ALP LAB

DeepX DX-M1 First Inference Walkthrough

AN-009 · E1M-X V2N-M1 (V2N + DX-M1)

Document Number: AN-009 **Revision:** 0.3 **Date:** July 2026 **Status:** Preliminary

alplab.ai

1 Scope

Bring the on-module **DeepX DX-M1** 25-TOPS AI accelerator from cold-power to running a first inference. The DX-M1 is PCIe-attached on the V2N-M1 SoM, multiplexed with the carrier-board M.2 Key M slot via an on-module switch (see **HG-V2N-M1-001 §6.7**).

This AN assumes the V2N base is already running (you've completed **QS-E1M-X-EVK-001** and at least one V2N example from **AN-008**).

Audience	ML engineers shipping high-throughput vision-AI firmware on V2N-M1.
Prerequisites	V2N-M1 SoM on the E1M-X EVK, V2N base bring-up complete, Linux on the A55 cluster (the DX-M1 driver requires a Linux PCIe stack), DeepX **DX-COM™** host tool installed.
Outcome	A correct DX-M1 bring-up sequence in your firmware, and a first inference dispatched through `<alp/inference.h>`.
Time	60 minutes (first-time DX-COM model-compile is the long pole).
Source	`docs/bring-up-v2n-m1.md` in alp-sdk vendor DX-COM documentation from DeepX.

Table 1 Scope summary

Warning: **Nothing in this note has been run on DX-M1 silicon.** The host-side bring-up driver, the rail drivers, and the DEEPX inference path are all code-complete and bench-unverified: the drivers compile and pass null-argument smoke tests, the reference example is `build_only` on V2N-M1, and the Linux-side inference body header-compiles against the real `dx_rt` headers but has never been linked against a Yocto sysroot or run against a DX-M1. Treat every number and every sequencing claim below as paper-correct until Alp Lab's verification sweep lands.

2 Architecture

The V2N's single PCIe Gen3 controller (1 or 2 lanes — the RZ/V2N has no 4-lane PCIe) is shared between the **on-module DX-M1** and the **external M.2 Key M slot** via a pair of on-module passive muxes (Diodes **PI3DBS12212A**). Software selects which endpoint is active at boot or at runtime.

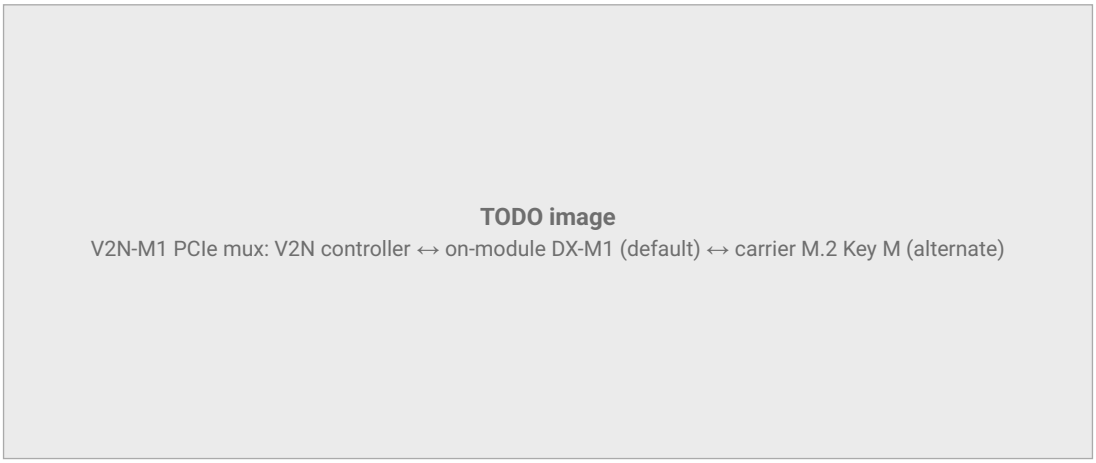

Figure 1 V2N-M1 PCIe mux: V2N controller ↔ on-module DX-M1 (default) ↔ carrier M.2 Key M (alternate)

Warning: Only one PCIe endpoint can be active at a time. Switching to the M.2 slot disables the on-module DX-M1, and vice versa.

