Hardware Architecture

1. Purpose of This Document

This document defines the **hardware architecture** for the CAN-based embedded monitoring system. Building on the high-level system architecture, it describes how system responsibilities are allocated to physical hardware components.

The objective is to explain *what hardware elements are used* and *why*, without yet introducing detailed pin assignments, schematics, or electrical design calculations.

---

2. Hardware Architecture Principles

The hardware design follows these principles:

- **Clear Responsibility Allocation**: Each hardware component has a well-defined role.
- **Real-Time Isolation**: Time-critical CAN handling is isolated from non-real-time processing.
- **Modularity**: External interfaces are implemented using modular components.
- **Extensibility**: The architecture can be adapted for additional signals or processing needs.

These principles directly support the system and non-functional requirements defined earlier.

---

3. Major Hardware Components

3.1 Embedded Development Board

The central hardware platform is a **dual-processor development board**, consisting of:

- A microcontroller for real-time tasks
- A Linux-capable processor for higher-level processing

Responsibilities: - Hosting real-time CAN reception firmware - Running Linux-based applications for decoding, storage, and analysis - Providing communication between processing domains

This separation mirrors architectures used in production embedded systems.

3.2 External CAN Controller Module

An external CAN controller module is used to interface with the physical CAN bus.

Responsibilities: - Handling CAN protocol timing - Managing CAN message buffering - Interfacing with the microcontroller via SPI

Using an external controller simplifies firmware design and keeps CAN timing isolated from application logic.

3.3 Physical CAN Interface

The physical CAN interface includes:

- CAN transceiver
- Bus connectors
- Termination (external or internal)

This interface ensures electrical compatibility with standard CAN networks while keeping the embedded system passive.

3.4 Inter-Processor Communication Interface

A communication interface connects the microcontroller and Linux-based processor.

Responsibilities: - Transferring received CAN data - Maintaining separation between real-time and non-real-time domains

The specific protocol is intentionally not defined at this stage.

---

4. Hardware Responsibility Allocation

Hardware Component	Primary Responsibility
CAN Controller Module	CAN frame reception
Microcontroller	Real-time handling, timestamping
Linux Processor	Decoding, storage, analytics
CAN Transceiver	Physical bus interface

This allocation supports deterministic CAN reception while enabling flexible data processing.

---

5. Relationship to System Architecture

This hardware architecture implements the layered system architecture defined in PDF 0.4:

- CAN Interface Layer → CAN controller and transceiver
- Real-Time Processing Layer → Microcontroller
- Non-Real-Time Processing Layer → Linux processor
- User & Data Interaction Layer → Linux environment and peripherals

---

6. What This Document Does Not Define

This document intentionally does **not** define:

- Electrical schematics
- Power supply design
- Pin assignments
- SPI timing details
- CAN bit rates

These details will be introduced incrementally in later design stages.

---

7. Next Steps

With the hardware architecture defined, the next logical step is to describe the **software architecture**, including firmware responsibilities, Linux application structure, and data flow between components.