pccx

Parallel Compute Core eXecutor

Instruction Set Architecture

Software Developer's Manual

Volume 1 — Basic Architecture
Volume 2 — Instruction Set Reference
Volume 3 — System Programming Guide

Edge-FPGA NPU for Transformer-based LLM decoding
Target platform: Xilinx Kria KV260 (Zynq UltraScale+ ZU5EV)
Revision: v002 ▪ Order Number: pccx-v002-SDM

Preface

This manual is the authoritative reference for the **pccx v002 Instruction Set Architecture**, a fixed-length 64-bit VLIW-style ISA that drives a dedicated Neural Processing Unit (NPU) built to accelerate autoregressive decoding of Transformer-based Large Language Models on the Xilinx Kria KV260 platform (Zynq UltraScale+ ZU5EV).

The manual is organised as three conceptual volumes bound into a single book: Volume 1 describes the **basic architecture** of the NPU and its interaction with the host processing system; Volume 2 is the per-instruction reference written in the classical CPU reference-manual format (Opcode table, Description, Operation pseudocode, Flags Affected, Exceptions); Volume 3 is the **system programming guide** covering memory ordering, completion fences, and the host driver surface.

NOTATIONAL CONVENTIONS

- Bit ranges use the `[MSB:LSB]` convention, inclusive on both ends. `[63:60]` is a 4-bit field.
- Hexadecimal constants use the SystemVerilog literal form, e.g. `4'h0` (a 4-bit hex zero).
- The symbol ← is used in Operation pseudocode to denote assignment; ≡ denotes definitional equivalence.
- `dest` always precedes `src` in field ordering.
- A field annotated "reserved" must be written as zero by software. The decoder does not currently check this, but future revisions may assign the field.

RELATIONSHIP TO OTHER DOCUMENTATION

The pccx web portal at https://pccx.pages.dev/ contains a living, cross-referenced copy of the encoding tables together with interactive Mermaid, WaveDrom, and Graphviz diagrams. This PDF is the *offline preprint* mirror; both sources are rebuilt from the authoritative RTL definitions in the external repository pccxai/pccx-FPGA-NPU-LLM-kv260 under `hw/rtl/NPU_Controller/NPU_Control_Unit/ISA_PACKAGE/isa_pkg.sv`. Where this manual and `isa_pkg.sv` disagree, the SystemVerilog package is normative.

FEEDBACK

File issues against the `pccx` repository or submit pull requests to the same. The author reads every report.

Table of Contents

List of Figures

CHAPTER 1

ABOUT THIS MANUAL

1.1 Scope

This manual is the authoritative reference for the **pccx v002 Instruction Set Architecture** (hereafter "the ISA"). The ISA defines the exact binary contract between a host driver running on the Zynq UltraScale+ Processing System (PS) and the NPU execution complex that lives on the Programmable Logic (PL). Every opcode, every field, every reserved bit documented here mirrors `isa_pkg.sv` in the v002 RTL repository.

1.2 Audience

This document assumes familiarity with the following:

- Systolic arrays with Weight Stationary and Weight Streaming dataflow.
- The AXI4 and AXI4-Lite protocols, in particular the Zynq UltraScale+ HP and ACP port semantics.
- Transformer LLM decoding: Q/K/V projection, scaled dot-product attention, the FFN block, RMSNorm, RoPE, and token streaming.
- Block-floating-point (BFloat16) numerics and INT4 weight quantization (WNx/AxAy notation).

This manual is *not* a tutorial. For onboarding material, start from the pccx documentation portal.

1.3 Manual Layout

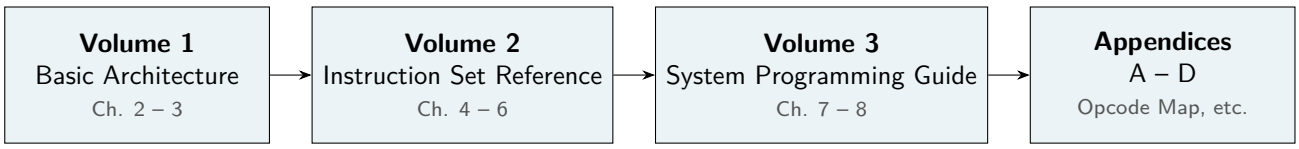


Figure 1.1: Manual layout — four volumes bound into one book.

CHAPTER 2

ARCHITECTURAL OVERVIEW

This chapter provides the architectural context required to read subsequent chapters. It describes the host–device boundary, the internal datapath of the NPU, the memory hierarchy, and the execution pipeline at a block level. Subsequent chapters refer back to the figures defined here.

2.1 System Context

The pccx NPU sits inside the PL of the Zynq UltraScale+ and is driven by the ARM Cortex-A53 cluster on the PS. The host issues instructions via an AXI4-Lite command FIFO and receives completion status via a second AXI4-Lite FIFO. Bulk data crosses via four AXI4 High-Performance ports (HP0–HP3) and one Accelerator Coherency Port (ACP) for low-latency transfers.

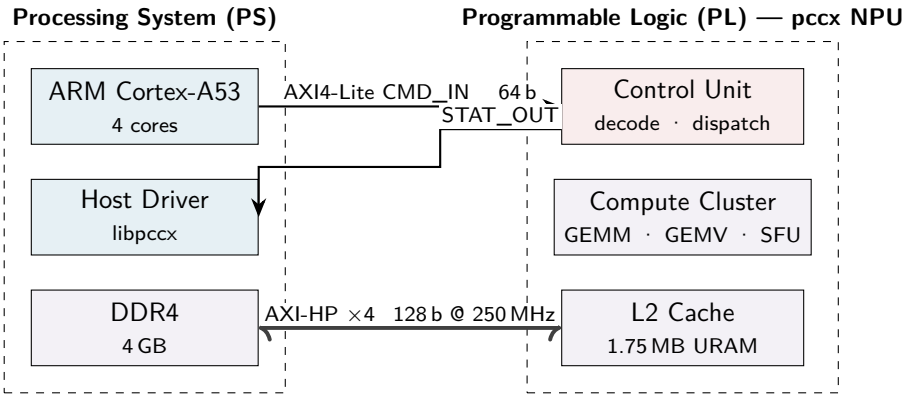


Figure 2.1: Host–device partition. PS issues 64-bit instructions to the Control Unit over AXI4-Lite, while bulk activations and weights cross via four 128-bit HP ports.

2.2 NPU Top-Level Block Diagram

Figure 2.2 expands the PL side of Figure 2.1. The Control Unit decodes each 64-bit instruction in one cycle and emits micro-ops (μops) into the Dispatcher, which resolves hazards and feeds the GEMM, GEMV, SFU, and DMA back-end resources.

2.3 Clock Domains and CDC

Two clock domains meet inside the NPU: the 250 MHz AXI domain adjacent to the PS, and the 400 MHz compute core. A bank of Xilinx Parameterized Macro (XPM) asynchronous FIFOs in `independent_clock` mode is the only path that crosses the boundary; there is no manual synchronizer logic.

AXI-HP Read Handshake

Every weight or activation byte that crosses from the PS side of the chip to the PL side does so over the AXI-HP 4-channel burst-capable read protocol. The ingress direction of one such handshake

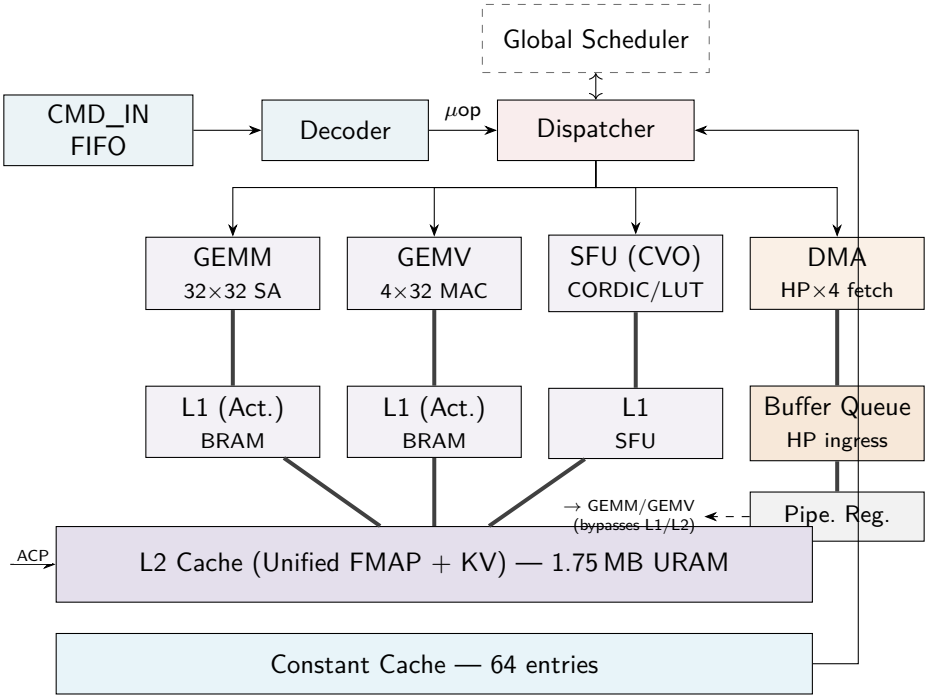


Figure 2.2: NPU top-level block diagram. **Activation path** (solid bus): ACP → L2 → L1(Act.) → compute. **Weight path** (labelled dashes): AXI-HP → Buffer Queue → Pipe.Reg → GEMM/GEMV direct, bypassing L1 and L2. SFU uses L1/L2 for CVO operands.