Note: The mux control lines, the `M1_RESET` line, and the PCIe routing between them are **module-internal**. They are not brought out to the E1M-X edge connector and are not published in the V2N-M1 datasheet. The edge connector is the customer contract: it is the interface Alp Lab's form-fit-function guarantee covers across hardware revisions of a SoM SKU. Internal routing

can change in a bridge re-spin without breaking that guarantee, so drive the DX-M1 through `deepx_dxm1_bring_up()` rather than against any assumed internal pin assignment. I²C addresses on the shared `BRD_I2C` bus **are** part of the contract you need — they are given below so you can avoid collisions with your own devices.

3 Bring-up Sequence

The DX-M1 bring-up is a strictly ordered sequence. Getting the order wrong — in particular releasing `M1_RESET` before the PCIe muxes are routed to the DEEPX path — is the most common way to end up with rails that look healthy and a link that never trains.

The order is:

1. Confirm the three DEEPX rail PMICs acknowledge on `BRD_I2C`.
2. Bring up the 0.75 V DEEPX rail on the secondary PMIC's CH2.
3. Route the PCIe muxes to the DEEPX path, **then** release `M1_RESET`.
4. Hand off to Linux.

3.1 Step 1 — Confirm the DEEPX rail PMICs ACK

Three TPS628640 buck instances power the DEEPX-specific rails. They self-regulate to their factory OTP voltages with no host writes; firmware only confirms the parts are present and responding.

Address	Voltage	Rail	Scope
`0x44`	1.05 V	`DDR5_VDD`	DEEPX DDR5 VDD bank. V2N-M1 only.
`0x48`	0.85 V	`VDD0V85_LPDDR`	DEEPX LPDDR + DDR core. V2N-M1 only.
`0x4F`	0.50 V	`DDR5_VDDQ_0V5`	DEEPX DDR5 IO termination. V2N-M1 only.
`0x4D`	0.60 V	`LPD4x_0V6`	Renesas LPDDR4X. V2N-common, **assembly option** — when unpopulated this rail is supplied by the primary PMIC or the secondary PMIC instead.

Table 2 TPS628640 instances on `BRD_I2C`

Warning: On the three DEEPX rails (`0x44` / `0x48` / `0x4F`), an `ALP_ERR_NOT_READY` from `tps628640_init()` is a **real fault**, not an assembly option. All three are populated on every V2N-M1. A part that does not ACK is either powered down (check its EN line) or address-strapped wrong; probe the rail directly. Only the `0x4D` Renesas LPDDR4X instance is optional, and only on the V2N base — a skip there is expected, a skip on the DEEPX three is a board fault.

3.2 Step 2 — Bring up the 0.75 V DEEPX rail

The DEEPX DDR / NPU rail is CH2 (phases 3 + 4) of the secondary PMIC, a Renesas DA9292 at I²C `0x1E`. On the V2N base this channel stays disabled; on V2N-M1 it must be brought up before the DX-M1 leaves reset.

The SDK helper `da9292_v2n_m1_enable_deepx_rail()` performs the whole channel sequence: it sets CH2's voltage-select register to 0.75 V, **reads it back** to confirm the write took, sets `CH2_EN`, and polls `CH2_PG` until the rail is in regulation. Expect `ALP_OK` within roughly 5 ms. An `ALP_ERR_TIMEOUT` means the rail is not coming up — typically a downstream short on the 0.75 V plane.

3.3 Step 3 — Route the muxes, then release `M1_RESET`

Order matters here. The SDK's `deepx_dxm1_bring_up()` sequencer, from `<alp/chips/deepx_dxm1.h>`, does these in the correct order in a single call: it routes the PI3DBS12212A muxes from their quiescent OFF state to the DEEPX path (a glitch-free transition), **then** releases `M1_RESET`, then waits `boot_us` for the DEEPX boot ROM to execute before returning. PCIe link training starts a few hundred microseconds after the reset release.

Its pre-conditions are the caller's responsibility, and they are steps 1 and 2: all DEEPX rails stable at their target voltages for at least 1 ms, and the Renesas PCIe controller initialised but not yet attempting link-up. The sequencer does not orchestrate rail bring-up itself.

`M1_RESET` is active-low on V2N-M1, which is the driver's default — no polarity override is required.

