ALP LAB

Multi-Processor Mailbox Communication

AN-007 · E1M-X V2N and V2N-M1 (CA55 ↔ CM33)

Document Number: AN-007 **Revision:** 0.4 **Date:** July 2026 **Status:** Preliminary

alplab.ai

1 Scope

Exchange messages between the **Cortex-A55 application cluster** (running Yocto Linux) and the **Cortex-M33 system-manager core** (running Zephyr) on the E1M-X V2N / V2N-M1 SoM. The two cores share a non-cacheable carve-out for the **RPMsg** (Remote Processor Messaging) OpenAMP virtio rings; the V2N hardware-mailbox doorbell provides the notify-on-write side channel.

This pattern is the Alp SDK™'s way to combine deterministic real-time control (motor loops, sensor sampling at 1 ms cadence) with rich application logic (UI, network, AI) on a single SoM.

> **Important:** **Read this before you plan around it.** On V2N today the two halves of this link are **not symmetric**, and this AN does not pretend otherwise. The A55 side is served by a real `<alp/rpc.h>` backend. The shipped M33 slice **deliberately bypasses `<alp/rpc.h>`** and speaks raw OpenAMP, because it was shaped to interoperate with Renesas's licence-gated `rpmsg_sample_client`. What is bench-proven on silicon is the **transport**; what does not exist yet is a portable `<alp/rpc.h>` backend on the M33. **Section 2** states exactly what you can run today.

Audience	Firmware engineers building heterogeneous-compute applications.
Prerequisites	V2N or V2N-M1 SoM, **QS-E1M-X-EVK-001** completed for **both** core targets (A55 and M33). Working knowledge of OpenAMP / RPMsg semantics — on the M33 side you are writing OpenAMP code directly, not a portable SDK call.
Outcome	An attached A55 ↔ M33 OpenAMP link with a proven byte-exact echo round-trip. The A55 drives it through the production `<alp/rpc.h>` API; the M33 runs a raw-OpenAMP echo responder. Round-trip latency is not yet characterised.
Time	45 minutes.
Source	`docs/heterogeneous-builds.md`, `examples/multicore/rpmsg-v2n/`, and `examples/multicore/mproc-mailbox/` in alp-sdk.

Table 1 Scope summary

2 What Works Today

Capability	State	Detail
Raw A55 ↔ M33 UIO / OpenAMP transport	**Bench-proven on silicon**	Proven on E1M-X V2N-M1: the A55 attaches to the already-running M33's published resource table, and attach plus a byte-exact echo round-trip (1, 4, 16, and 64 B) passes end-to-end over the real MHU doorbell and real shared-memory vrings. The concurrent-close use-after-free hardening (GHSA-xhm8-7f87-93q5) is proven on the same bench. Exercised by tests/yocto/rpc_uio_bench_main.c, which links the real dispatcher and drives the public <alp/rpc.h> API — not a test double.
<alp/rpc.h> on the **A55**	**Available**	src/backends/rpc/yocto_uio_drv.c binds the RPC dispatcher to userspace OpenAMP / libmetal over /dev/uio* in ATTACH mode. It is registered for V2N silicon specifically and outranks the generic /dev/rpmsg* backend on this SoC; every other Linux target keeps the chardev backend. So alp_rpc_open() / _call() / _send() / _subscribe() / _close() are the real API on the A55.
<alp/rpc.h> on the **M33**	**Not available**	The shipped M33 slice bypasses it by design and hand-wires OpenAMP against the board devicetree. A V2N-specific <alp/rpc.h> M33 backend is explicitly deferred follow-up work. **Budget for writing raw OpenAMP on the M33**, or keep the M33 slice's protocol surface small enough to migrate later.
The rpmsg-v2n example, end to end	**Not runnable as shipped**	Its M33 slice builds and is the basis of the silicon proof, but its linux/ half subscribes to a temperature method the M33 echo responder **never publishes** — run the pair unmodified and the Linux side simply waits, then exits having received nothing. Treat the example as two references, not a working demo. See **Section 4**.

Table 2 V2N A55 ↔ M33 link: state as of SDK v0.10.1

3 Architecture

TODO image
Inter-core mailbox architecture: A55 cluster ↔ shared non-cacheable RPMsg vring carve-out ↔ M33 real-time core, with the V2N hardware-mailbox doorbell providing the notify path

Figure 1 Inter-core mailbox architecture: A55 cluster ↔ shared non-cacheable RPMsg vring carve-out ↔ M33 real-time core, with the V2N hardware-mailbox doorbell providing the notify path

<alp/rpc.h> defines no pre-baked services: method names are **application-defined**. On the wire every RPMsg frame carries a single-line ASCII method header (<method>\0, up to 32 bytes including the NUL) followed by an opaque application-defined payload that the SDK passes through verbatim.

