change or as the Phase 7 timing closure, still in progress, continues (ch. 14).