> **Note:** If the rails come up but PCIe never trains, suspect a reset-polarity mismatch first. If the link trains but the kernel driver reports BAR errors, the muxes are likely on the wrong path — confirm the board's DEEPX-side mux state matches what you passed to `deepx_dxm1_init()`.

3.4 Step 4 — Hand off to Linux

The DX-M1 driver requires a Linux PCIe stack, so the accelerator is driven from the A55 cluster. With the `meta-deepx-m1` Yocto layer wired in, the `dx_rt_npu_linux_driver` opens the PCIe device at `lspci` time and `dxrt_init()` succeeds from user space; from there you load a `.dxnn` model and run inferences.

4 Compile the Model with DX-COM

The DeepX `dxcom` host compiler (DX-COM) converts ONNX models into a DX-M1-optimised `.dxnn` model file. The output is a single `.dxnn` file — the format the SDK loads as `ALP_INFERENCE_MODEL_DXNN`. It bundles the DX-M1 NPU executable, the quantised INT8 weights, and the input / output tensor metadata (shapes + scale factors).

DX-COM is vendor tooling: Alp Lab does not redistribute it, does not pin its version, and its invocation is not reproduced here. Obtain the compiler and its documentation from DeepX and follow the vendor's instructions. The DX-M1 is optimised for INT8 CNN inference (ResNet, MobileNet, YOLO and similar topologies); check DeepX's release notes for the authoritative per-op coverage table before committing to a model architecture.

5 Load and Run

The portable surface is `<alp/inference.h>`: open a handle with `backend = ALP_INFERENCE_BACKEND_DEEPX_DXM1` and `format = ALP_INFERENCE_MODEL_DXNN`, then invoke. `examples/v2n/v2n-m1-deepx-inference/` in alp-sdk is the reference.

Where that dispatches depends on the core:

- On the **A55 cluster under Linux / Yocto** this is the real path. The SDK dispatches to DEEPX's `dx_rt` runtime, which the customer pulls from `github.com/DEEPX-AI` at integration time — it ships under DeepX's customer-only licence and is not vendored into the SDK. This body is code-complete and header-compiles against the real `dx_rt` headers, but has not been linked against an RZ/V Yocto sysroot or run on a DX-M1.
- On the **Zephyr `m33_sm` core** the dispatcher returns `NOSUPPORT`. The M33 is a supervisor, not an inference host; the DX-M1 driver requires a Linux PCIe stack.

> **Warning:** Alp Lab publishes no Python binding for the DX-M1. There is no `alp.deepx` module, no `alp-pcie-select` command-line tool, and no Alp-defined `/dev` node for the accelerator. From Alp Lab's side the interfaces are the C `<alp/inference.h>` surface and the `<alp/chips/deepx_dxm1.h>` host sequencer; everything else in user space comes from DeepX's own runtime and is documented by DeepX.

6 Verifying the Bring-up

Alp Lab's own bring-up regression list for the V2N-M1 delta, in order — a useful checklist for your board:

1. Power-on idle current under 500 mA (the DX-M1 adds roughly 150 mA static).
2. Primary and secondary PMIC status clean: no thermal warning, no event latches.
3. Secondary PMIC CH2 in regulation (`da9292_get_status().ch2_pg == true`).
4. All three DEEPX TPS628640 instances ACK at their addresses.

5. `lspci` lists the DEEPX device.
6. `dxrt_init()` returns success.
7. A reference DeepX inference application runs to completion.

7 Parallel Inference (DRP-AI3 + DX-M1)

The V2N-M1 carries two accelerators: the RZ/V2N's on-die DRP-AI3 (4 dense TOPS, up to 15 sparse) and the DX-M1 (25 TOPS). Both are reachable through `<alp/inference.h>` — `ALP_INFERENCE_BACKEND_DRPAI` and `ALP_INFERENCE_BACKEND_DEEPX_DXM1` — and a handle is opened per backend, so an application can hold both open and pin different stages of a pipeline to each. A common intended pattern is lightweight per-frame pre-processing (e.g. RoI detection) on the DRP-AI3 feeding heavier classification on the DX-M1. The cross-accelerator demo is `examples/v2n/v2n-m1-ros-perception/`.