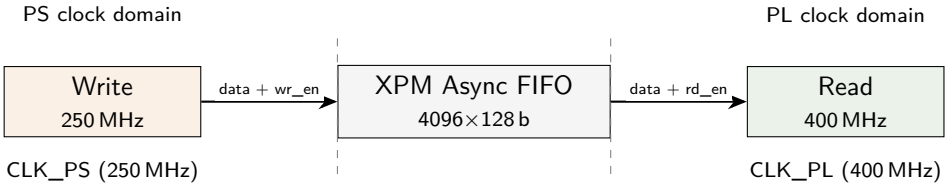


Figure 2.3: Clock-domain crossing path: a 4096×128 b XPM FIFO in `independent_clock` mode decouples the 250 MHz AXI side from the 400 MHz compute core.

(single beat shown; a full burst chains 16 of these on the data phase) is illustrated in Figure 2.4. The protocol enforces three rules the software driver must respect:

- `ARVALID` must remain asserted until both `ARVALID` and `ARREADY` are sampled high on the same rising edge (the *address handshake*).
- `ARADDR` must be stable across that window.
- After the handshake, the master must be ready to accept `RDATA` within at most `T_rdata_max` cycles of the corresponding `RVALID` assertion.

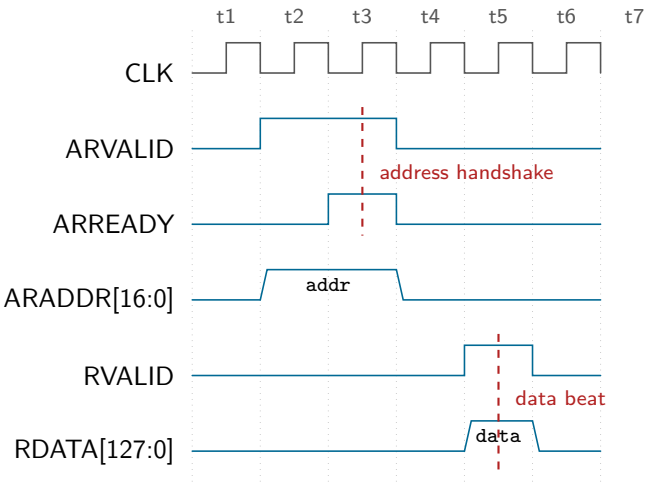


Figure 2.4: AXI-HP read protocol, single beat. On cycle `t3` both `ARVALID` and `ARREADY` are high — the address is captured. Two cycles of round-trip latency later (cycle `t5`) the slave returns `RDATA` with `RVALID`. A full burst chains 16 data beats on back-to-back cycles once the address phase has committed.

2.4 Instruction Pipeline

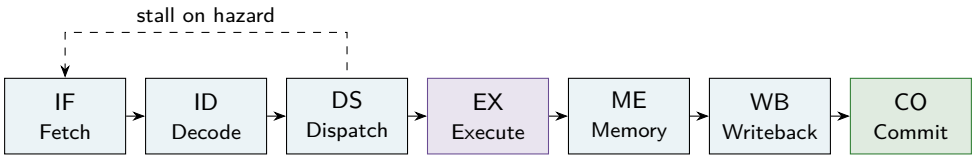


Figure 2.5: Conceptual 7-stage NPU pipeline. Because each instruction can launch thousands of MAC cycles, the **Execute** stage dominates real latency; the front end merely streams instructions in as fast as resources free up.

2.5 Memory Hierarchy

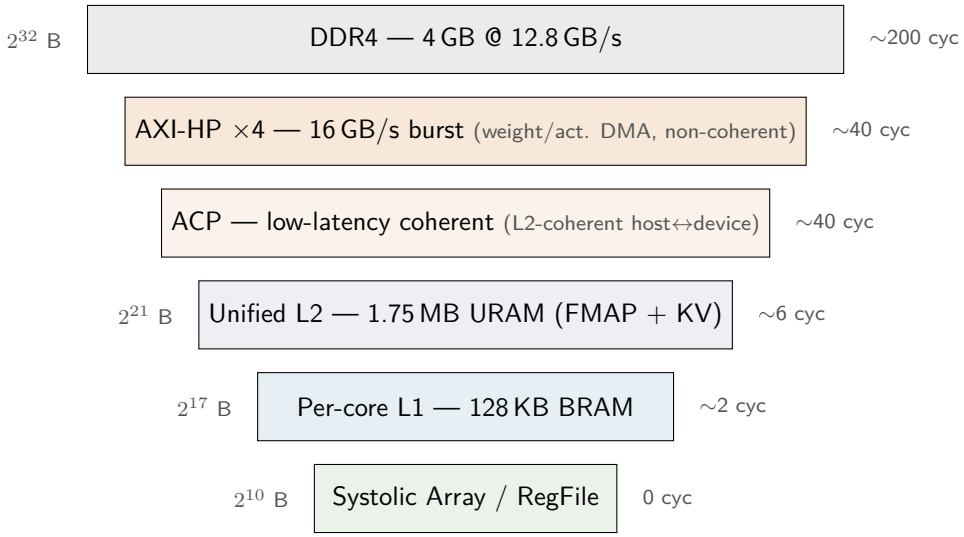


Figure 2.6: Memory hierarchy pyramid. AXI-HP ×4 carries non-coherent weight/activation DMA (direct to Buffer Queue, bypassing L2); ACP provides coherent host↔L2 transfers. The unified 1.75 MB L2 dominates the on-chip budget.

URAM allocation breakdown

Of the 64 URAM blocks on the KV260, 8 are claimed by the 4-port Weight FIFO (2 per port) and the remaining 56 are paired into 28 structural pairs to form the unified L2 (28 pairs × 4096 depth × 128-bit = 114 688 entries × 16 B = ~1.75 MB). See Figure 2.7.

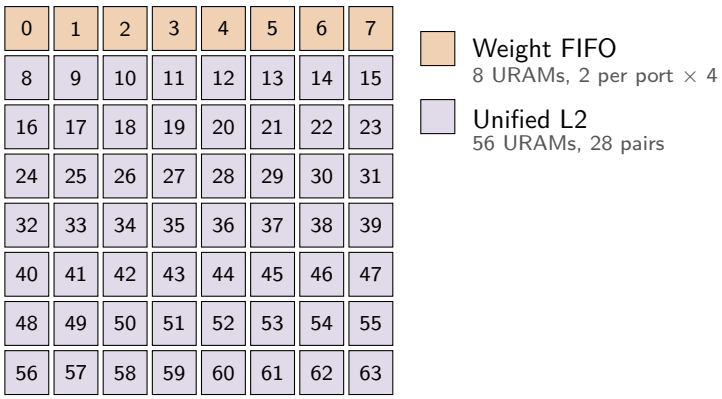


Figure 2.7: Physical URAM allocation on the Kria KV260 (64 blocks total). Blocks 0–7 form the 4-port Weight FIFO; blocks 8–63 pair up to create the 1.75 MB unified L2.

CHAPTER 3

NUMERICS AND EXECUTION MODEL

3.1 Block Floating Point + INT4 Weight Quantization

pccx uses a **W4A8** quantisation scheme: weights are stored as INT4 and activations (feature maps, FMAP) are kept as INT8 throughout the datapath. A single DSP48E2 executes the 9-bit × 4-bit inner multiplication natively. The accumulated integer result is scaled by the joint BF16 factor $S_{\text{fmap}} \cdot S_{\text{weight}}$ *after* accumulation, in the Post-process stage.

Feature maps remain INT8 for all linear layers (Q/K/V projection, FFN). Promotion to BF16 is performed *only* before entering the Special Function Unit (SFU) for operations that require floating-point precision: RMSNorm, RoPE (sin/cos), softmax, and GELU. This keeps BF16 traffic confined to the SFU lane and off the main datapath.

