CHALMERS
UNIVERSITY OF TECHNOLOGY

# Simulation of battery management system using CANoe

Degree project report in Electrical and Electronics Engineering

Ahmad Khalaili

DEPARTMENT OF Electrical Engineering

Chalmers University of Technology
Gothenburg, Sweden 2026
www.chalmers.se

Bachelor Thesis

# Simulation of a Battery Management System Using CANoe

Modeling and CAN-Based Communication in an Electrified Vehicle Battery System

Ahmad Khalaili


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2026

Simulation of a Battery Management System Using CANoe
Modeling and CAN-Based Communication in an Electrified Vehicle Battery System
Ahmad Khalaili



Industrial Supervisor: Tariq Omari, Improve Engineering
University Supervisor: Pingal Pratyush Nath, Department of Electrical Engineering
Examiner: John Camilleri, Department of Computer Science and Engineering

Degree project report 2026
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

# Chapter 1

# Abstract

The rapid electrification of the automotive sector has increased demand for early-stage validation tools that can verify the behaviour of battery-related electronic control units before physical prototypes are available. This thesis presents the design and implementation of a simulation environment for a Battery Management System (BMS) built on the Vector CANoe platform, developed in collaboration with Improve Engineering.

The simulation environment models a distributed battery network consisting of sixteen Battery Pack Controller (BPC) nodes communicating over a CAN FD bus. The network architecture is formally described using an AUTOSAR XML system description (ARXML), which defines all ECU nodes, CAN frames, Protocol Data Units (PDUs), and individual signals. Active ECU behaviour is implemented for one BPC node (`BPC_00`) using CAPL (Communication Access Programming Language), while the remaining fifteen nodes provide the structural topology of a scalable distributed battery system, supported by a pre-existing Vector simulation environment.

Battery-related signals, including minimum and maximum cell voltages, pack voltage, temperature, capacity, pack energy, and state of charge, are generated dynamically within the active CAPL node and transmitted at regular intervals over the CAN network. A fault detection subsystem monitors temperature and cell voltage imbalance, triggering internal alert states that are visualised through a custom CANoe panel. Diagnostic services based on the Unified Diagnostic Services (UDS) protocol are implemented using a CANdela diagnostic description file, enabling the simulated ECU to respond to requests for ECU identification, serial number, battery voltage, and odometer readings.

Verification of the simulation is performed using CANoe monitoring tools, including the Trace Window, the Diagnostic Console, and logged measurement data. Analysis of measurement log files confirms that all sixteen BPC nodes transmit CAN FD frames at a mean cycle time of 100.00 ms across a 21-second observation period, demonstrating correct and deterministic communication behaviour. Fault alerting and diagnostic response handling are confirmed through targeted test scenarios.

The results demonstrate that a structured, AUTOSAR-based CANoe simulation environment can effectively serve as a validation platform for CAN communication, diagnostic services, and fault detection logic in early automotive development phases.

**Keywords:** Battery Management System, CAN FD, AUTOSAR, UDS diagnostics, CANoe, CAPL, ECU simulation, automotive embedded systems.

# Acknowledgements

I would like to express my sincere gratitude to everyone who contributed to the completion of this bachelor's thesis.

First and foremost, I thank Improve Engineering for providing the opportunity to conduct this project and for offering valuable insight into industrial automotive development practices. Working with professional toolchains such as Vector CANoe and CANdelaStudio in an industrial context has been an invaluable learning experience.

I am grateful to my industrial supervisor, Tariq Omari, for his guidance, practical expertise, and continuous support throughout the project. His experience in automotive embedded systems development significantly shaped the direction and depth of this work.

I would also like to thank my university supervisor, Pingal Pratyush Nath, for his academic guidance and constructive feedback during the project period. Special thanks to the examiner, John Camilleri, for his time and academic oversight.

Finally, I would like to thank my family and friends for their patience and encouragement throughout my studies at Chalmers University of Technology.