That framing is what makes the current asymmetric link usable at all. The M33 echo responder has **no method routing**: it reflects whatever bytes arrive, verbatim. Because an `<alp/rpc.h>` request frame is `<method>\0<payload>`, echoing the whole frame back returns a well-formed response **on the same method** — so an A55 `alp_rpc_call()` completes normally. It works, but understand what it is:

> **Warning: The M33 is not implementing your methods — it is reflecting them.** Any method name you call will "succeed" and return your own payload. That is a transport proof, not application logic. The moment your M33 slice must compute a distinct response, you write that responder against raw OpenAMP yourself; there is no `alp_rpc_subscribe()` on the M33 to hang it off.

On V2N the A55 slice's channel contract comes from the build-time-generated `<alp/system_ipc.h>` (endpoint IDs, mailbox channel, carve-out address), emitted by the orchestrator from the project's `ipc:` block. The M33 slice does **not** consume that header — it takes its addresses from the board devicetree and fixes its endpoint at the vendor sample's address. The generated header is therefore not yet the shared contract across both cores on this link; it is the A55's half of one.

4 Software Walkthrough

4.1 M33 side (real-time)

There is no `<alp/rpc.h>` code to show here, because the shipped M33 slice does not use it. `examples/multicore/rpmsg-v2n/m33_sm/src/main.c` is a from-scratch OpenAMP firmware written against the real RZ/V2N devicetree and the Renesas MHU mailbox driver. In outline it:

1. Publishes a **liveness beacon** into the resource-table region as its very first action, so a bench `devmem` read from Linux proves in one shot that the M33 is alive, the shared window is backed, and the M33 ↔ A55 address translation is correct.
2. Runs `metal_init()`, registers the shared-memory device with two regions (the RPMsg buffer pool and the resource table), and copies its resource table into the shared region for the A55's attach to parse.
3. Creates a virtio device in **device** (slave) role and blocks in `rproc_virtio_wait_remote_ready()` — the M33 is started by the boot chain before Linux, so it waits for the A55 to catch up. This is an **attach**-mode link: nothing loads an ELF into the M33 at runtime.
4. Creates one endpoint, `"rpmsg-service-0"`, and echoes every frame it receives back verbatim.

The doorbell is **asymmetric**, and this is the part most likely to surprise you: the forward A55 → M33 kick and the reverse M33 → A55 kick go through **different** mailbox units, because the reply path the mailbox driver would naturally use raises an interrupt that reaches no A55 interrupt line at all. The reverse kick therefore pokes a specific software-interrupt unit directly. If you write your own M33 responder, budget time for this: it is bench-measured behaviour, not something the vendor documentation hands you.

> **Note:** Two details in that firmware are load-bearing and worth carrying into your own responder. The receive endpoint is created **before** the receive loop starts, not from the responder thread — otherwise the A55's very first request lands with no matching endpoint and is dropped, and the first `alp_rpc_call()` times out. And received frames are queued rather than held in a single buffer, so two rapid frames cannot overwrite each other before the responder runs.

Build both slices from the one `board.yaml` with the orchestrator:

```
cd alp-workspace/alp-sdk/examples/multicore/rpmsg-v2n
# tan build finds board.yaml by walking up from the cwd (it takes no
# explicit app path), then fans out the M33-SM Zephyr image
# (BOARD=alp_e1m_v2n101_m33_sm) and the A55 Yocto slice from that one file.
tan build

# Bundle + flash, walking the preset's boot_order:
tan image
tan flash
```

