ALP LAB

DeepX DX-M1 First Inference Walkthrough

AN-009 · E1M-X V2N-M1 (V2N + DX-M1)

Document Number: AN-009 **Revision:** 0.9 **Date:** November 2026 **Status:** Preliminary

alplab.ai

© 2026 Alp Lab AB. All rights reserved.

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**, section **PCIe (V2N-M1 specifics)**).

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 built for the module's own Yocto `MACHINE` (`e1m-v2m101-a55` / `e1m-v2m102-a55` / `e1m-v2m103-a55`) with DEEPX's `meta-deepx-m1` layer in `bblayers.conf` (the DX-M1 driver requires a Linux PCIe stack), DeepX **DX-COM™** host tool installed.
Outcome	A DX-M1 bring-up you have verified at each stage, and a first inference dispatched through `<alp/inference.h>` on the A55.
Time	60 minutes (first-time DX-COM model-compile is the long pole).
Source	`docs/bring-up-v2n-m1.md` and `docs/soms/v2n-m1.md` in alp-sdk vendor DX-COM documentation from DeepX.

Table 1 Scope summary

> **Note: Verification status (Alp SDK™ v0.17.0).** On `E1M-V2M103` with DX-RT 3.2.0 and DX-M1 firmware 2.4.0, the DX-M1 flash-boots, enumerates as PCIe `1ff4:0000` Gen3 × 2, and `dxrt-cli -s` returns rc `0`; a `.dxnn` model opens and invokes through `<alp/inference.h>` under both `ALP_INFERENCE_BACKEND_DEEPX_DXM1` and `ALP_INFERENCE_BACKEND_AUTO`, with outputs matching an ONNX Runtime CPU reference. Running one model on the DRP-AI3 and one on the DX-M1 at the same time is **not** bench-verified.

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**). The Alp Lab U-Boot routes them to the DX-M1 at boot; the Alp SDK™ has no runtime API to switch them.

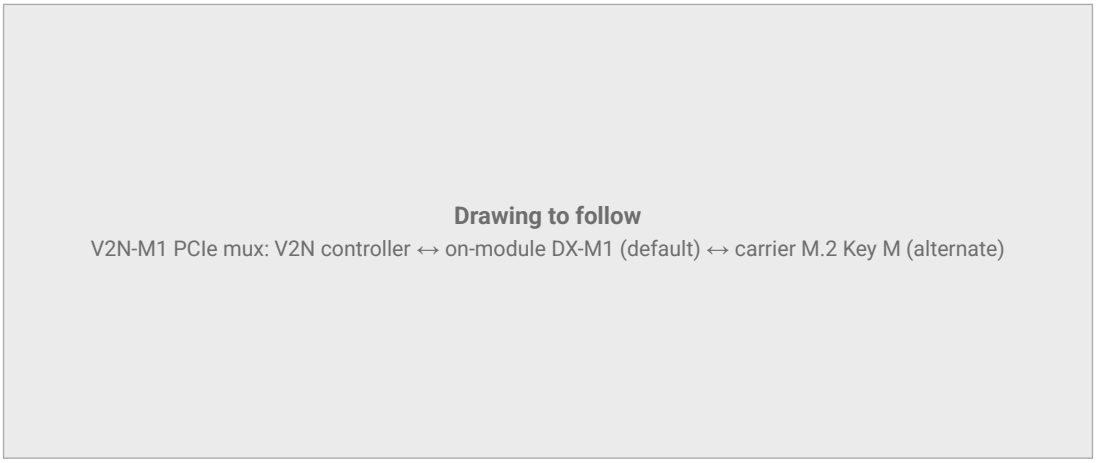


Figure 1 V2N-M1 PCIe mux: V2N controller ↔ on-module DX-M1 (default) ↔ carrier M.2 Key M (alternate)

Copyright © 2026 Alp Lab AB

Warning: Only one PCIe endpoint can be active at a time. While the lanes serve the on-module DX-M1, the carrier M.2 slot is disconnected, 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 rely on the Alp Lab boot firmware to bring the DX-M1 up rather than driving any assumed internal pin assignment. I^2C 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

On V2N-M1 the DX-M1 bring-up runs entirely in **U-Boot's `board_late_init()`** on the Cortex-A55, before Linux or the Cortex-M33 image starts. No application firmware (Cortex-M33 or Linux) performs or re-runs it. This section describes the sequence so that you can recognise each stage in the U-Boot log and verify it on your board.

The order is:

1. Bring up the 0.75 V DEEPX rail on the secondary PMIC's CH2 and confirm it is power-good.
2. Route the PCIe muxes to the DEEPX path, then release `M1_RESET` — only once the rail is power-good.
3. Hand off to Linux.