For two activation–weight pairs sharing scale factors S_{fmap} (for BF16) and S_{weight} (for INT4):

$$\begin{aligned} A &= a \cdot S_{\text{fmap}}, & B &= b \cdot S_{\text{fmap}} \\ C &= c \cdot S_{\text{weight}}, & D &= d \cdot S_{\text{weight}} \end{aligned} \tag{3.1}$$

The quantity $(A \cdot C) + (B \cdot D)$ factors to:

$$\text{Result} = \underbrace{(a \cdot c + b \cdot d)}_{\text{DSP48E2 integer MAC}} \cdot \underbrace{S_{\text{fmap}} \cdot S_{\text{weight}}}_{\text{post-process scalar}} \tag{3.2}$$

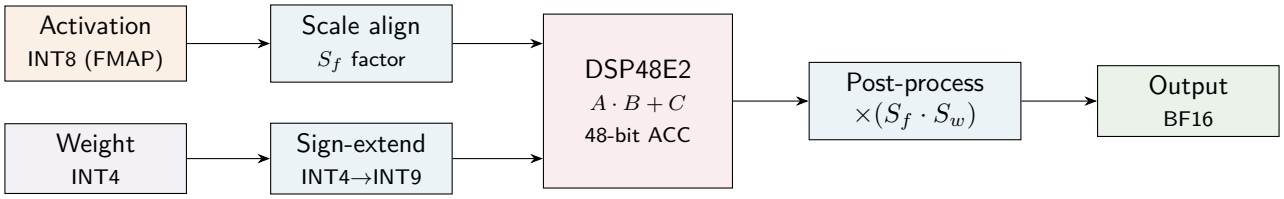


Figure 3.1: DSP48E2 datapath (W4A8). INT8 activations are sign-extended to INT9; INT4 weights are sign-extended to INT4. The 9×4 b product accumulates in a 48-bit DSP48E2 register; the joint BF16 scale factor $S_f \cdot S_w$ is applied once in the Post-process stage.

3.2 Systolic Dataflow — Weight Stationary (GEMM)

In the Weight Stationary schedule, a tile of INT4 weights is preloaded once into the 32×32 PE array and held fixed for the duration of the tile computation. Activations (INT8, one row per cycle) enter from the left and propagate right; partial sums (48-bit) enter from the top and propagate downward. Each PE performs one MAC per cycle with no data movement on the weight side, maximising weight reuse and minimising the HP bandwidth required per GFLOP.

After the tile completes, the accumulator values exit at the bottom edge of the array, pass through the Post-process stage (BF16 scale multiply), and are written back to L2. The next weight tile is then preloaded and the loop repeats. See Figure 3.2 for the dataflow diagram.

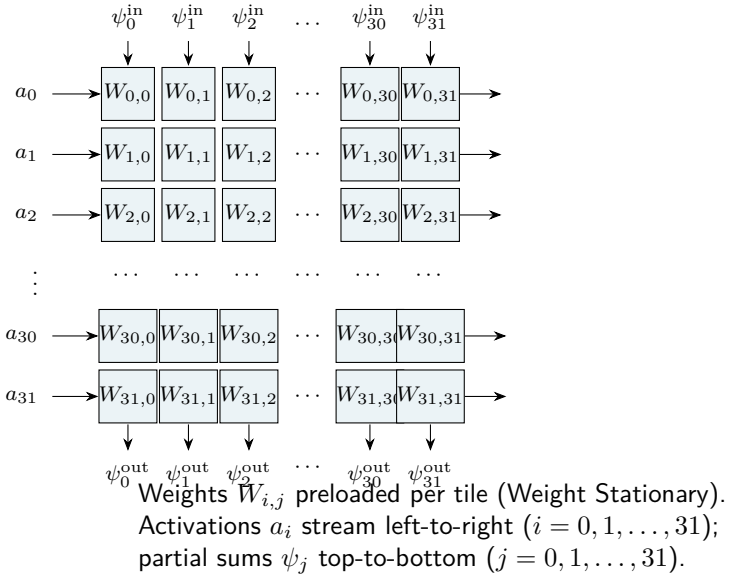


Figure 3.2: Systolic array dataflow (32×32; middle row/column compressed for clarity). Weights $W_{i,j}$ are preloaded once per tile; activations a_i stream left-to-right; partial sums ψ_j propagate top-to-bottom every cycle.

3.3 Systolic Dataflow — Weight Streaming (GEMV)

GEMV differs from GEMM in that the weight matrix is streamed through 4 parallel cores row-partition-wise rather than pinned inside PEs. A single GEMV dispatches 32-lane LUT-based MAC units followed by a 5-stage reduction tree, producing one scalar per decoded token.

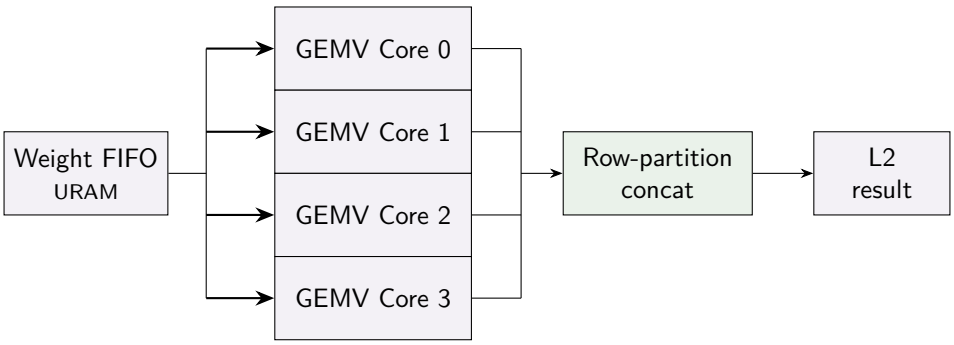


Figure 3.3: GEMV weight streaming: the 4 GEMV cores each consume a row-partition stream from the URAM Weight FIFO; results concatenate into a single output vector in L2.

Each GEMV core reduces its 32 partial products to a single scalar through a five-stage binary tree (Figure 3.4). Stage 1 is implemented with 16 DSP48E2 slices running in $A + B$ mode; stages 2–5 fall to 6-LUT adders to conserve the DSP budget for the GEMM array.

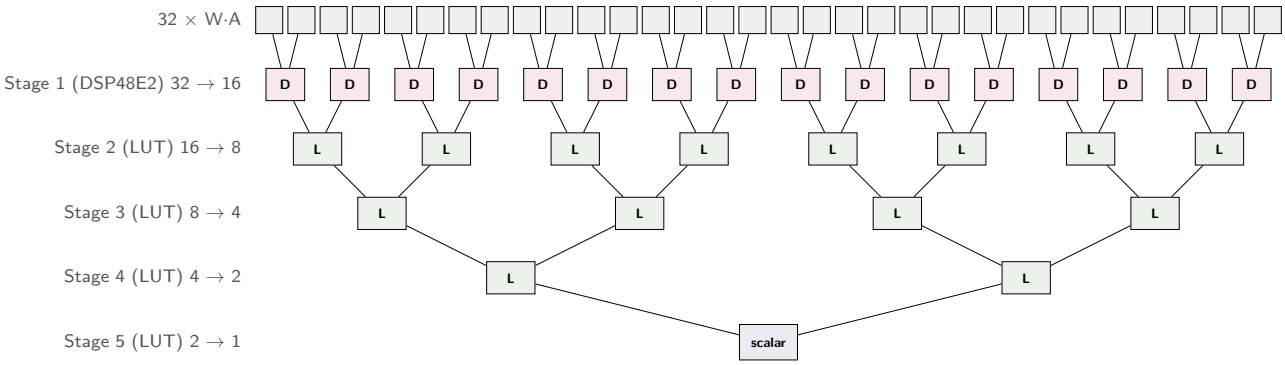


Figure 3.4: Per-core GEMV reduction tree (one of four cores shown). Stage 1 absorbs 32 partial products into 16 sums using DSP48E2 $A + B$ adders; stages 2–5 use 6-LUT adders. Each stage carries a pipeline register to support the 400 MHz timing target.

INSTRUCTION ENCODING

4.1 Top-Level 64-Bit Layout

Every pccx v002 instruction is exactly **64 bits wide** and encodes a 4-bit opcode in the most-significant nibble followed by a 60-bit instruction body whose shape depends on the opcode.

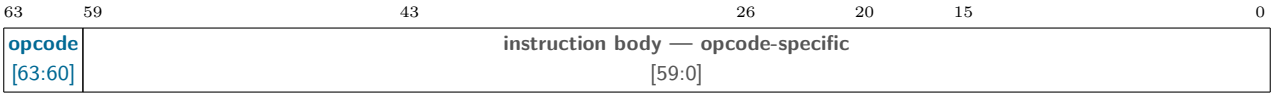


Figure 4.1: Top-level 64-bit instruction word.

4.2 Opcode Map

Of the 16 possible opcodes, five are defined in v002; the remaining eleven are reserved. Decoder behaviour on reserved opcodes is **undefined**.