Ahmad Khalaili, Gothenburg, May 2026

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| **ARXML** | AUTOSAR XML System Description File |
| **AUTOSAR** | AUTomotive Open System ARchitecture |
| **BMS** | Battery Management System |
| **BPC** | Battery Pack Controller |
| **CAN** | Controller Area Network |
| **CAN FD** | Controller Area Network with Flexible Data-rate |
| **CAPL** | Communication Access Programming Language |
| **CDD** | CANdela Diagnostic Description |
| **DID** | Data Identifier |
| **DLC** | Data Length Code |
| **ECU** | Electronic Control Unit |
| **HIL** | Hardware-in-the-Loop |
| **ISO** | International Organization for Standardization |
| **MIL** | Model-in-the-Loop |
| **NRC** | Negative Response Code |
| **PDU** | Protocol Data Unit |
| **SID** | Service Identifier |
| **SIL** | Software-in-the-Loop |
| **SoC** | State of Charge |
| **UDS** | Unified Diagnostic Services |

# Nomenclature

Below is the nomenclature of parameters and variables that have been used throughout this thesis.

### Parameters

| | |
|---|---|
| $V_{\text{cell,min}}$ | Minimum cell voltage measured in the battery module |
| $V_{\text{cell,max}}$ | Maximum cell voltage measured in the battery module |
| $V_{\text{pack}}$ | Total battery pack voltage |
| $T$ | Battery module temperature |
| $C$ | Battery module capacity |
| $E_{\text{pack}}$ | Estimated battery pack energy |
| SoC | State of Charge of the battery pack |
| $\Delta V$ | Difference between maximum and minimum cell voltage |

### Variables

| | |
|---|---|
| $v_{\text{min}}$ | Simulated minimum cell voltage generated in the CAPL model |
| $v_{\text{max}}$ | Simulated maximum cell voltage generated in the CAPL model |
| $v_{\text{avg}}$ | Average cell voltage used to estimate pack voltage |
| soc | Calculated state of charge in percentage |
| $\text{soc}_{\text{raw}}$ | Raw encoded state-of-charge value transmitted on the CAN bus |
| capacity | Simulated battery capacity value |
| voltage | Simulated diagnostic battery voltage value |
| ret | Return value from CAPL diagnostic parameter setting functions |

# Contents

**1 Abstract** **v**

**Acknowledgements** **vii**

**List of Acronyms** **ix**

**Nomenclature** **xi**

**2 Introduction** **1**
2.1 Background . . . 1
2.2 Problem Statement . . . 2
2.3 Purpose and Scope . . . 2
2.4 Objectives . . . 3
2.5 Limitations . . . 3
2.6 Thesis Structure . . . 4

**3 Theoretical Background** **5**
3.1 Battery Management Systems in Electric Vehicles . . . 5
3.2 Controller Area Network and CAN FD . . . 6
3.3 AUTOSAR Communication Architecture . . . 6
3.4 Unified Diagnostic Services . . . 7
3.5 Simulation-Based Development in the Automotive V-Model . . . 7

**4 System Design and Implementation** **9**
4.1 System Architecture Overview . . . 9
4.2 Communication Database Design . . . 10
4.2.1 AUTOSAR ARXML Architecture . . . 10
4.2.2 BPC State PDU Signal Definition . . . 10
4.2.3 Maintenance PDU . . . 11
4.3 BPC ECU Simulation Using CAPL . . . 11
4.3.1 Timer-Based Cyclic Transmission . . . 11
4.3.2 Battery Signal Generation . . . 11
4.4 Fault Detection and Alert Subsystem . . . 12
4.4.1 Temperature Alert . . . 12
4.4.2 Cell Voltage Imbalance Alert . . . 12
4.5 UDS Diagnostic Service Implementation . . . 13
4.5.1 Diagnostic Description File . . . 13
4.5.2 CAPL Diagnostic Event Handlers . . . 13
4.6 Measurement Setup and Monitoring Configuration . . . 14
4.7 Automated Functional Testing . . . 15

**5 Results** **17**
5.1 CAN Communication Behaviour . . . . . 17
5.2 Decoded Signal Values . . . . . 18
5.3 Fault Alert Verification . . . . . 19
5.4 Diagnostic Communication Results . . . . . 19
5.5 Functional Test Outcome . . . . . 20

**6 Discussion** **21**
6.1 Evaluation Against Objectives . . . . . 21
6.2 Design Decisions and Trade-offs . . . . . 22
6.3 Limitations . . . . . 22
6.4 Relation to Industrial Practice . . . . . 24
6.5 Ethical and Sustainability Considerations . . . . . 24