Warning: Keep this sequence in the Alp Lab-supplied U-Boot. Do not re-implement or re-run it from the Cortex-M33, from Linux, or from a custom boot loader: a second, uncoordinated writer of the PMIC, or a sequence run in the wrong order, can damage the DX-M1. The `BRD_I2C` bus that carries the PMIC is owned by the Cortex-A55; the Cortex-M33 must never master it.

3.1 Step 1 — The 0.75 V DEEPX rail (U-Boot)

The DEEPX DDR / NPU rail is CH2 (phases 3 + 4) of the secondary PMIC, a Renesas DA9292 at I^2C `0x1E`. On the V2N base this channel stays disabled; on V2N-M1 U-Boot programs it to 0.75 V, reads the settings back, enables it and polls for power-good. On success the U-Boot log shows:

```
ALP: DA9292 programmed CTRL_01=0x01 VOUT_CH2=0x96/0x96
ALP: DEEPX rail 0.75V up (PG)
```

If the log instead shows `ALP: DA9292 ...` followed by `abort`, `DEEPX rail not enabled` or `DEEPX rail disabled`, the rail did not sequence and `M1_RESET` stays asserted, so nothing later in this note will work. The printed register bytes name the check that failed; probe the 0.75 V rail directly and contact Alp Lab support.

3.2 Step 2 — Route the muxes, then release `M1_RESET` (U-Boot)

Order matters here. Right after a good rail step, U-Boot routes the PCIe muxes to the DEEPX path and then releases `M1_RESET` (active-low on V2N-M1), no sooner than 16 ms after the rail is enabled, which covers the DX-M1's own reset-supervisor delay. PCIe link training starts after the reset release.

Note: The Alp SDK™ also ships a host sequencer, `deepx_dxm1_bring_up()` in `<alp/chips/deepx_dxm1.h>`, which routes the muxes and then releases `M1_RESET` in one call. It is for platforms where a portable caller owns `M1_RESET`. On V2N-M1 U-Boot is the sole driver of `M1_RESET`, so do **not** call it there.

Copyright © 2026 Alp Lab AB

3.3 Step 3 — 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; you only confirm the parts are present and responding. The `0x48` instance only ACKs once Step 1 has run.

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()` after a good Step 1 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 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. `BRD_I2C` belongs to the Cortex-A55: run any such check from Linux, never from the Cortex-M33.

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. In an image built for the module's `e1m-v2m10x-a55` `MACHINE` with `meta-deepx-m1`, the DEEPX kernel modules (`dx_dma`, `dxrt_driver`) load at boot and the DX-M1 enumerates as PCIe `1ff4:0000`. Check the device with:

```
dxrt-cli -s
```

It should return rc `0` and report the DX-RT, driver and firmware versions; compare the firmware version with the runtime the image pins. From user space, constructing a `dxrt::InferenceEngine` (with `dxrt.service` running) opens `/dev/dxrt0`; from there you load a `.dxnn` model and run inferences.

> **Warning:** **Do not replace the DX-M1 firmware.** The DX-M1 boots an Alp Lab firmware build for this SoM from its on-module SPI NAND, provisioned at the factory. Updating it with stock DEEPX firmware leaves the NPUs dead (`dxrt_polling_ack:timeout`), and recovery needs a boot-strap change on the module, which is not a field operation. The shipped `dxrt-cli` warns before `-u` (firmware update), `-w` (firmware upload) and `-C` (firmware config) but does not block them; use them only with firmware supplied by Alp Lab for this module.

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. A `.dxnn` file is compiled for a specific DX-M1 firmware release; use one that matches the firmware version `dxrt-cli -s` reports.

Copyright © 2026 Alp Lab AB

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.

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 DEEPX publishes on `github.com/DEEPX-AI` for customers supplied with a DEEPX NPU; it is not vendored into the SDK, and the `e1m-v2m10x-a55` machines pull it in through DEEPX's `meta-deepx-m1` Yocto layer.
- On the **Zephyr `m33_sm` core** there is no DEEPX backend and the open fails. The M33 is a system manager, not an inference host. `examples/v2n/v2n-m1-deepx-inference/` in alp-sdk is such an `m33_sm` image: it shows the `<alp/inference.h>` calls, but it runs no DX-M1 inference on silicon.

> **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 interface is the C `<alp/inference.h>` surface (DEEPX-specific options in `<alp/ext/deepx/inference.h>`); 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. The U-Boot log shows `ALP: DEEPX rail 0.75V up (PG)`.
4. All three DEEPX TPS628640 instances ACK at their addresses.
5. `lspci` lists the DEEPX device (`1ff4:0000`).
6. `dxrt-cli -s` returns rc `0`, and constructing a `dxrt::InferenceEngine` succeeds.
7. A reference DeepX inference application (e.g. `run_model` with a `.dxnn` model) 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 one-model-per-accelerator example is `examples/v2n/v2n-two-models/`.