Opcode	Mnemonic	Family	Function
`4'h0`	`GEMV`	Type A	Matrix × vector. Drives 4-core 32-MAC streaming engine.
`4'h1`	`GEMM`	Type A	Matrix × matrix. Drives 32×32 weight-stationary systolic array.
`4'h2`	`MEMCPY`	Type B	Bulk data movement: host ↔ device, L2 ↔ L2.
`4'h3`	`MEMSET`	Type C	Constant Cache writes (shape / size / scale).
`4'h4`	`CVO`	Type D	Complex Vector Op. Transcendentals + reductions via SFU.
`4'h5`–`4'hF`	—	—	*Reserved.*

4.3 Decode and Dispatch Pathway

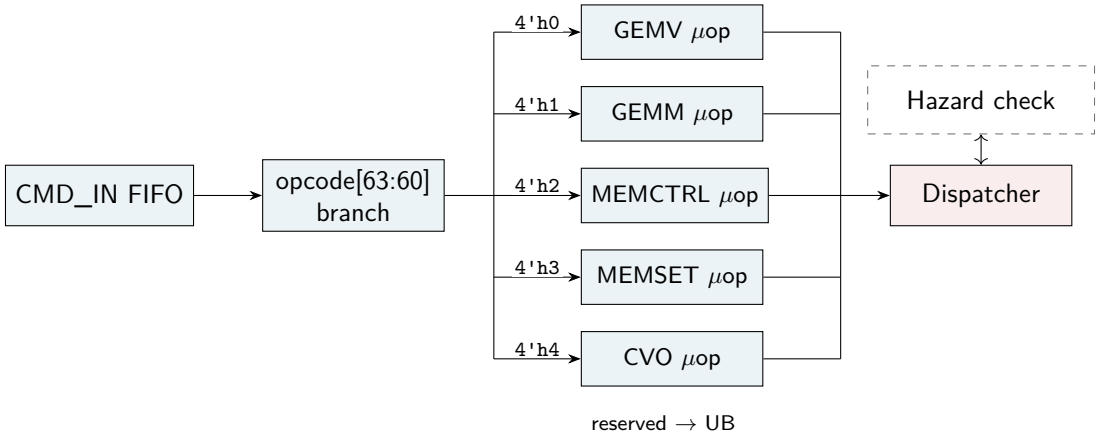


Figure 4.2: Decode and dispatch pathway. The opcode selects one of five μop types; the dispatcher resolves inter-instruction hazards before handing off to the back end.

4.4 μop Decomposition Summary

μop type	Constituent fields (width)
`gemm_control_uop_t`	`flags` (6 b) + `size_ptr_addr` (6 b) + `parallel_lane` (5 b)
`GEMV_control_uop_t`	Identical layout to `gemm_control_uop_t`.
`memory_control_uop_t`	`data_route` (8 b) + `dest_addr` (17 b) + `src_addr` (17 b) + `shape_ptr` (6 b) + `async` (1 b)
`memory_set_uop_t`	`dest_cache` (2 b) + `dest_addr` (6 b) + `a_value` (16 b) + `b_value` (16 b) + `c_value` (16 b)
`cvo_control_uop_t`	`cvo_func` (4 b) + `src_addr` (17 b) + `dst_addr` (17 b) + `length` (16 b) + `flags` (5 b) + `async` (1 b)

4.5 Instruction Format Families — Bit-Level Diagrams

The 60-bit body falls into four structural families. Figures 4.3, 4.4, 4.5, and 4.6 show each layout as a true bit-field diagram.

4.5.1 Type A — Compute Format (GEMV / GEMM)

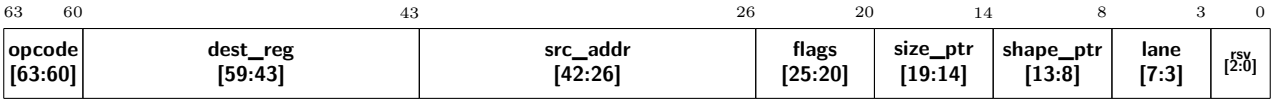


Figure 4.3: **Type A** body layout, used by `GEMV` (opcode `4'h0`) and `GEMM` (opcode `4'h1`).

4.5.2 Type B — Memory Control Format (MEMCPY)

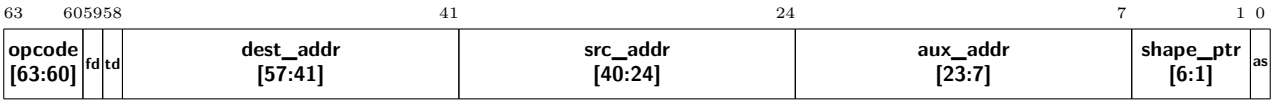


Figure 4.4: **Type B** body layout, used by `MEMCPY` (opcode `4'h2`). `fd`=`from_device`, `td`=`to_device`, `as`=`async`.

4.5.3 Type C — Memory Set Format (MEMSET)

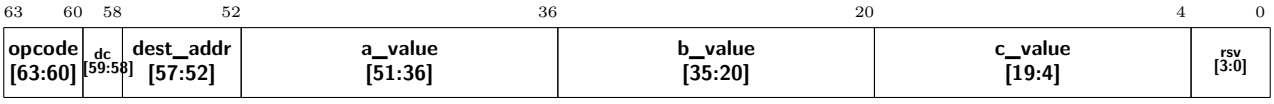


Figure 4.5: **Type C** body layout, used by `MEMSET` (opcode `4'h3`). `dc`=`dest_cache`.

4.5.4 Type D — Complex Vector Op Format (CVO)

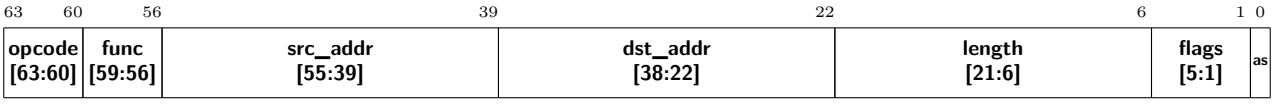


Figure 4.6: **Type D** body layout, used by `CVO` (opcode `4'h4`). `as`=`async`.

INSTRUCTION SET REFERENCE (A-Z)

GEMM / GEMV

Matrix-matrix / matrix-vector multiply

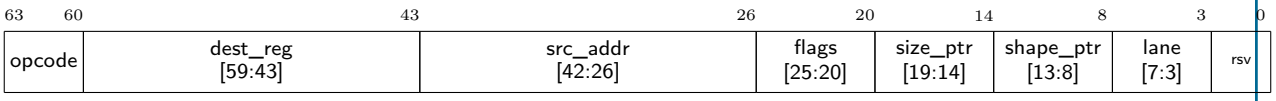
OPCODE & INSTRUCTION SUMMARY

Opcode	Instruction	Op/En	Description
`4'h0` *body*	`GEMV dest, src, flags, size_ptr, shape_ptr, lane`	A	Matrix-vector multiply. 4 GEMV cores consume weights via Weight Streaming.
`4'h1` *body*	`GEMM dest, src, flags, size_ptr, shape_ptr, lane`	A	Matrix-matrix multiply. 32×32 systolic array runs Weight Stationary.

INSTRUCTION OPERAND ENCODING

Op/En	Operand 1 — Operand 6
A	`dest` (Mod ModReg, W) | `src` (Mod ModReg, R) | `flags` (Imm, R) | `size_ptr` (Ptr, R) | `shape_ptr` (Ptr, R) | `lane` (Imm, R)

BIT-LEVEL ENCODING (TYPE A)



DESCRIPTION

Performs a tiled matrix multiplication. `src` points at the activation source in L2; `dest` receives the result. The dispatcher reads `shape_ptr` and `size_ptr` from the Constant Cache to obtain (M, N, K) and tile sizes, then streams a weight tile into either (a) the 32×32 systolic array for `GEMM`, or (b) the 4 parallel GEMV cores for `GEMV`. Accumulation is done by the DSP48E2 integer MAC; the BF16 scale product is applied at post-process.

Setting `lane = 0` activates all lanes. Values 1–31 restrict execution to the specified lane count — useful for partial-tile fallbacks at the trailing edge of a matrix.

FLAGS FIELD

Bit	Name	Description
[5]	`findemax`	Update the `e_max` register for output normalization (used by softmax).
[4]	`accm`	Accumulate into `dest` (default is overwrite).
[3]	`w_scale`	Apply the weight scale factor during MAC.
[2:0]	reserved	Must be zero.

OPERATION

```
DISPATCH gemm_or_gemv:
    shape  <- CONST_CACHE[shape_ptr_addr]       // (M, N, K)
    size   <- CONST_CACHE[size_ptr_addr]        // tile sizes
    FOR each tile (m, n, k) IN iteration_space(shape, size):
        IF opcode == GEMM:
            ARRAY_PRELOAD_WEIGHTS(tile_weights)     // Weight Stationary
            STREAM_ACTIVATIONS(src_addr + offset)
        ELSE:                                       // GEMV
            PARTITION_WEIGHTS_OVER_4_CORES(tile_weights)
            STREAM_ACTIVATIONS(src_addr + offset)
```

```
        ENDIF

        IF flags.accm = 1:
            acc <- L2[dest_reg + offset]
        ELSE:
            acc <- 0
        ENDIF

        acc <- acc + INT_MAC(activations, weights) * (S_fmap * S_weight)
        IF flags.findemax = 1:
            EMAX_REG <- max(EMAX_REG, exp(acc))
        ENDIF
        L2[dest_reg + offset] <- acc
    ENDFOR
END DISPATCH
```

FLAGS AFFECTED

`e_max` is updated when `findemax = 1`. No other architectural flags are affected.

EXCEPTIONS

- `#UD` — **undefined instruction.** Raised if `shape_ptr` or `size_ptr` reference an uninitialised Constant Cache entry.
- `#RSV` — reserved bits `[2:0]` must be zero; decoder asserts on non-zero.