**7 Conclusion** **27**
Future Work . . . . . 28

**Bibliography** **29**

**Appendix A: Annotated CAPL Source Code (BPC_00)** **31**

**Appendix B: AUTOSAR PDU Signal Definitions** **35**

# List of Figures

4.1 CANoe simulation topology showing sixteen BPC ECU nodes and one BMS node connected to the BATTERY CAN FD network (Channel 2). `BPC_00` (leftmost, bottom row) is the actively developed node; `BPC_01`–`BPC_15` transmit via the Vector TrainingCar simulation environment. . . . . 9
4.2 AlertPanel for `BPC_00`. At 60 °C the temperature gauge needle is displaced and the alert LED changes from grey to red, confirming correct operation of the temperature alert mechanism. . . . . 12
4.3 UDS diagnostic request–response sequence for the `ECU_Serial_Number_Read` service. The tester sends a ReadDataByIdentifier request; the simulated ECU responds with the positive response (SID `0x62`) containing the serial number `'BPC0'`. . . . . 14
4.4 CANoe Measurement Setup for the simulation, showing CAN Statistics, Graphics, Trace, Data, and Logging modules connected to the simulation source. The Logging module records all CAN traffic to binary and ASC log files for off-line analysis. . . . . 14

5.1 CANoe Trace Window showing a decoded `pduStateBPC_00` AUTOSAR PDU. The frame is transmitted at CAN ID `0x10` (decimal 16) on CAN FD Channel 1, with a DLC of 10 (indicating 14 bytes of payload in CAN FD encoding). Decoded signal values are shown below the frame entry. . . . . 18

# List of Tables

4.1 Signal definitions for the BPC state PDU (`pduStateBPC_00`). All signals use little-endian byte ordering. Total frame length is 14 bytes (112 bits), requiring CAN FD. . . . . 10
4.2 UDS diagnostic services implemented in `BPC_00`, with response parameters and encoding methods. . . . . 13

5.1 CAN communication statistics derived from analysis of ASC log files. Cycle time statistics are computed from 211 consecutive frame timestamps for `BPC_01`, representative of all active nodes. . . . . 17
5.2 Decoded signal values from a representative `pduStateBPC_00` frame observed in the CANoe Trace Window (Figure 5.1). All values fall within their specified ranges, confirming correct signal encoding and AUTOSAR PDU mapping. . 18
5.3 Results of UDS diagnostic service verification using the CANoe Diagnostic Console. All four services returned positive responses with the expected parameter values. . . . . 19

1 Complete AUTOSAR signal definitions for the BPC state PDU (`pduStateBPC_xx`). Note: `sigAlertHighTemp` and `sigAlertUnbalanced` are implemented as CANoe system variables, not as AUTOSAR-defined I-Signals, and are therefore not present in the ARXML file. They are included here for completeness. . . 35
2 Signal definitions for the Maintenance PDU (`frmMaintenance`, CAN ID `0x01`, 6 bytes). This PDU carries fleet-wide battery management commands from the BMS node to all BPC nodes. . . . . 36

# Chapter 2

# Introduction

## 2.1 Background

The automotive industry is undergoing a fundamental transition driven by the rapid electrification of passenger and commercial vehicles, largely motivated by increasingly stringent emissions regulations, declining battery costs, and growing consumer demand for sustainable transportation alternatives. Battery-powered electric vehicles (EVs) and hybrid electric vehicles (HEVs) depend on high-voltage battery systems that must operate safely, reliably, and efficiently across a wide range of conditions. At the centre of this requirement is the Battery Management System (BMS), a critical set of electronic control units responsible for monitoring battery cell conditions, detecting fault states, communicating battery status to the rest of the vehicle, and ensuring that the battery operates within safe limits [1].

In modern electric vehicles, the battery pack is frequently distributed across multiple modules, each monitored by a dedicated Battery Pack Controller (BPC). These distributed controllers continuously measure parameters such as individual cell voltages, module temperature, and state of charge (SoC), and report this information to a supervisory BMS node over an in-vehicle communication bus. The Controller Area Network (CAN) protocol remains the dominant communication standard for such inter-ECU data exchange in automotive systems. CAN offers real-time capability, deterministic arbitration, and robust fault tolerance. It has remained dominant over alternatives such as LIN, FlexRay, and Automotive Ethernet in many vehicle network domains due to its low implementation cost, widespread toolchain support, and proven reliability in safety-critical embedded environments. FlexRay offers higher bandwidth but at significantly greater hardware and integration cost, while Automotive Ethernet provides substantially higher throughput but requires more complex network infrastructure and is typically reserved for high-bandwidth applications such as camera systems or over-the-air software updates [2].

