PDF 3.1 – MCU Responsibilities & Task Design

1. Purpose

This document defines the **architectural responsibilities** of the microcontroller (MCU) within the predictive maintenance CAN system.

The intent of this document is to remain **implementation-agnostic**. It describes *what* the MCU must do and *what it must not do*, independent of specific libraries, polling vs interrupt choices, or platform-specific tooling.

---

2. Scope

This document applies to **Phase 3 – MCU Software Architecture** only.

In scope:

- CAN controller interaction
- Deterministic CAN frame reception
- Timestamping and buffering of raw frames
- Instrumentation to validate correct behavior

Out of scope:

- DBC decoding
- Persistent storage (MF4)
- Predictive maintenance logic
- Alerting or notifications
- Linux-side processing

---

3. MCU Role in the Overall System

The MCU acts as the **real-time edge acquisition component**.

Its role is to: - Receive CAN frames reliably at the electrical interface - Preserve ordering and timing information - Provide loss visibility rather than failing silently - Prepare data for later transport to the Linux processor

The MCU is not a data interpretation or analytics node.

---

4. Core Responsibilities

R3.1-1 CAN Controller Interface

- Configure the CAN controller to the correct bus bitrate
- Monitor controller status and error conditions

R3.1-2 Frame Reception

- Read CAN ID, DLC, and payload
- Support both standard and extended identifiers

R3.1-3 Timestamping

- Assign a reception timestamp at frame arrival
- Preserve relative timing between frames

R3.1-4 Buffering and Decoupling

- Push each received frame into a fixed-size ring buffer
- Prevent CAN reception from being blocked by printing or slow I/O

R3.1-5 Observability

- Maintain counters for:
- total frames received
- frames successfully buffered
- buffer overflow events
- driver-level receive errors

---

5. Explicitly Excluded Responsibilities

To preserve deterministic behavior, the MCU shall not: - Decode signals or apply DBC scaling - Perform file or network I/O - Execute long-running algorithms - Depend on Linux availability for correct reception

---

6. Task Design Philosophy

The MCU firmware follows these principles: - **Determinism first** – bounded execution time in the receive path - **Separation of concerns** – acquisition is isolated from reporting - **Non-blocking behavior** – no waits in the critical path - **Fixed memory usage** – no dynamic allocation in reception logic

---

7. Conceptual Task Breakdown

Although implemented as a single superloop in Phase 3, the design is structured conceptually as multiple tasks.

7.1 CAN Receive Task (Highest Priority)

- Acquire frames from the CAN controller
- Timestamp and enqueue each frame

7.2 Buffer Management Task

- Maintain ring buffer head/tail
- Detect and record overflow conditions

7.3 Diagnostics / Reporting Task (Lowest Priority)

- Periodically report system health metrics
- Optionally output a limited number of frames

This separation prevents diagnostics from affecting reception integrity.

---

8. Timing and Throughput Considerations

Timing behavior is influenced by: - CAN bus load - Controller read latency - Buffer size and consumer rate - Diagnostic output frequency

Phase 3 introduces instrumentation to make these effects observable.

---

9. Validation Hooks

The MCU design includes explicit validation hooks: - `rx_count` - `push_ok` - `buf_overflow` - `rx_error`

Expected behavior at light to moderate load: - `rx_count == push_ok` - `buf_overflow == 0` - `rx_error == 0`

---

10. Implementation Reference

The Phase 3 MCU implementation is provided in:

`MCU.ino`

Source code is intentionally not embedded in this document.

---

11. What This Document Does Not Decide

This document does not prescribe: - Polling vs interrupt-driven reception - Specific driver library choice - Inter-processor communication mechanisms

These are addressed in subsequent Phase 3 documents.

---

12. Next Document

Proceed to **PDF 3.2 – MCP2515 Driver Overview**.