USE CASE

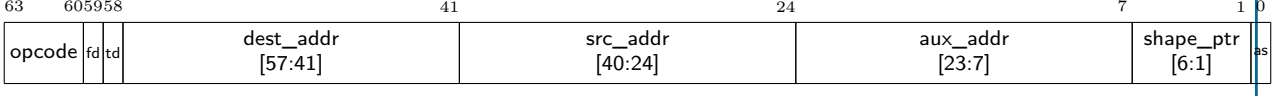
- `GEMM` — prefill. Q/K/V projection, FFN up/down projection over a sequence tile.
- `GEMV` — autoregressive decoding. Every projection in the single-token-per-step loop.

MEMCPY — Move a block between host and device / L2 and L2

OPCODE & INSTRUCTION SUMMARY

Opcode	Instruction	Op/En	Description
`4'h2` *body*	`MEMCPY from_dev, to_dev, dest, src, aux, shape_ptr, async`	B	Bulk data move. `from_dev/to_dev` determines direction and path.

BIT-LEVEL ENCODING (TYPE B)



DESCRIPTION

Moves a shape-defined block between two address spaces. The `from_device/to_device` pair encodes one of four source–destination combinations; the decoder expands the pair into the 8-bit `data_route_e` enum consumed by the back-end DMA engines. See Table 6.1.

`aux_addr` carries the host-side DDR offset when the transfer crosses ACP; for NPU-to-NPU transfers it is ignored.
When `async = 0` the dispatcher stalls the front end until the completion fence returns. When `async = 1` the instruction retires immediately and completion is surfaced to the host via `STAT_OUT`.

SUPPORTED (FROM_DEVICE, TO_DEVICE) COMBINATIONS

from_dev	to_dev	Path
1 (Host)	0 (NPU)	ACP → L2 cache. Used for weight / input loads.
0 (NPU)	1 (Host)	L2 cache → ACP. Returns output tokens / KV entries.
0 (NPU)	0 (NPU)	Intra-L2 block move (on-device rearrangement).
1 (Host)	1 (Host)	*Reserved* — Control Unit raises `#UD`.

OPERATION

```
DISPATCH memcpy:
    route <- ENCODE_ROUTE(from_device, to_device)   // data_route_e
    shape <- CONST_CACHE[shape_ptr_addr]
    CASE route OF
        from_host_to_L2: DMA_ACP_READ(dest_addr, src_addr, aux_addr, shape)
        from_L2_to_host: DMA_ACP_WRITE(dest_addr, src_addr, aux_addr, shape)
        from_L2_to_L1_GEMM: DMA_L1_FWD(dest_addr, src_addr, GEMM, shape)
        from_L2_to_L1_GEMV: DMA_L1_FWD(dest_addr, src_addr, GEMV, shape)
        OTHERWISE: RAISE #UD
    ENDCASE

    IF async = 1:
        SIGNAL fence_id TO STAT_OUT    // fire-and-forget
        RETURN
    ELSE:
        WAIT fence_id
    ENDIF
END DISPATCH
```

EXCEPTIONS

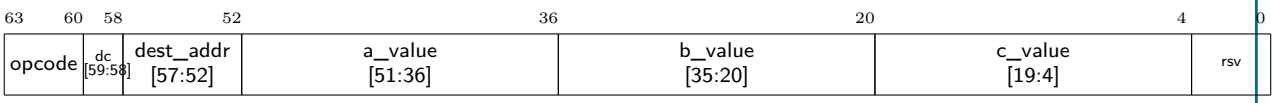
- `#UD` on reserved (Host, Host) combination.
- `#RSV` on shape pointer referencing an uninitialised Constant Cache entry.
- `#AXI` if the HP / ACP read returns an error response (SLVERR / DECERR). Surfaced via `STAT_OUT`.

MEMSET — Constant Cache write (shape / size / scale)

OPCODE & INSTRUCTION SUMMARY

Opcode	Instruction	Op/En	Description
`4'h3` *body*	`MEMSET dest_cache, dest_addr, a, b, c`	C	Write three 16-bit values to a Constant Cache entry.

BIT-LEVEL ENCODING (TYPE C)



DESCRIPTION

Writes directly into the Constant Cache. This is the *only* opcode that does not touch the L2 cache. `dest_cache` selects which bank is targeted: 0 = `fmap_shape`, 1 = `weight_shape` (2–3 reserved).
The three value slots `a / b / c` let a single instruction write an entire (M, N, K) tuple in one shot; this is the idiomatic use at layer boundaries.

OPERATION

```
DISPATCH memset:
    CASE dest_cache OF
        0: CONST_CACHE.fmap_shape[dest_addr]   <- (a_value, b_value, c_value)
        1: CONST_CACHE.weight_shape[dest_addr] <- (a_value, b_value, c_value)
        OTHERWISE: RAISE #UD
    ENDCASE
END DISPATCH
```

FLAGS AFFECTED

None.

EXCEPTIONS

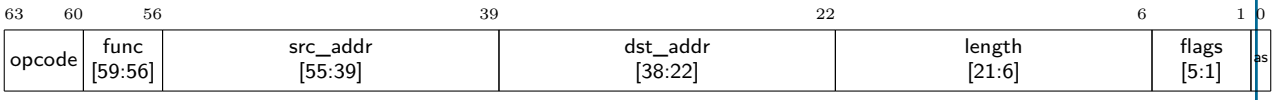
- `#UD` on reserved `dest_cache` values (2–3).
- `#RSV` on non-zero reserved bits `[3:0]`.

CVO — Complex Vector Op — transcendentals & reductions via SFU

OPCODE & INSTRUCTION SUMMARY

Opcode	Instruction	Op/En	Description
`4'h4` *body*	`CVO func, src, dst, length, flags, async`	D	Run an SFU function (exp, sqrt, gelu, sin, cos, reduce, scale, recip) over a length-N vector.

BIT-LEVEL ENCODING (TYPE D)



DESCRIPTION

Dispatches an SFU (“Scalar Function Unit”) operation on a `length`-element BF16 vector. Functions are implemented via a combination of CORDIC iterations and LUT interpolation; in steady state the SFU retires one element per cycle.
The `flags.sub_emax` bit enables the numerical-stability pre-subtraction used by softmax ($e^{x-e_{\max}}$). The `flags.recip_scale` bit turns `CVO_SCALE` into a division by flipping the scalar at the input (softmax denominator divide is implemented this way).

FUNCTION CODES

Function	Code	Semantics	Use
CVO_EXP	4'h0	e^x	Softmax
CVO_SQRT	4'h1	$\sqrt{x}$	RMSNorm
CVO_GELU	4'h2	$\mathrm{gelu}(x)$	FFN nonlinearity
CVO_SIN	4'h3	$\sin(x)$	RoPE
CVO_COS	4'h4	$\cos(x)$	RoPE
CVO_REDUCE_SUM	4'h5	$\sum_i x_i$	Softmax denominator
CVO_SCALE	4'h6	$\alpha \cdot x$	Bias / scale fuse
CVO_RECIP	4'h7	$1/x$	Softmax / RMSNorm
4'h8–4'hF	—	*Reserved*	—

FLAGS FIELD

Bit	Name	Description
[4]	sub_emax	Subtract e_max before computing.
[3]	recip_scale	Use the reciprocal of the scalar.
[2]	accm	Accumulate into dst.
[1:0]	reserved	Must be zero.

OPERATION

```
DISPATCH cvo:
    FOR i FROM 0 TO length - 1 DO
        x <- L2[src_addr + i]
        IF flags.sub_emax = 1:
            x <- x - EMAX_REG
        ENDIF
        CASE func OF
            CVO_EXP:        y <- exp(x)
            CVO_SQRT:       y <- sqrt(x)
            CVO_GELU:       y <- 0.5 * x * (1 + tanh(sqrt(2/pi) * (x + 0.044715*x^3)))
            CVO_SIN:        y <- sin(x)
            CVO_COS:        y <- cos(x)
            CVO_REDUCE_SUM: acc <- acc + x; CONTINUE
            CVO_SCALE:      y <- (flags.recip_scale ? 1/scalar : scalar) * x
            CVO_RECIP:      y <- 1 / x
            OTHERWISE:      RAISE #UD
        ENDCASE

        IF flags.accm = 1:
            y <- y + L2[dst_addr + i]
        ENDIF
        L2[dst_addr + i] <- y
    ENDFOR

    IF func = CVO_REDUCE_SUM:
        L2[dst_addr] <- acc
    ENDIF
    IF async = 1: SIGNAL fence_id; RETURN
    ELSE:         WAIT fence_id
    ENDIF
END DISPATCH
```

FAST PATH

If the preceding instruction is a `GEMV` whose `dest_reg` matches a special *bypass tag*, the SFU consumes the output directly from the GEMV FIFO and skips the L2 round trip. Figure 5.1 illustrates the data path.

EXCEPTIONS

- `#UD` on reserved function codes (`4'h8`–`4'hF`).
- `#RSV` on non-zero reserved flag bits.