The development and validation of distributed battery architectures of this kind presents significant engineering challenges. Each BPC must correctly encode and transmit battery state data; gateway ECUs must correctly receive, process, and forward this data between different vehicle network domains; and diagnostic tools must be able to query any ECU in the network to retrieve status information during development and in the field. Validating all of these interactions using physical hardware is expensive, time-consuming, and impractical during the early phases of development when hardware is not yet available [3].

Simulation-based development has therefore become an essential methodology in the automotive industry. By modelling ECU behaviour in a virtual environment, engineers can test

communication protocols, diagnostic services, and system-level interactions before any physical hardware exists. Vector CANoe is the industry-standard tool for this purpose, providing an integrated environment for CAN network simulation, ECU behaviour modelling using the CAPL programming language, diagnostic testing, and measurement analysis [4].

To investigate these challenges in an industrial context, this thesis was conducted in collaboration with Improve Engineering, a Gothenburg-based consultancy specialising in systems engineering, embedded software development, and quality assurance for the automotive sector. Improve works with major automotive customers including AB Volvo, Volvo Cars, and Polestar, and provides professional training and consultancy services in tools such as CANoe and CANalyzer. The project was motivated by the practical need to demonstrate and evaluate a structured CANoe-based simulation methodology for a distributed BMS architecture.

## 2.2 Problem Statement

Validating the communication behaviour, diagnostic functionality, and fault detection logic of a distributed Battery Management System requires a test environment that can realistically represent the interactions between multiple Battery Pack Controllers, a supervisory BMS node, and external diagnostic tools. Constructing such an environment using physical ECU hardware is costly and unavailable during the early development phase.

This thesis investigates how a structured simulation environment in Vector CANoe can be designed and implemented to model a multi-ECU battery communication architecture, validate diagnostic services, and demonstrate fault alerting behaviour in the absence of physical hardware.

## 2.3 Purpose and Scope

The purpose of this thesis is to design and implement a CAN-based simulation environment for a Battery Management System using Vector CANoe. The simulation environment is intended to model the communication behaviour of a distributed battery network, implement diagnostic services according to the UDS standard, and demonstrate fault detection alerting within a controlled virtual environment.

The project was structured to prioritise core simulation fidelity, ensuring that the communication architecture, signal generation logic, and diagnostic layer were thoroughly validated within the CANoe environment before addressing more advanced integration features. Advanced extensions — such as integration of a physical hardware ECU via Arduino, inter-network gateway routing between a battery domain and a powertrain domain, and a CI/CD pipeline for automated nightly testing — were identified as valuable future work and are addressed as such in Chapter 7. This prioritisation allowed for a deeper and more verifiable implementation of the core simulation components.

The scope of the thesis is therefore limited to: AUTOSAR-based network modelling for a battery CAN network; CAPL-based ECU simulation for one actively developed BPC node within a sixteen-node architectural topology; UDS diagnostic service implementation; fault detection and visualisation using CANoe system variables and the Panel Designer; and verification using CANoe monitoring tools and logged measurement data.

## 2.4 Objectives

The following objectives were defined for the project:

1. Design a CAN network architecture for a distributed battery system using an AUTOSAR system description (ARXML), defining sixteen BPC nodes, one BMS node, and all associated CAN frames, PDUs, and signals.
2. Implement active ECU behaviour for the `BPC_00` node using CAPL, including periodic transmission of battery state signals and event-driven fault detection logic.
3. Implement UDS-based diagnostic services for the BPC node using a CANdela diagnostic description file, covering ECU identification, serial number reading, battery voltage reading, and odometer reading.
4. Implement a fault detection and visualisation subsystem using CANoe system variables and the Panel Designer to monitor temperature alerts and cell voltage imbalance.
5. Develop a CAPL-based automated test case within the CANoe Test Environment to verify communication and signal behaviour.
6. Verify the complete simulation environment using CANoe monitoring tools, logged measurement data, and the Diagnostic Console.