> **Note:** The SDK does **not** ship a cross-accelerator scheduler, and nothing keeps both accelerators saturated on your behalf. Concurrency, buffering, and load balancing between the two are your application's responsibility. As with the single-accelerator path, none of this has been measured on silicon — do not budget a frame rate from this section.

8 Troubleshooting

Symptom	Likely cause / fix
`tps628640_init()` returns `ALP_ERR_NOT_READY` on `0x44` / `0x48` / `0x4F`	A real fault, not an assembly option — these three rails are always populated on V2N-M1. The part is powered down (check EN) or address-strapped wrong. Probe the rail directly.
`da9292_v2n_m1_enable_deepx_rail()` returns `ALP_ERR_TIMEOUT`	The 0.75 V rail is not reaching regulation — typically a downstream short on the 0.75 V plane.
Rails come up but PCIe never trains	Most likely an `M1_RESET` polarity mismatch. Also confirm the muxes were routed to the DEEPX path **before** the reset release.
PCIe link trains but the kernel driver reports BAR errors	The muxes are on the wrong path — the E1M edge rather than DEEPX. Confirm the DEEPX-side mux state matches the board.
`lspci` shows no DeepX device	PCIe mux still pointed at the M.2 slot, or `deepx_dxm1_bring_up()` was never called.
DX-M1 link trains then drops	Power-supply ripple on the on-module 5V rail. Add a barrel-jack supply (12 V @ 5 A) instead of relying on USB-PD.
DX-COM rejects the model	Op not supported by your DX-COM version. Check the per-op compatibility table in DeepX's release notes; pre-process the unsupported op on the A55 if it's at the boundary.
DEEPX silicon flaky under load	Check the three TPS628640 rails with a scope — their factory OTP voltages assume a specific load envelope.
`dxrt_init()` fails	See DeepX's own troubleshooting documentation at `github.com/DEEPX-AI/dx_rt`.

Table 3 Common failures

9 References

- **Canonical bring-up**: `docs/bring-up-v2n-m1.md` in alp-sdk.
- **Accelerator-backend design**: `docs/aen-accelerator-backends-design.md` in alp-sdk.
- **Example**: `examples/v2n/v2n-m1-deepx-inference/` in alp-sdk (V2N-M1 SoM); cross-accelerator demo at `examples/v2n/v2n-m1-ros-perception/`.
- **Hardware**: **HG-V2N-M1-001 §6.7** (PCIe mux design rules).
- **SDK API**: `<alp/inference.h>` (portable inference surface); `<alp/chips/deepx_dxm1.h>` (`deepx_dxm1_bring_up` host sequencer).
- **DeepX tooling**: `dxcom` model compiler + `dx_rt` runtime (vendor-supplied from `github.com/DEEPX-AI`; see `docs/vendor-partnerships.md` §DEEPX).

10 Revision History

Revision	Changes	Date
0.1	Initial draft.	May 2026
0.2	Re-synced to current alp-sdk: example path `examples/v2n/v2n-m1-deepx-inference`; canonical API `<alp/inference.h>` (ALP_INFERENCE_BACKEND_DEEPX_DXM1 / MODEL_DXNN) + host sequencer `<alp/chips/deepx_dxm1.h>` (`deepx_dxm1_bring_up`); corrected PCIe-mux description to the PI3DBS12212A muxes; model artifact `.dxnn` via `dxcom`; refreshed references. Flagged unconfirmed `alp-pcie-select` CLI and `alp.deepx/alp.drp_ai` Python bindings.	June 2026
0.3	Removed module-internal pin designators (the PCIe-mux PD/SEL pads and the `M1_RESET` pad): the E1M-X edge connector is the customer contract and the only interface the lifecycle statement's form-fit-function guarantee covers — customers drive this through `deepx_dxm1_bring_up()`. Corrected the TPS628640 rails: the three DEEPX rails are always populated, so a `NOT_READY` is a board fault; only the `0x4D` rail is optional. Corrected the bring-up order (route the PCIe muxes, then release reset). Removed fabricated tooling, device nodes, Python bindings and a `~120 fps` throughput figure that was never measured; nothing in the DeepX path is silicon-proven today. Fixed the Alp SDK repository links (`alpDevs` → `alplabai`).	July 2026

Table 4 Revision History