5.1 CVO Pipeline Internal Structure

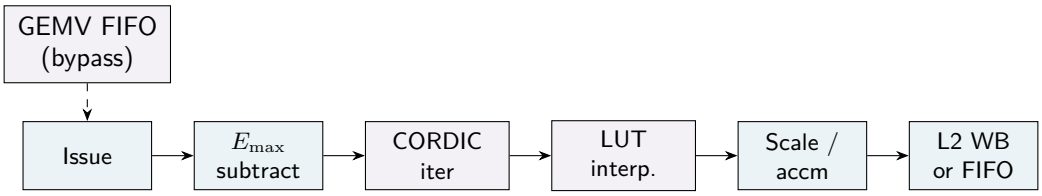


Figure 5.1: CVO pipeline internals. Dashed red arrow is the fast-path from a preceding `GEMV`.

MEMORY ROUTING & ADDRESS SPACES

6.1 Data-Route Encoding (`data_route_e`)

The `from_device/to_device` pair on `MEMCPY` expands inside the Control Unit into an 8-bit route enum. Every DMA controller in the back end consumes this single byte rather than re-interpret the two encoding bits.

Route symbol	Hex	Direction	Path
`from_host_to_L2`	`8'h01`	Ingress	Host DDR4 → L2 cache
`from_L2_to_host`	`8'h10`	Egress	L2 cache → Host DDR4
`from_L2_to_L1_GEMM`	`8'h12`	Compute	L2 → GEMM L1
`from_L2_to_L1_GEMV`	`8'h13`	Compute	L2 → GEMV L1
`from_L2_to_CVO`	`8'h14`	Compute	L2 → SFU
`from_GEMM_res_to_L2`	`8'h21`	Writeback	GEMM result → L2
`from_GEMV_res_to_L2`	`8'h31`	Writeback	GEMV result → L2
`from_CVO_res_to_L2`	`8'h41`	Writeback	SFU result → L2

Table 6.1: `data_route_e` encoding and associated datapath.

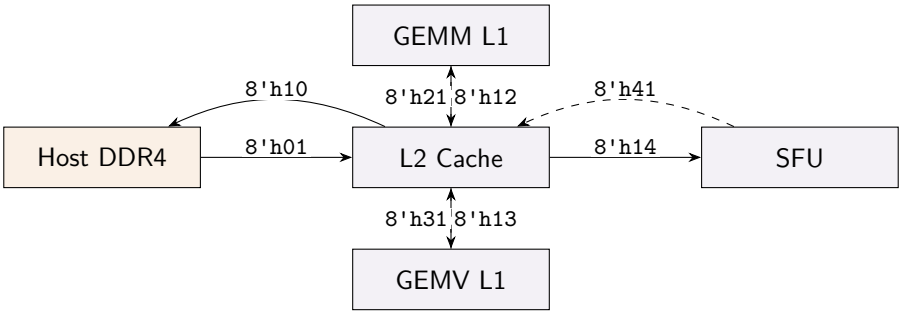


Figure 6.1: Visual summary of Table 6.1. Solid arrows are ingress / compute reads; dashed arrows are writeback paths.

6.2 Constant Cache & Pointer Registers

The Constant Cache is 64 entries deep and holds shape metadata, tile sizes, and scale factors. ISA-visible pointers into this cache are 6 bits wide (since $\log_2 64 = 6$).

Pointer	Width	Content
`shape_ptr_addr`	6	(M, N, K) tuple for the tile dimension.
`size_ptr_addr`	6	Tile sizes, loop bounds, stride.
`ptr_addr_t`	6	Generic index into the 64-entry Constant Cache.

Pointer entries are populated by `MEMSET`. A canonical layer prologue is:

```
MEMSET fmap_shape,   idx=0, a=M, b=N, c=K
MEMSET weight_shape, idx=0, a=scale, b=zero_point, c=0
GEMM   dest=..., src=..., flags={w_scale=1}, size_ptr=0, shape_ptr=0, lane=0
```

6.3 Address Space

Field	Width	Address space
`dest_addr` / `src_addr`	17	2^{17} entries = 128 K, indexed by L2 cache block.
`aux_addr`	17	MEMCPY auxiliary address (e.g., host DDR offset in units of the block size).

With an on-device block size of 128 bits (16 B) on KV260, the 17-bit field yields a **2 MB** linear L2 address space.

Caution

`aux_addr` is measured in **block units**, not bytes. A host-side byte offset must be divided by 16 before loading into `aux_addr`.

DATA FLOW & COMPLETION

7.1 Per-Instruction Dataflow Summary

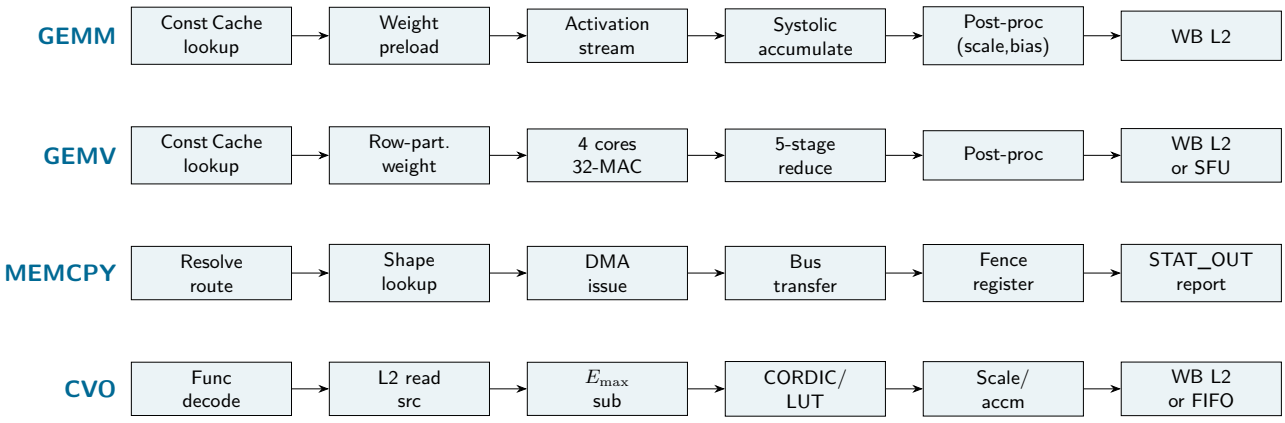


Figure 7.1: Per-instruction dataflow summary. Each row is one instruction's journey through the NPU from dispatch to writeback.

7.2 Hazards and Completion Tracking

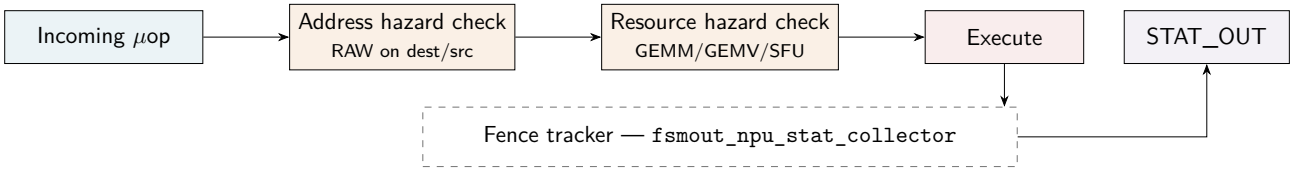


Figure 7.2: Hazard pipeline. Each μop passes an address-hazard and a resource-hazard check before executing; `async` completion is reported via the Fence tracker into `STAT_OUT`.

7.3 Asynchronous Completion Protocol

For `async = 1` instructions (MEMCPY or CVO), the dispatcher retires immediately and the host tracks completion through `STAT_OUT`. Figure 7.3 shows the wire-level sequence.

7.4 Fence Tracker State Machine

The fence tracker is a small FSM inside the Dispatcher with one instance per outstanding `async` instruction. Each instance owns a single `fence_id` slot in `STAT_OUT`.