## 2.5 Limitations

This thesis is subject to the following limitations, which are acknowledged explicitly to characterise the scope of the work:

- The simulation environment does not include integration with a physical hardware ECU. Hardware-in-the-loop (HIL) testing using an Arduino-based ECU was considered during planning but was not implemented in the final system.
- The battery network operates as a single CAN domain. No active gateway routing or signal translation between a battery network and a powertrain network was implemented in the BMS node, although the network topology includes a BMS ECU node for this purpose.
- Of the sixteen BPC nodes defined in the AUTOSAR topology, only `BPC_00` contains custom CAPL simulation logic. The remaining fifteen nodes transmit CAN FD traffic through the underlying Vector TrainingCar simulation environment, providing a realistic multi-node communication background without requiring custom CAPL development for each node.
- The battery behaviour model is intentionally simplified and does not represent electrochemical dynamics, temperature gradients, or ageing effects. The model is adequate for communication and diagnostic validation but not for physical battery analysis.
- The SoC estimation is based on a linear voltage-to-SoC mapping rather than a physics-based model.
- No CI/CD pipeline for automated test execution was implemented.
- The simulation does not include boundary condition testing for communication parameters. For example, while the nominal message transmission period is 100 ms, no systematic investigation was conducted into the system's behaviour under atypical timing conditions such as 10 ms or 1000 ms message intervals. Similarly, the behaviour of

signal values at the extremes of their defined ranges, as well as the system's response to out-of-range inputs, was not systematically characterised. A complete validation environment would specify and verify these boundary conditions for all communication parameters. This is acknowledged as a limitation of the current scope.

## 2.6 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 3 provides the theoretical background covering battery management systems, the CAN and CAN FD protocols, AUTOSAR communication architecture, UDS diagnostics, and simulation-based development methodology. Chapter 4 describes the system design and implementation in detail, covering the network architecture, ARXML-based communication modelling, CAPL ECU simulation, the fault detection subsystem, diagnostic implementation, and the measurement setup. Chapter 5 presents the results of simulation verification, including communication timing measurements, decoded signal values, fault alert behaviour, and diagnostic test outcomes. Chapter 6 discusses the results in terms of design decisions, limitations, relation to industrial practice, and ethical and sustainability considerations. Chapter 7 concludes the thesis and outlines directions for future work.

# Chapter 3

# Theoretical Background

## 3.1 Battery Management Systems in Electric Vehicles

A Battery Management System (BMS) is the electronic system responsible for supervising, monitoring, and protecting a rechargeable battery pack throughout its operational lifetime. In electric vehicles, the BMS performs several critical functions:

- Monitoring individual cell voltages and temperatures
- Estimating the state of charge (SoC) and state of health (SoH)
- Balancing charge distribution across cells
- Communicating battery status to the vehicle control network
- Initiating protective actions when predefined operating limits are exceeded

These functions are essential for ensuring battery safety, reliability, performance, and longevity throughout the vehicle's service life [1].

Modern EV battery packs are composed of large numbers of lithium-ion cells arranged in series and parallel configurations to achieve the required voltage and energy capacity. Because individual cells exhibit slight variations in electrical characteristics due to manufacturing tolerances, operating conditions, and ageing effects, each cell must be monitored independently. This requirement motivates the use of a distributed BMS architecture, in which multiple Battery Pack Controllers (BPCs), each responsible for a number of the cells, report measurements to a central supervisory BMS node [5].

Each BPC continuously measures the minimum and maximum cell voltages within its module, the module temperature, the estimated pack voltage, and the state of charge. These measurements are typically performed at intervals ranging from tens to hundreds of milliseconds, with shorter intervals providing more responsive fault detection at the expense of increased communication bus utilisation. In the simulation developed in this thesis, a cyclic transmission period of 100 ms is used, representing a common configuration for battery management communication systems. The measured parameters are encoded and transmitted over the vehicle communication bus at regular intervals. The supervisory BMS aggregates this information, applies system-level fault logic, and communicates relevant data to other vehicle systems such as the instrument cluster and the powertrain controller.

Two operating conditions of particular importance in BMS design are cell voltage imbalance and over-temperature. Cell imbalance, defined as the difference $\Delta V$ between the maximum and minimum cell voltages within a module, indicates uneven charge distribution that may

accelerate cell degradation and reduce overall battery performance. Over-temperature conditions present significant safety risks, including thermal runaway, a potentially catastrophic failure mode in lithium-ion systems [5].

