PDF 3.X – Driver Library Change (Root Cause & Resolution)

1. Purpose

This document records the **driver library change** made during Phase 3 of the project.

Its purpose is to: - document the observed problem - describe the investigation and root cause assessment - justify the final technical decision

This document is intentionally separate from the architectural PDFs (3.1–3.4) to preserve architectural stability while capturing real-world integration learning.

---

2. Context

During Phase 3, the MCU software was integrated with the MCP2515 CAN controller on the Arduino UNO Q platform.

An initial driver library was selected based on: - common usage in Arduino CAN tutorials - familiarity within the embedded community

At this stage, no assumptions were made about platform-specific behavior.

---

3. Initial Driver Selection

The first MCP2515 driver library used was a widely adopted Arduino MCP2515 library.

The selection was reasonable because: - it is well-documented - it is commonly used on AVR- and ESP-based platforms - it provides a simple, high-level API

No architectural dependency on this specific library was introduced in Phase 3.1.

---

4. Observed Failure Mode

During integration on the Arduino UNO Q platform, the following behavior was observed:

- SPI communication with the MCP2515 could be verified directly
- Register-level reads returned valid values
- CAN controller reset and low-level access succeeded

However:

- The library's CAN initialization routine consistently **blocked**
- No error code or timeout was returned
- The system never progressed to normal operation

This behavior prevented CAN reception entirely.

---

5. Investigation and Isolation

A structured debugging approach was applied:

1. **Hardware validation**
2. Wiring and power were verified
3. CAN transceiver and bus were confirmed operational
4. **SPI verification**
5. Direct SPI register reads returned expected values
6. MCP2515 status registers behaved correctly
7. **Software isolation**
8. The failure was isolated to the library initialization path
9. The block occurred consistently at the CAN initialization call

This eliminated hardware and SPI transport as root causes.

---

6. Root Cause Assessment

The failure was attributed to **library behavior on the UNO Q toolchain**.

Likely contributing factors include: - blocking wait loops without timeouts - assumptions about timing or interrupt behavior - limited testing on newer MCU platforms

Such issues are common when combining: - newer hardware platforms - legacy or platform-specific libraries

---

7. Resolution Strategy

Rather than modifying the failing library, the following strategy was adopted:

- treat the driver as a replaceable component
- select an alternative MCP2515 driver with clearer behavior
- preserve the existing MCU architecture

This approach minimized risk and avoided introducing forked or patched libraries.

---

8. Final Driver Selection

The driver was replaced with the **Autowp MCP2515 library**.

This library demonstrated: - reliable initialization on Arduino UNO Q - non-blocking receive semantics - clear separation between "no message" and error conditions

No changes were required to: - hardware wiring - CAN bus configuration - higher-level MCU architecture

---

9. Validation After Change

Following the library change:

- CAN initialization completed successfully
- Frames were received consistently
- Timing behavior matched transmit periodicity
- Buffer counters confirmed loss-free operation under nominal load

This validated both the driver choice and the architectural abstraction.

---

10. Lessons Learned

Key lessons from this change:

- Driver libraries are **system-level dependencies**, not trivial details
- Architectural abstraction reduces integration risk
- Blocking behavior without timeouts is unacceptable in real-time paths
- Documenting failures provides valuable learning for others

---

11. Impact on Project Documentation

This driver change: - does **not** invalidate PDFs 3.1–3.4 - is fully contained within the driver layer - reinforces the correctness of the original architectural decisions

---

12. Conclusion

The driver library change represents a controlled engineering decision based on evidence, not preference.

By preserving architectural boundaries and documenting the rationale, the project reflects realistic embedded system development rather than idealized examples.

---

13. Next Phase

Proceed to **Phase 4 – Inter-Processor Communication**.