PDF 3.4 – Timing & Data Integrity

1. Purpose

This document defines how **timing behavior** and **data integrity** are handled and validated on the MCU during CAN frame acquisition.

It explains: - how timing information is captured - what integrity means at the MCU boundary - how correctness is verified without signal decoding

This document builds on: - **PDF 3.1 – MCU Responsibilities & Task Design** - **PDF 3.2 – MCP2515 Driver Overview** - **PDF 3.3 – CAN Receive Loop & Buffering**

---

2. Timing Model Overview

2.1 Source of Time

Each CAN frame is timestamped using the MCU's millisecond system clock at the moment the frame is accepted by the receive loop.

This timestamp represents: - reception time at the MCU boundary - relative ordering between frames

Absolute wall-clock accuracy is not a requirement at this stage.

---

3. Timing Objectives

The MCU timing model is designed to:

- Preserve relative inter-frame timing
- Detect jitter and irregular transmit behavior
- Enable validation of expected periodicity
- Support later correlation with Linux-side timestamps

The MCU does not attempt clock synchronization in Phase 3.

---

4. Timestamping Strategy

4.1 When Timestamping Occurs

Timestamping is performed:

- immediately after a frame is read from the CAN controller
- before any buffering or diagnostic output

This minimizes distortion caused by software latency.

4.2 Timestamp Resolution

Millisecond resolution is sufficient for:

- low to moderate CAN frame rates
- educational demonstration purposes
- detecting dropped or delayed frames

Higher-resolution timing is deferred to later phases if required.

---

5. Data Integrity Definition (MCU Scope)

Within the MCU, data integrity means:

- frame ID is captured correctly
- DLC matches the number of valid payload bytes
- payload bytes are copied without modification
- frame ordering is preserved

Signal-level correctness is explicitly out of scope at this stage.

---

6. Integrity Preservation Mechanisms

6.1 Immediate Copying

Payload data is copied from the driver buffer into MCU-owned memory immediately upon reception.

This prevents corruption due to: - driver buffer reuse - asynchronous access

6.2 Fixed-Size Structures

Frames are stored in fixed-size structures with no dynamic allocation, reducing the risk of memory-related faults.

7. Integrity Validation via Counters

The MCU uses counters to validate integrity indirectly:

- `rx_count` – frames successfully received
- `push_ok` – frames successfully buffered
- `buf_overflow` – frames dropped due to buffer saturation
- `rx_error` – driver-level receive errors

Expected invariant under nominal load:

```
rx_count == push_ok
buf_overflow == 0
rx_error == 0
```

Any deviation is treated as a diagnostic condition.

---

8. Detection of Abnormal Conditions

The following conditions are detectable in Phase 3:

- Increased inter-frame spacing (timing jitter)
- Unexpected frame loss (buffer overflow)
- Driver-level faults (receive errors)

The MCU records these events but does not attempt recovery.

---

9. Diagnostic Output and Measurement Effects

9.1 Observer Effect

Diagnostic output itself can perturb timing if not controlled.

Mitigations applied: - rate-limited output - single-line message construction - separation of reception and reporting paths

9.2 Platform Constraints

On the Arduino UNO Q platform, diagnostic output is routed through RouterBridge infrastructure.

This affects presentation but not internal timing correctness.

10. Limitations

The Phase 3 timing and integrity model does not address:

- hard real-time deadlines
- sub-millisecond jitter analysis
- clock drift between MCU and Linux
- error recovery or retransmission

These topics are deferred to later phases.

---

11. Implementation Reference

Timing and integrity mechanisms are implemented in:

`MCU.ino`

Source code is intentionally excluded from this document.

---

12. Next Document

Proceed to **PDF 3.X – Driver Library Change: Root Cause & Resolution**.