4.2 A55 side (Yocto Linux)

This half **is** the portable surface, and this is the call the silicon proof exercises:

```c
#include <alp/rpc.h>
#include <alp/system_ipc.h>   // generated by alp_orchestrate.py
#include <stdio.h>
#include <string.h>

int main(void)
{
    printf("[a55] echo client coming up\n");

    // The A55's ids come from the generated header. Note the M33 slice
    // does NOT consume this header -- it fixes its endpoint at the
    // vendor sample's address, so the two must be kept consistent by
    // hand until an <alp/rpc.h> M33 backend lands.
    alp_rpc_channel_t *ch = alp_rpc_open(&(alp_rpc_config_t){
        .name    = ALP_IPC_ALP_DEFAULT_RPMSG_NAME,
        .src_ept = ALP_IPC_ALP_DEFAULT_RPMSG_DST_EPT,
        .dst_ept = ALP_IPC_ALP_DEFAULT_RPMSG_SRC_EPT,
        .mbox_ch = ALP_IPC_ALP_DEFAULT_RPMSG_MBOX_CH,
    });
    if (ch == NULL) {
        printf("[a55] alp_rpc_open failed: %d\n", (int)alp_last_error());
        return 1;
    }

    // The M33 echoes the frame verbatim, so the response payload comes
    // back byte-identical to the request. This proves the transport; it
    // is NOT the M33 computing a reply.
    const char req[]    = "hello";
    char       resp[64] = { 0 };
    size_t     resp_len = sizeof resp;

    alp_status_t rv = alp_rpc_call(ch, "echo_test",
                                   req, sizeof req,          // request
                                   resp, &resp_len,          // response
                                   /*timeout_ms*/ 3000);
    printf("[a55] echo rv=%d len=%zu match=%d\n",
           (int)rv, resp_len, memcmp(req, resp, sizeof req) == 0);

    alp_rpc_close(ch);
    printf("[rpmsg-v2n] done\n");
    return 0;
}
```

On V2N the Linux backend reaches the M33 over libmetal + OpenAMP user-space access to `/dev/uio*` in **attach** mode — not `/dev/rpmsg*`. V2N's Yocto BSP does not carry the mainline `rpmsg_char` remoteproc glue, only the raw generic-UIO regions a userspace OpenAMP master can attach to directly. The SDK selects this backend automatically on V2N silicon; every other Linux target keeps the `/dev/rpmsg*` backend unchanged. Your application code is the same `<alp/rpc.h>` calls either way.

5 Expected Output

A55 console — what the bench proves is a byte-exact round-trip (validated at 1, 4, 16, and 64 B):

```
[a55] echo rv=0 len=6 match=1
[rpmsg-v2n] done
```

M33 console:

```
rpmsg_v2n_m33_sm: rpmsg-v2n/m33_sm: starting (alp-sdk #683, Path B Phase 1)
rpmsg_v2n_m33_sm: rpmsg-v2n/m33_sm: bringing up the rpmsg virtio device
rpmsg_v2n_m33_sm: rpmsg-v2n/m33_sm: Linux responder started
```

> **Note:** Before debugging the link itself, confirm the M33 is alive: read the **liveness beacon** from Linux with `devmem` against the resource-table region's A55 alias. A magic word plus a version proves the M33 booted, the shared window is backed, and the address translation is correct; a **changing** heartbeat word on a re-read distinguishes "still running" from "wrote once, then faulted".

6 Performance

Round-trip latency on this link is **not yet characterised**, and this note will not print a figure it cannot stand behind. The silicon work to date proves **correctness** — attach, byte-exact echo at 1/4/16/64 B, and safe concurrent close — not timing. Budget your own measurement against your own payload sizes and scheduler configuration.

Two structural points hold regardless of the measured number. The mailbox only ever carries a **wakeup**, never data: the payload is already in shared memory before the doorbell rings, so per-message cost is dominated by the vring and interrupt path rather than by payload size. And where the M33 samples at >= 10 kHz but only periodically reports, prefer fire-and-forget sends of **batched** payloads over per-sample request/response — a blocking `alp_rpc_call()` per sample pays the full round trip every time.

7 Troubleshooting

Symptom	Likely cause / fix
A55 `alp_rpc_open()` returns NULL	The M33 never published its resource table, or the A55 cannot map the UIO regions. Confirm the M33 is alive first via the liveness beacon (above), then check the process can open `/dev/uio*` — run as root or grant read/write on those devices. Read `alp_last_error()` for the specific failure.
`alp_rpc_open()` succeeds but every call times out	The M33 is blocked in its wait-for-remote-ready and never saw the attach, or the reverse doorbell is not reaching the A55. The reverse M33 → A55 kick uses a **different** mailbox unit from the forward path (see **Section 3**); a responder that rings the "natural" reply channel raises an interrupt no A55 line receives, so replies are silently never delivered.
The first call after a cold attach times out, later ones work	The responder endpoint is being created **after** the receive loop starts, so the A55's first frame arrives with no matching endpoint and is dropped. Create the endpoint before the loop.
Two rapid requests, only the second answered	A single receive buffer is being overwritten before the responder thread runs. Queue received frames instead.
The `rpmsg-v2n` example's Linux half prints nothing and exits	**Expected — the example is incoherent as shipped.** Its `linux/` half subscribes to `temperature`, which the M33 echo responder never publishes. Drive an echo method with `alp_rpc_call()` instead (see **Section 3**).
Channel comes up but no frames arrive	DMA-capable carve-out not declared non-cacheable. Confirm the orchestrator placed the carve-out in the preset's non-cacheable region so the backend keeps the virtio rings coherent.