On a V2N-M1 build `ALP_INFERENCE_BACKEND_AUTO` resolves to the DX-M1; a model for the DRP-AI3 must name `ALP_INFERENCE_BACKEND_DRPAI` explicitly.

> **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. Running both at once is not bench-verified — do not budget a frame rate from this section.

Copyright © 2026 Alp Lab AB

8 Troubleshooting

Symptom	Likely cause / fix
U-Boot logs `ALP: DA9292 ...` with `abort`, `DEEPX rail not enabled` or `DEEPX rail disabled`	The 0.75 V rail did not sequence and `M1_RESET` stays asserted. The printed register bytes name the failed check; probe the 0.75 V plane (a short is the usual cause) and contact Alp Lab support.
`tps628640_init()` returns `ALP_ERR_NOT_READY` on `0x44` / `0x48` / `0x4F`	If the U-Boot rail step failed, fix that first (`0x48` only ACKs after it). Otherwise a real fault, not an assembly option — these three rails are always populated on V2N-M1. The part is powered down or address-strapped wrong. Probe the rail directly.
`lspci` shows no DeepX device	Check the U-Boot log for the rail step. On a module whose identity EEPROM does not mark it as V2N-M1, U-Boot releases `M1_RESET` but leaves the PCIe muxes alone. Contact Alp Lab support.
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.
`dxrt_polling_ack: timeout`, NPUs not responding	The DX-M1 firmware was replaced with non-Alp Lab firmware. Not a field fix; contact Alp Lab at contact@alplab.ai.
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::InferenceEngine` construction fails	Check that the DEEPX kernel driver is loaded and `dxrt.service` is running, and that `dxrt-cli -s` returns rc `0`. Then 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` and `docs/soms/v2n-m1.md` in alp-sdk.
- **Accelerator-backend design**: `docs/aen-accelerator-backends-design.md` in alp-sdk.
- **Examples**: `examples/v2n/v2n-m1-deepx-inference/` (`<alp/inference.h>` usage, Cortex-M33 image) and `examples/v2n/v2n-two-models/` (one model per accelerator) in alp-sdk.
- **Hardware**: **HG-V2N-M1-001**, section **PCIe (V2N-M1 specifics)** (PCIe mux design rules).
- **SDK API**: `<alp/inference.h>` (portable inference surface); `<alp/ext/deepx/inference.h>` (DEEPX-specific options).
- **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`	June 2026

Copyright © 2026 Alp Lab AB

Revision	Changes	Date
	▸ refreshed references. • Flagged unconfirmed `alp-pcie-select` CLI and `alp.deepx/alp.drp_ai` Python bindings.	
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
0.4	• Retensed to Alp SDK™ v0.17.0. • The DX-M1 rail bring-up, PCIe-mux routing and `M1_RESET` release run in U-Boot on the Cortex-A55; application firmware does not run them, and `deepx_dxm1_bring_up()` must not be called on V2N-M1. • Reordered the sequence (rail first; the `0x48` TPS628640 only ACKs after it) and added a warning that re-running it elsewhere can bring the 0.75 V DEEPX core rail up at 1.80 V. • Removed the non-existent `da9292_v2n_m1_enable_deepx_rail()` and `dxrt_init()`; verification now uses the U-Boot log, `lspci 1ff4:0000`, `dxrt-cli -s` and `dxrt::InferenceEngine`. • Replaced the "nothing run on silicon" warning with the v0.17.0 bench status on `E1M-V2M103`. • Added a warning against replacing the factory-provisioned DX-M1 firmware. • Corrected the cross-accelerator example (`examples/v2n/v2n-two-models/`), noted that `ALP_INFERENCE_BACKEND_AUTO` resolves to the DX-M1, and noted that `examples/v2n/v2n-m1-deepx-inference/` is a Cortex-M33 image that runs no DX-M1 inference. • Fixed the HG-V2N-M1-001 cross-reference.	October 2026
0.5	• Added a note to the DEEPX-rail warning that `BRD_I2C` belongs to the Cortex-A55 and that PMIC checks must run from Linux, never from the Cortex-M33.	October 2026
0.6	• Release history: every entry formatted as a bullet list.	October 2026
0.7	• Contact: contact@alplab.ai is the only contact address (DX-M1 firmware troubleshooting row).	October 2026
0.8	• Product name on page 1 links to the product page on alplab.ai.	November 2026
0.9	• U-Boot sequence warning: no longer quotes a PMIC default voltage.	November 2026

Table 4 Revision History

Copyright © 2026 Alp Lab AB