PDF 4.2 – Bridge Sanity Check & First Message

1. Purpose

This document defines the **first, minimal interaction across the MCU ↔ Linux bridge**.

The goal is not to transfer CAN data, but to **prove that inter-processor communication works at all**, in a controlled and observable way.

This is the first deliberate “baby step” of Phase 4.

---

2. Why a Sanity Check Is Required

Before moving real data across the bridge, several assumptions must be validated:

- The bridge infrastructure is operational
- Messages sent from the MCU arrive at the Linux processor
- Message boundaries are preserved
- Communication does not block or destabilize the MCU

Attempting to send CAN frames immediately would make failures ambiguous and difficult to diagnose.

---

3. Scope of This Step

In scope:

- One-way communication: **MCU → Linux**
- A fixed, known message
- Periodic transmission at a low rate
- Observable confirmation on the Linux side

Out of scope:

- CAN data
- Buffering
- Serialization complexity
- Error recovery
- Performance optimization

---

4. Message Definition (Minimal Contract)

The first message across the bridge is intentionally trivial.

Characteristics:

- Fixed message type
- Fixed payload
- Human-readable

Example content (conceptual):

- A static string (e.g. “bridge_alive”)
- Or a simple counter value

The exact content is less important than **reliable delivery**.

---

5. Transmission Strategy

5.1 Rate

Messages are transmitted at a **slow, fixed rate** (e.g. once per second).

This ensures:

- minimal load on the MCU
- clear visibility on the Linux side

5.2 MCU Behavior

- Message transmission must be non-blocking
- Failure to transmit must not affect CAN reception

---

6. Linux-Side Expectations

On the Linux processor, a simple application is responsible for:

- attaching to the bridge endpoint
- receiving messages
- printing or logging received content

No parsing or interpretation is required at this stage.

---

7. Validation Criteria

This step is considered successful if:

- messages appear on the Linux side at the expected rate
- message content matches what was sent
- MCU remains stable and responsive
- no degradation in CAN reception (Phase 3 behavior preserved)

---

8. Failure Modes to Observe

Potential failures at this stage include:

- no messages received
- intermittent reception
- message duplication
- MCU stalls or resets

Each failure mode provides clear diagnostic information before proceeding further.

---

9. Documentation Rationale

This step is intentionally documented as a standalone PDF to:

- demonstrate disciplined, incremental integration
- make assumptions explicit
- show how risk is reduced step-by-step

---

10. Implementation Reference

The reference implementation for this step will be provided in:

- `MCU-Phase4.2.ino` (MCU-side)
- Linux-side Phase 4 application (to be introduced in Phase 5)

Source code is intentionally excluded from this document.

---

11. What This Step Does Not Prove

This step does **not** yet demonstrate:

- throughput capability
- data integrity under load
- buffering across the bridge
- CAN data transfer

Those concerns are addressed incrementally in later Phase 4 documents.

---

12. Next Document

Proceed to **PDF 4.3 – Message Formats & Contracts**.