Table 3 Common failures

8 References

- **Canonical walkthrough**: `docs/heterogeneous-builds.md` in alp-sdk (end-to-end dual-OS A55 ↔ M33 build + RPMsg flow).
- **Examples**: `examples/multicore/rpmsg-v2n/` (V2N A55 ↔ M33) and `examples/multicore/mproc-mailbox/` (single-SoC AEN M55-HP ↔ M55-HE) in alp-sdk. **Caveat on `rpmsg-v2n`**: its README still banners `[UNTESTED] -- v0.6 paper-correct`, which is stale — the M33 slice it describes has since been rewritten and bench-proven on V2N-M1 silicon. Its `m33_sm/testcase.yaml` is `build_only: true` (a `native_sim` build was dropped because the firmware needs real RZ/

V2N devicetree plus the vendor MHU / FSP / OpenAMP modules), so CI proves it **compiles**, not that it runs. And its `linux/` half targets a method the M33 does not publish. Read the two slices as references; do not expect the pair to run.

- **Silicon proof**: `tests/yocto/rpc_uio_bench_main.c` — the A55 attach + echo bench binary. It links the real dispatcher and drives the public `<alp/rpc.h>` API against the live M33, so it is the closest thing to a working end-to-end demo of this link.
- **SDK API**: `<alp/rpc.h>` (framed RPMsg) plus the generated `<alp/system_ipc.h>`; lower-level mailbox / shmem / hwsem primitives in `<alp/mproc.h>`. V2N A55 backend: `src/backends/rpc/yocto_uio_drv.c`.
- **Companion tutorial**: `docs/tutorials/15-mproc-mailbox.md` in alp-sdk – the `<alp/mproc.h>` mailbox primitives on AEN.
- **V2N HW Design Guide** §6.7 (RZ/V2N internal interconnect / DMA / mailbox).

9 Revision History

Revision	Changes	Date
0.1	Initial draft.	May 2026
0.2	Re-based on the current alp-sdk: high-level IPC moved to `<alp/rpc.h>` (`alp_rpc_open` / `_call` / `_send` / `_subscribe`) with `<alp/mproc.h>` as the mailbox/shmem/hwsem primitive layer; corrected example paths to `examples/multicore/rpmsg-v2n/` and `.../mproc-mailbox/`; board target `alp_e1m_v2n101_m33_sm`; `west alp-build` orchestrator flow; replaced the fictional `alp.mproc.*` endpoints / CBOR framing and the Python `/dev/alp_mproc0` path with the real method-framed RPMsg surface over `/dev/rpmsg*`.	June 2026
0.3	Corrected the portable-RPC story: the shipped `rpmsg-v2n` M33 slice deliberately bypasses `<alp/rpc.h>` and speaks raw OpenAMP (its peer is a licence-gated Renesas sample), so the two slices are not a matched pair. The A55 side of `<alp/rpc.h>` is real and bench-proven over UIO; the gap is M33-only. Replaced a fabricated M33 walkthrough that called `alp_rpc_open`/`alp_rpc_subscribe` on a slice that does not implement them, removed fabricated console output and latency figures, and explained why the echo appears to answer any method. Corrected `/dev/rpmsg*` to `/dev/uio*` on V2N. Fixed the Alp SDK repository links (`alpDevs` → `alplabai`).	July 2026
0.4	Retired the `west alp-*` build commands: alp-sdk v0.12.0 removed the SDK build executor (ADR-0020 Phase 4), so `west alp-build` → `tan build`, `west alp-image` → `tan image`, `west alp-flash` → `tan flash`. `tan build` finds `board.yaml` by walking up from the cwd and takes no explicit app path, so the sequence now `cd`s into the example first; it has no per-core `--core` flag, so the separate M33-slice iterate command was dropped.	July 2026

Table 4 Revision History