High-Level System Architecture

1. Purpose of This Document

This document defines the **high-level system architecture** for the CAN-based embedded monitoring system. Building on the previously defined requirements and system context, it describes the major functional blocks of the system and their responsibilities, without yet introducing low-level implementation details.

The goal is to answer: - How the system is structured - How responsibilities are separated - How data flows through the system

This architecture provides the foundation for later hardware, software, and interface design decisions.

---

2. Architectural Principles

The system architecture is guided by the following principles:

- **Separation of Concerns**: Real-time CAN handling is separated from non-real-time processing.
- **Modularity**: Major functions are grouped into clearly defined blocks.
- **Scalability**: The system can be extended to support additional signals or algorithms.
- **Traceability**: Architectural elements can be traced back to system requirements.

These principles align with both embedded best practices and beginner-friendly design.

---

3. Architectural Overview

At a high level, the system is composed of four primary layers:

1. CAN Interface Layer
2. Real-Time Processing Layer
3. Non-Real-Time Processing Layer
4. User & Data Interaction Layer

Each layer has a clearly defined responsibility and interacts with adjacent layers through well-defined interfaces.

---

4. Major Architectural Blocks

4.1 CAN Interface Layer

Responsibilities: - Physical connection to the CAN bus - Reception of raw CAN frames - Basic frame validation

This layer is responsible only for reliable data acquisition and does not perform signal decoding or analysis.

4.2 Real-Time Processing Layer (Microcontroller)

Responsibilities: - Managing CAN controller communication - Timestamping received CAN frames - Buffering and forwarding CAN data

This layer operates in a real-time execution context to minimize frame loss and jitter.

4.3 Non-Real-Time Processing Layer (Linux-Based Processor)

Responsibilities: - Receiving CAN data from the microcontroller - Decoding CAN frames into signals using a DBC file - Running monitoring and analysis algorithms - Managing data storage

This layer handles compute-intensive and flexible tasks that are not time-critical.

4.4 User & Data Interaction Layer

Responsibilities: - Configuration of monitoring parameters - Storage of measurement data - Notification and alert generation

This layer provides visibility and interaction without interfering with real-time behavior.

---

5. High-Level Data Flow

1. CAN frames are transmitted on the physical CAN bus.
2. The CAN interface layer receives raw frames.
3. The real-time processing layer timestamps and buffers data.
4. Data is transferred to the non-real-time processing layer.
5. Frames are decoded into signals.
6. Monitoring algorithms evaluate signal behavior.
7. Data is stored and notifications are generated as needed.

---

6. Mapping to System Requirements

This architecture supports the system requirements as follows:

- **FR-01 to FR-03**: Implemented by the CAN interface and real-time layers.
- **FR-04**: Addressed by the data transfer between processing layers.
- **FR-05 to FR-07**: Implemented in the non-real-time processing layer.
- **FR-08 and FR-09**: Supported by the user and data interaction layer.

Non-functional requirements related to separation, maintainability, and learning are satisfied through the layered structure.

---

7. What This Architecture Does Not Define

This document intentionally does **not** define: - Specific hardware components or pin assignments - Communication protocols between processors - CAN message or signal definitions - Algorithm details or thresholds

These will be addressed in subsequent design documents.

---

8. Next Steps

With the high-level architecture defined, the next steps are:

- Selection of hardware components
- Definition of inter-processor communication
- Detailed software architecture for each processing layer

This staged approach ensures clarity, traceability, and maintainability as the project progresses.