

PDF 3.2 – MCP2515 Driver Overview

1. Purpose

This document describes the **CAN driver layer** used on the MCU to interface with the MCP2515 CAN controller.

It explains: - the role of the driver within the MCU architecture - the expectations placed on the driver - how the driver is integrated and validated

This document deliberately sits **between architecture (PDF 3.1) and implementation details (PDF 3.3)**.

2. Driver Role in the MCU Architecture

The MCP2515 driver forms the **lowest software layer** in the MCU stack.

Its role is to: - translate SPI transactions into CAN controller operations - expose a simple interface for CAN configuration and frame reception - surface error conditions in a detectable way

All higher-level MCU logic (buffering, timestamping, diagnostics) depends on the *behavior* of the driver, not on its internal implementation.

3. Design Requirements for the Driver

The driver is required to meet the following architectural constraints:

D3.2-1 Deterministic behavior

- No unbounded blocking calls during normal operation
- CAN reception must return promptly if no frame is available

D3.2-2 Clear failure visibility

- Initialization success or failure must be observable
- Receive errors must be detectable via return codes or counters

D3.2-3 Hardware abstraction

- Shield the rest of the application from SPI register-level access
- Allow driver replacement without architectural change

D3.2-4 Platform compatibility

- Operate correctly on the Arduino UNO Q platform
 - Function correctly with RouterBridge / Monitor-based diagnostics
-

4. MCP2515 Functional Responsibilities

The driver is responsible for managing the MCP2515 controller lifecycle:

- Resetting the controller to a known state
- Configuring CAN bitrate and oscillator settings
- Placing the controller into normal operating mode
- Reading received CAN frames from internal buffers
- Reporting receive status and errors

The driver does *not* interpret CAN payloads or signals.

5. Driver Integration Model

Within the MCU firmware:

- The driver is initialized once during system startup
- Configuration occurs before entering the main receive loop
- Frame reception is invoked repeatedly from the superloop

The MCU treats the driver as **stateless between calls**, aside from the controller's internal buffers.

6. Polling-Based Reception (Phase 3 Choice)

During Phase 3, the driver is used in **polling mode**:

- The MCU periodically checks for new frames
- If no frame is present, the call returns immediately

Polling was selected for Phase 3 because it: - simplifies early bring-up - improves observability during debugging - avoids interrupt-related complexity on a new platform

Interrupt-driven reception is intentionally deferred to later phases.

7. Error Handling Expectations

The driver is expected to distinguish between:

- “no message available” conditions
- genuine receive or controller errors

The MCU records these conditions using counters rather than attempting recovery actions at this stage.

8. Driver Selection Outcome

The final driver selected for Phase 3 is the **Autowp MCP2515 library**.

The selection is based on: - stable initialization behavior on UNO Q - non-blocking receive semantics - clear error reporting

The rationale and comparative analysis leading to this selection are documented separately in **PDF 3.X – Driver Library Change**.

9. Validation Strategy

Driver correctness is validated indirectly via:

- successful initialization and entry into normal mode
- consistent reception of expected CAN frames
- absence of receive errors under nominal load
- consistency between received frame counts and buffer counts

No driver-specific unit tests are introduced at this stage.

10. Implementation Reference

The driver integration is implemented in:

MCU.ino

This document intentionally avoids embedding source code.

11. Assumptions and Limitations

- MCP2515 oscillator frequency must match configuration

- CAN bus bitrate must be configured consistently across the system
 - Driver behavior is assumed correct within tested operating ranges
-

12. Next Document

Proceed to **PDF 3.3 – CAN Receive Loop & Buffering.**