Arduino UNO Q Architecture Explained

1. Purpose of This Document

This document explains the **hardware architecture of the Arduino UNO Q development board** and its role in the CAN-based embedded monitoring system. The goal is to provide a clear understanding of the MCU and Linux-capable processor split, the responsibilities of each processor, and why this board is well-suited for the project.

This is the first step in Phase 1 – Hardware Architecture, focusing entirely on **high-level architecture**, without delving into detailed wiring, peripherals, or CAN integration.

---

2. Board Overview

The Arduino UNO Q is a dual-processor development board featuring:

- **Real-Time Microcontroller (MCU)**
- Handles time-critical tasks
- Interfaces with external peripherals like the CAN controller
- **Linux-Capable Processor (MPU)**
- Runs a Linux environment
- Performs higher-level data processing, storage, and analytics

This architecture allows for a **clear separation of real-time and non-real-time responsibilities**.

---

3. Processor Roles and Responsibilities

3.1 Microcontroller (MCU)

Responsibilities: - Real-time CAN frame reception - Timestamping received frames - Buffering frames for transfer to Linux processor - Handling deterministic, time-sensitive tasks

The MCU operates in isolation from high-level processing, ensuring **predictable real-time performance**.

3.2 Linux-Capable Processor (MPU)

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

By offloading non-real-time tasks to Linux, the system remains **flexible and extensible** without compromising real-time behavior.

---

4. Internal Architecture Concept

At a conceptual level, the Arduino UNO Q architecture can be represented as follows:

```
+---------------------------+
| Linux-Capable Processor   |
|  - Data decoding          |
|  - Monitoring algorithms  |
|  - Storage & notification |
+-------------+-------------+
              |
      Buffered CAN frames
              |
+-------------v-------------+
| Real-Time Microcontroller  |
|  - CAN reception           |
|  - Timestamping            |
|  - Frame buffering         |
+-------------+-------------+
              |
       SPI / CAN Interface
              |
+-------------v-------------+
| External CAN Controller    |
+---------------------------+
```

This diagram emphasizes **responsibility separation** without showing low-level pin or wiring details.

---

5. Why Arduino UNO Q Was Chosen

The board was selected because it:

- Combines an MCU and Linux processor on a single platform
- Supports **real-time peripheral handling** alongside flexible software execution
- Simplifies the learning curve for beginners
- Provides industry-relevant architecture for embedded monitoring projects
- Allows incremental hardware and software exploration

This combination makes the board ideal for teaching, prototyping, and demonstrating **dual-processor embedded design**.

---

6. Next Steps

After understanding the UNO Q architecture, the next documents will focus on:

- **PDF 1.2 – Why MCP2515? Design Trade-Offs**
- Explaining why an external CAN controller is used
- Hardware trade-offs and benefits
- **PDF 1.3 – Hardware Block Diagram**
- High-level wiring and connectivity
- **PDF 1.4 – Wiring & Power Considerations**
- How to safely power and connect the board to peripherals

These steps will complete Phase 1 – Hardware Architecture.