State-of-charge estimation is a fundamental BMS function. A commonly used simplified approach maps the measured open-circuit cell voltage to an SoC value using a predefined voltage window. For a typical lithium-ion cell, a cell voltage of approximately 3.40,V corresponds to 0of approximately 4.10,V corresponds to 100expressed as:

$$\text{SoC}\ (\%) \;=\; \frac{V_{\text{cell}} - V_{\text{min}}}{V_{\text{max}} - V_{\text{min}}} \times 100 \tag{3.1}$$

where $V_{\text{cell}}$ is the measured average cell voltage, $V_{\text{min}} = 3.40\,\text{V}$, and $V_{\text{max}} = 4.10\,\text{V}$. This approach is a simplification of more accurate methods such as coulomb counting or extended Kalman filtering, but is sufficient for communication validation purposes [5].

## 3.2 Controller Area Network and CAN FD

The Controller Area Network (CAN) protocol, originally developed by Robert Bosch GmbH and standardised as ISO 11898, is the dominant communication protocol for in-vehicle ECU networks. CAN provides a multi-master, broadcast-based communication architecture in which all nodes share a common differential bus. A transmitting node asserts its message onto the bus, and all other nodes simultaneously receive it [2].

A standard CAN data frame consists of a start-of-frame bit, an 11-bit arbitration identifier (or 29 bits in the extended format), a 6-bit control field containing the Data Length Code (DLC), up to 8 bytes of payload data, a 16-bit cyclic redundancy check (CRC), and an acknowledgement field. The arbitration mechanism, Carrier Sense Multiple Access with Collision Detection by Bit Arbitration (CSMA/CD-BA), ensures that when multiple nodes attempt to transmit simultaneously, the message with the lowest identifier value gains access to the bus without data loss or communication collisions [2].

The original CAN protocol limits the payload to 8 bytes per frame, which is insufficient for many modern automotive applications requiring larger data packets. CAN with Flexible Data-rate (CAN FD), standardised in ISO 11898-1:2015, extends this limit to 64 bytes per frame while also allowing the data phase to be transmitted at a higher bitrate than the arbitration phase. In this thesis, each BPC state frame carries 14 bytes of payload data. Since this exceeds the 8-byte payload limit imposed by the classic CAN standard, the messages are transmitted using CAN FD on the simulated battery network [2, 6].

The arbitration identifier determines message priority. In the AUTOSAR network description used in this project, BPC state frame identifiers are assigned sequentially from `0x10` (decimal 16) for `BPC_00` through `0x1F` (decimal 31) for `BPC_15`. The maintenance frame carrying fleet-wide balance and bypass status is assigned identifier `0x01`, giving it the highest priority on the battery bus.

## 3.3 AUTOSAR Communication Architecture

AUTOSAR (AUTomotive Open System ARchitecture) is an industry-standard framework for the development of automotive software and communication systems. The AUTOSAR

layered architecture separates application logic from underlying communication infrastructure, enabling ECU software to be developed and integrated independently of the specific hardware and network configuration [7].

In the context of CAN communication, AUTOSAR defines a hierarchical description model in which individual data items (I-Signals) are grouped into I-Signal Protocol Data Units (I-Signal I-PDUs), which are in turn mapped to CAN frames. This mapping is described in an ARXML file, which is the XML-based system description format defined by the AUTOSAR standard. The ARXML specification provides a formal description of all ECU nodes, their communication roles, the signals they transmit and receive, the bit positions and widths of individual signals within each PDU, and the byte ordering of multi-byte fields [7].

Vector CANoe can import ARXML files and use them as the communication database for a simulation configuration. This enables CAPL code to access signals by their symbolic names (e.g., `$BPC_00::pduStateBPC_00::sigPackVoltage`) rather than by raw byte offsets, which greatly simplifies simulation programming and reduces the risk of encoding errors.

The separation of communication specification from implementation logic is a core principle of AUTOSAR. It ensures that if the signal layout changes, only the ARXML description needs to be updated, while the CAPL logic that references signals by name remains valid. This approach supports scalability: adding a new BPC node requires adding its node and frame definitions to the ARXML, not modifying existing CAPL code.

## 3.4 Unified Diagnostic Services

Unified Diagnostic Services (UDS), standardised in ISO 14229-1, defines a set of diagnostic communication services for automotive ECUs. UDS follows a client–server model in which a diagnostic tester (client) sends requests to a target ECU (server), and the ECU responds with either a positive response containing the requested data or a negative response containing a Negative Response Code (NRC) indicating the reason for failure [8].