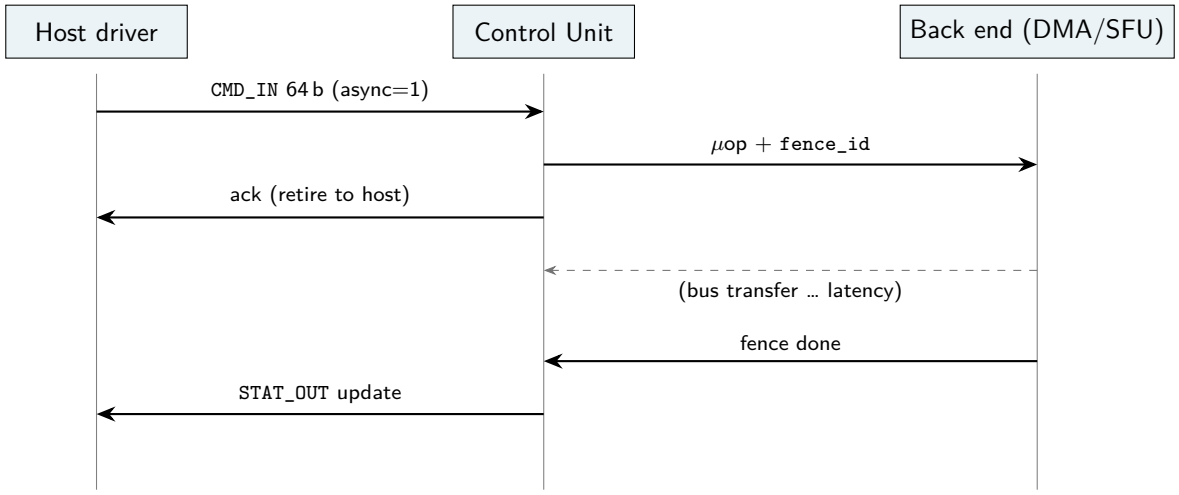
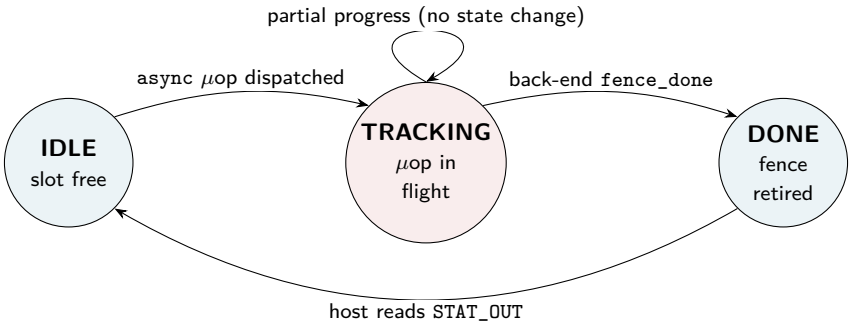


Figure 7.3: Asynchronous completion sequence. The host can issue the next instruction as soon as it receives the retire ack; true completion arrives later via `STAT_OUT`.



Global reset returns every tracker slot to **IDLE**.
There are 16 slots in hardware — one per outstanding `fence_id`.

Figure 7.4: Per-slot fence-tracker state machine. A slot transitions IDLE→TRACKING the moment an `async` μop is accepted by the back end, and TRACKING→DONE when the back end asserts `fence_done`. The host reading `STAT_OUT` returns the slot to IDLE.

PROGRAMMING GUIDE

This chapter walks through representative instruction streams that a host driver emits.

8.1 Example 1 — Single GEMM tile (Q projection for one sequence tile)

Goal: compute $Q = XW_Q$ for a tile of shape $(M, N, K) = (64, 256, 256)$.

```
// 1. Preload shape + size pointers
MEMSET fmap_shape,   idx=0, a=64,  b=256, c=256     // (M, N, K)
MEMSET weight_shape, idx=0, a=scale_wq, b=zp_wq, c=0

// 2. Load weights if not already streaming
MEMCPY from_device=1, to_device=0, dest=L2_W_BASE,
       src=HOST_WQ_OFFSET, aux=HOST_BASE,
       shape_ptr=0, async=0

// 3. Dispatch the GEMM
GEMM   dest_reg=L2_Q_BASE, src_addr=L2_X_BASE,
       flags={findemax=0, accm=0, w_scale=1},
       size_ptr=0, shape_ptr=0, lane=0                // lane=0 -> all lanes
```

8.2 Example 2 — Autoregressive step (GEMV + softmax + RoPE)

```
// --- Q, K, V projection ---
GEMV dest=L2_Q, src=L2_X, flags=w_scale, size_ptr=0, shape_ptr=0
GEMV dest=L2_K, src=L2_X, flags=w_scale, size_ptr=1, shape_ptr=1
GEMV dest=L2_V, src=L2_X, flags=w_scale, size_ptr=2, shape_ptr=2

// --- RoPE rotation on Q and K ---
CVO  func=CVO_COS, src=L2_Q, dst=L2_Q_R, length=D_HEAD, flags=0
CVO  func=CVO_SIN, src=L2_K, dst=L2_K_R, length=D_HEAD, flags=0

// --- Attention score + softmax ---
GEMV dest=L2_SCORES, src=L2_Q_R, flags={findemax=1, w_scale=0},
     size_ptr=3, shape_ptr=3                        // Q . K^T
CVO  func=CVO_EXP, src=L2_SCORES, dst=L2_SCORES,
     length=SEQ_LEN, flags=sub_emax
CVO  func=CVO_REDUCE_SUM, src=L2_SCORES, dst=L2_DENOM,
     length=SEQ_LEN, flags=0
CVO  func=CVO_SCALE, src=L2_SCORES, dst=L2_SCORES,
     length=SEQ_LEN, flags=recip_scale              // divide by denom

// --- Attention-weighted V gather ---
GEMV dest=L2_CTX, src=L2_SCORES, flags=w_scale,
     size_ptr=4, shape_ptr=4                        // V . P
```

8.3 Example 3 — Weight hot-swap (async prefetch overlapped with compute)

```
// Fire and forget -- continues to the next instruction immediately
MEMCPY from_device=1, to_device=0,
       dest=L2_W_NEXT_BASE,
       src=HOST_W_LAYER_NP1_OFFSET,
       aux=HOST_BASE,
       shape_ptr=5, async=1
```

```
// ... overlap with layer N GEMMs ...
GEMM   dest=..., src=..., ...
GEMM   dest=..., src=..., ...

// Before starting layer N+1, the scheduler knows the MEMCPY fence has
// retired, so no explicit barrier is required.
```

8.4 Performance Rules of Thumb

- Amortize `MEMSET` — one at the top of each layer is enough; avoid per-tile MEMSETs.
- Prefer `async = 1` for MEMCPY when the transfer is followed by ≥ 2 compute instructions that do not depend on its result. The scheduler's fence tracker will stall automatically when the consumer arrives.
- Keep `parallel_lane = 0` unless you are deliberately executing a trailing partial tile — the dispatcher's resource allocator is more efficient when given the full lane set.
- Fuse softmax: on the same SFU dispatch, chain `CVO_EXP` $\rightarrow$ `CVO_REDUCE_SUM` $\rightarrow$ `CVO_SCALE` (`recip_scale`) so the pipeline drains once rather than three times.
- If a GEMV feeds directly into a CVO, pre-tag the GEMV `dest` with the SFU-bypass marker so the L2 round trip is skipped.

8.5 Roofline Model

Figure 8.1 plots the classical roofline for pccx v002: the ceiling is the 800 GMAC/s peak compute of 2000 DSP48E2 slices running at 400 MHz in 9×4 packed mode, and the slope is the 12.8 GB/s DDR4 bandwidth delivered through the 4 AXI-HP ports. The three operators the ISA exposes land in distinct regimes:

- **GEMM** (prefill) — weight reuse of $\sim 256\times$ inside one tile pushes arithmetic intensity above the ridge point; the systolic array runs compute-bound.
- **GEMV** (decode) — every weight is used once per token, so AI is ~ 0.5 MAC/byte and throughput tracks the memory slope. Speculative decoding and tied-embedding reuse are the only ways to cross the ridge.
- **CVO** — dominated by the activation/scale memory traffic; runs left of the ridge but is a small fraction of total cycles.

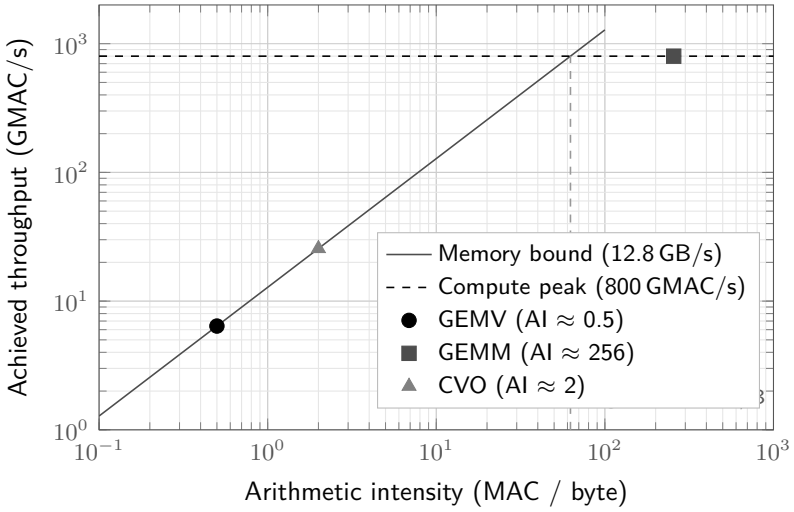


Figure 8.1: Roofline model for pccx v002. GEMM sits at the compute ceiling; GEMV is firmly memory-bound. CVO is off-critical-path but also bandwidth-limited.