Each UDS service is identified by a one-byte Service Identifier (SID). The most commonly used service for data retrieval is ReadDataByIdentifier (SID `0x22`), which allows the tester to read the current value of a named data element from the ECU. Each data element is addressed by a two-byte Data Identifier (DID). The ECU must respond with a positive response (SID `+ 0x40` = `0x62`) containing the data value, or a negative response (SID `0x7F`) with the NRC.

In practice, UDS diagnostic descriptions for a given ECU are defined in a CANdela Diagnostic Description (CDD) file, created using Vector CANdelaStudio. The CDD file specifies all services, DIDs, their expected data types and lengths, and the encoding of response parameters. Within CANoe, CAPL code can intercept incoming diagnostic requests using the `on diagRequest` event handler, construct a response object using the CAPL diagnostic API, populate the response parameters, and transmit the response using `diagSendResponse()`.

## 3.5 Simulation-Based Development in the Automotive V-Model

Automotive software and systems development is typically structured according to the V-model, in which requirements and design activities on the left side of the V are paired with corresponding test and validation activities on the right side [3]. Simulation plays a role at

multiple levels of this model, with different degrees of hardware involvement.

Model-in-the-Loop (MIL) testing validates the behaviour of a software model running within a simulation environment, with no interaction with real hardware. Software-in-the-Loop (SIL) testing executes compiled production code within a host simulation environment. Hardware-in-the-Loop (HIL) testing runs production code on the target ECU hardware, interfaced to a simulation environment that replaces the physical vehicle [3].

CANoe-based ECU simulation as practised in this thesis falls primarily within the SIL category: the CAPL-based ECU models execute within the CANoe simulation environment, interacting with the simulated CAN network. This approach enables validation of communication behaviour, diagnostic services, and fault detection logic without requiring physical hardware, and is widely used in industrial practice for early-stage ECU development and integration testing [4].

CAPL (Communication Access Programming Language) is the event-driven programming language integrated into CANoe. CAPL applications are structured around event handlers that respond to network events such as incoming CAN messages, diagnostic requests, system variable changes, and timer expirations. This programming paradigm closely reflects the operational behaviour of automotive ECUs, which must process incoming messages, execute periodic tasks, and react to sensor inputs and system state changes in real time. As a result, CAPL provides an effective framework for ECU simulation and validation within the CANoe environment [9].

# Chapter 4

# System Design and Implementation

## 4.1 System Architecture Overview

The simulation environment consists of a single CAN FD network, referred to as the `BATTERY` network, on which sixteen Battery Pack Controller nodes (`BPC_00` through `BPC_15`) and one Battery Management System node (`BMS`) communicate. The overall network topology is shown in Figure 4.1.

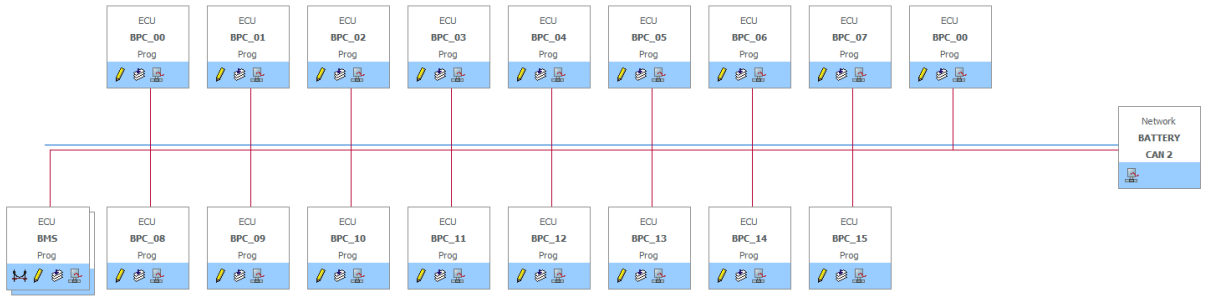



**Figure 4.1:** CANoe simulation topology showing sixteen BPC ECU nodes and one BMS node connected to the BATTERY CAN FD network (Channel 2). `BPC_00` (leftmost, bottom row) is the actively developed node; `BPC_01`–`BPC_15` transmit via the Vector TrainingCar simulation environment.

Of the sixteen BPC nodes, only `BPC_00` contains custom-developed CAPL simulation logic, diagnostic service handlers, and manually implemented signal behaviour. The remaining fifteen nodes (`BPC_01`–`BPC_15`) transmit CAN FD frames through the underlying Vector TrainingCar simulation environment, which is integrated into the CANoe project configuration. This architecture provides the communication context of a realistic distributed battery network while focusing active development effort on a single fully instrumented ECU node.