OPCODE MAP

A.1 Single-Byte Opcode Quick Reference

opcode[3:0]	Mnemonic	Body layout	Primary back-end	Family
`4'h0`	GEMV	Type A	4 × 32-MAC streaming cores.	Compute
`4'h1`	GEMM	Type A	32 × 32 systolic array.	Compute
`4'h2`	MEMCPY	Type B	ACP + HP DMA engines.	Memory
`4'h3`	MEMSET	Type C	Constant Cache (bypasses L2).	Memory
`4'h4`	CVO	Type D	SFU (BF16 CORDIC + LUT).	SFU
`4'h5`–`4'hF`	—	—	*Undefined.*	—

CHAPTER B

DATA-ROUTE ENUM REFERENCE

See Table 6.1 on the corresponding page for the full text; this appendix reproduces only the 8-bit hex values for quick lookup.

Hex	Route symbol
8'h01	from_host_to_L2
8'h10	from_L2_to_host
8'h12	from_L2_to_L1_GEMM
8'h13	from_L2_to_L1_GEMV
8'h14	from_L2_to_CVO
8'h21	from_GEMM_res_to_L2
8'h31	from_GEMV_res_to_L2
8'h41	from_CVO_res_to_L2

CVO FUNCTION-CODE REFERENCE

Function	Code	Semantics	Use
`CVO_EXP`	`4'h0`	e^x	Softmax numerator
`CVO_SQRT`	`4'h1`	$\sqrt{x}$	RMSNorm denominator
`CVO_GELU`	`4'h2`	$\mathrm{gelu}(x)$	FFN nonlinearity
`CVO_SIN`	`4'h3`	$\sin(x)$	RoPE
`CVO_COS`	`4'h4`	$\cos(x)$	RoPE
`CVO_REDUCE_SUM`	`4'h5`	$\sum_i x_i$	Softmax denominator
`CVO_SCALE`	`4'h6`	$\alpha \cdot x$	Bias / scale fuse
`CVO_RECIP`	`4'h7`	$1/x$	Softmax / RMSNorm
`4'h8`–`4'hF`	—	*Reserved*	—

GLOSSARY

Term	Definition
`accm`	GEMM/GEMV flag (bit [4] of the flags field) — when set, the result is **added** to the destination; when clear, the destination is overwritten.
ACP	Accelerator Coherency Port — a cache-coherent AXI port on the Zynq UltraScale+ PS side that the NPU uses for host-visible DMA.
μop	Micro-operation: a decoded, back-end-specific packet emitted by the Dispatcher. One architectural instruction fans out to one or more μops.
Async / `async`	MEMCPY / CVO flag (bit [0] of the bit-0 flag) — when set, the Control Unit retires the instruction immediately and reports completion through `STAT_OUT` instead of blocking.
AXI-Lite	Lightweight subset of the ARM AMBA AXI4 protocol used here for the 64-bit `CMD_IN` / `STAT_OUT` control path. Single-beat, non-burst.
AXI-HP	High-Performance AXI port on the KV260's PS — the NPU claims four 128-bit HP ports for bulk data movement between DDR4 and L2.
BF16	"Brain floating-point" half-precision format: 1 sign / 8 exponent / 7 mantissa bits. Used for activations, KV cache, and SFU intermediates. Chosen for its IEEE-float-compatible exponent range without the FP16 precision cliff.
BFP	Block Floating Point — an activation-level numeric format where a group of INT8 mantissas shares one BF16 exponent (S_{fmap}). Folds the MAC product into a single post- process scalar multiply.
BRAM / URAM	Xilinx on-chip RAM primitives: BRAM is 36 Kb / block, URAM is 288 Kb / block. v002 uses URAM as the unified L2 backing store and BRAM for per-core L1.
CDC	Clock Domain Crossing — the boundary between the 250 MHz AXI domain and the 400 MHz compute domain. Crossed by XPM async FIFOs; see Figure 2.3.
CORDIC	"COordinate Rotation DIgital Computer" — an iterative shift-and-add algorithm that evaluates trigonometric and hyperbolic functions without a hardware multiplier. Used inside the SFU for `CVO_SIN`, `CVO_COS`, `CVO_EXP`, `CVO_SQRT`.
CVO	Complex Vector Op — the family of non-linear scalar/vector activations (softmax, GELU, RMSNorm, RoPE) dispatched to the SFU. See `CVO_*` function codes in Appendix C.
`dest_cache` (`dc`)	MEMSET operand: 2-bit selector for which Constant Cache bank receives the write (0=`fmap_shape`, 1=`weight_shape`, 2–3 reserved).
DSP48E2	Xilinx UltraScale+ DSP slice: 30×18 signed multiplier plus 48-bit accumulator. v002 drives it in 9×4 packed W4A8 mode for two MACs / cycle per slice.
Dispatcher	Control-unit block that routes decoded μops to the back-ends (GEMM, GEMV, SFU, DMA). Also runs the fence tracker.
$E_{\max}$	Maximum exponent found during a softmax numerator pass; subtracted before the `CVO_EXP` to keep the sum finite.
Fence tracker	State machine inside the Dispatcher that tracks in-flight `async` instructions and writes `STAT_OUT` when the back-end fence retires; see Figure 7.4.
`findemax`	GEMV flag (bit [5]) — update the $E_{\max}$ register during this MAC epoch; consumed by a following `CVO_EXP`.
GEMM	General Matrix-Matrix Multiply (prefill-dominant). v002 uses a 32×32 systolic array with a cascade break at row 16.
GEMV	General Matrix-Vector Multiply (decode-dominant). v002 has 4 GEMV cores each with 32-MAC streaming and a 5-stage reduction tree; see Figure 3.4.
HP port	See AXI-HP.
INT4 / INT8	4-bit / 8-bit signed integer. INT4 is the weight quantization target (W4); INT8 is the activation quantization target (A8) in the W4A8 scheme.
KV cache	Key / Value cache — a per-layer BF16 buffer in L2 that stores attention keys and values across tokens during autoregressive decode.
Lane / `parallel_lane`	GEMM/GEMV operand (5 bits) that restricts execution to a subset of the 32 lanes. `lane = 0` activates all lanes.
L1 / L2	L1 = per-core weight buffer backed by URAM FIFOs. L2 = unified 1.75 MB central cache shared by GEMM, GEMV, SFU, DMA.
LUT (Xilinx)	6-input lookup table, the basic configurable-logic cell. v002 uses LUTs for GEMV dequantize pre-computation and for reduction tree stages 2–5.
MEMCPY	DMA move between two address spaces. Can be synchronous (blocking) or `async = 1` (fire-and-forget).

Term	Definition
MEMSET	Constant Cache scalar write. Used to push (M, N, K) and scale factors to pointer indices before a GEMM/GEMV.
opcode	4-bit field in bits [63:60] of every v002 instruction. See Appendix A for the full map.
PL / PS	Programmable Logic (FPGA fabric) / Processing System (ARM cores) — the two halves of the Zynq UltraScale+ SoC on KV260.
`recip_scale`	Reciprocal of the softmax denominator; supplied as an operand to the fused `CVO_SCALE` stage to complete the softmax division in one shot.
RoPE	Rotary Position Embedding — applies sin / cos rotations to Q and K vectors before attention. Expressed in pccx as paired `CVO_SIN` / `CVO_COS` dispatches.
`shape_ptr` / `size_ptr`	6-bit operand fields pointing into the Constant Cache for the (M, N, K) tuple and for the tile-size tuple. Populated by MEMSET.
SFU	Special Function Unit — the single-core back-end that executes CVO instructions. Mixes CORDIC iterations with LUT-based interpolation. BF16 datapath.
`src_addr` / `dest_addr`	17-bit operand fields addressing 128-bit blocks in L2 (2 MB linear).
STAT_OUT	Host-side status register written by the Dispatcher once an `async` instruction's fence retires. AXI-Lite read-only.
Systolic array	Regular grid of PEs used for GEMM. v002 pins weights inside PEs (weight-stationary) while activations march left-to-right and partial sums march top-to-bottom. See Figure 3.2.
Weight Stationary / Weight Streaming	Two systolic schedules. **Stationary** (GEMM) reuses the same weight across many activations; **Streaming** (GEMV) streams fresh weights every cycle because each weight is used once per token.
W4A8	The pccx v002 quantization scheme: 4-bit weights, 8-bit activations, BF16 scale factors applied at post-process.
XPM	Xilinx Parameterized Macro — vendor-provided primitives for FIFOs, CDC, memory etc. The NPU uses `xpm_fifo_async` in independent-clock mode for the PS ↔ PL crossing.

CHAPTER E
REVISION HISTORY

Revision	Date	Notes
v002	2026-04-21	Initial Software Developer's Manual release. Classical reference-manual-style per-instruction reference blocks, bit-level Type A/B/C/D `bytefield` diagrams, print-quality TikZ architectural illustrations (system context, NPU top-level, memory hierarchy, URAM allocation, CDC FIFO, DSP48E2 datapath, systolic array, GEMV streaming, decode/dispatch, hazard tracker, async completion sequence, CVO pipeline). Numbers mirror `isa_pkg.sv` in the RTL repo.
v002-r2	2026-04-22	Reference-manual polish pass. Added wire-level AXI-HP read-handshake timing diagram (Figure 2.4), GEMV 5-stage reduction tree (Figure 3.4), fence-tracker state machine (Figure 7.4), and a roofline performance model (Figure 8.1). New Appendix D glossary (40 terms).