The BMS node serves as the logical integration point within the network topology. It receives battery state information from all BPC nodes and serves as the termination point for fleet-wide maintenance commands broadcast via the Maintenance PDU. No active CAPL-based gateway routing between the battery network and an external powertrain network was implemented in this project; the BMS node's role is confined to the battery domain.

The simulation configuration additionally includes a `frmVehicleInformation` frame (CAN ID `0x02`) on the battery network, which carries vehicle-level status information generated by the

TrainingCar simulation environment.

## 4.2 Communication Database Design

### 4.2.1 AUTOSAR ARXML Architecture

The network topology, ECU nodes, CAN frames, PDUs, and signals are formally specified in an AUTOSAR ARXML system description file. This file serves as the communication database imported into CANoe, enabling all simulation nodes and CAPL code to reference signals by their symbolic names. The hierarchical structure of the AUTOSAR communication model follows the chain: I-Signal → I-Signal I-PDU → CAN Frame → CAN Frame Triggering.

The ARXML defines two distinct PDU groups on the BATTERY network: the BPC state PDUs (one per BPC node) and the Maintenance PDU. All frames use standard 11-bit CAN identifiers and little-endian (MOST-SIGNIFICANT-BYTE-LAST) byte ordering. BPC state frames carry 14 bytes of payload, which exceeds the 8-byte limit of classic CAN and requires CAN FD framing.

### 4.2.2 BPC State PDU Signal Definition

Each of the sixteen BPC nodes transmits a dedicated state PDU (`pduStateBPC_xx`) mapped to a corresponding CAN FD frame (`frmStateBPC_xx`). The frame identifiers follow a sequential scheme: `BPC_00` uses CAN ID `0x10` (decimal 16), `BPC_01` uses `0x11`, and so on through `BPC_15` at `0x1F`. The PDU contains nine signals, defined in Table 4.1.

**Table 4.1:** Signal definitions for the BPC state PDU (`pduStateBPC_00`). All signals use little-endian byte ordering. Total frame length is 14 bytes (112 bits), requiring CAN FD.

| Signal Name | Width | Start | Description | Simulated Range |
|---|---|---|---|---|
| `sigAlertBypassed` | 1 bit | 0 | Cell bypass alert flag | 0 / 1 |
| `sigAlertShortcut` | 1 bit | 1 | Short-circuit alert flag | 0 / 1 |
| `sigCellVoltageMi` | 16 bit | 8 | Minimum cell voltage (V) | 3.40 – 4.10 V |
| `sigCellVoltageMa` | 16 bit | 24 | Maximum cell voltage (V) | 3.40 – 4.10 V |
| `sigPackVoltage` | 16 bit | 40 | Battery pack voltage (V) | Derived, 8-cell |
| `sigTemperature` | 16 bit | 56 | Module temperature (°C) | Derived |
| `sigCapacity` | 21 bit | 72 | Battery capacity (Ah) | 20.0 – 80.0 Ah |
| `sigPackEnergy` | 16 bit | 96 | Pack energy (Wh) | Derived |
| `SoC` | 6 bit | 112 | State of charge (raw 0–63) | Mapped from $V_{\text{avg}}$ |

### 4.2.3 Maintenance PDU

The Maintenance PDU (`Maintenance_PDU`), mapped to CAN frame `frmMaintenance` at CAN ID `0x01`, carries 6 bytes of fleet-wide battery network management data. It contains 16 one-bit balance signals (`sigBalanceBPC_00` through `sigBalanceBPC_15`, at start bits 0–15) and 16 one-bit bypass signals (`sigBypassBPC_00` through `sigBypassBPC_15`, at start bits 16–31), plus a 16-bit target voltage field (`sigTargetVoltage`, start bit 32). The Maintenance PDU is generated by the BMS node and received by all BPC nodes, providing the bidirectional communication structure necessary for a realistic fleet management architecture.

## 4.3 BPC ECU Simulation Using CAPL

### 4.3.1 Timer-Based Cyclic Transmission

The active BPC simulation is implemented in the CAPL source file `BPC_00.can`. On simulation start, a millisecond timer (`tSend`) is initialised with a 100 ms period. On each timer expiry, the battery state signals are updated and the timer is restarted, producing a cyclic transmission of the BPC state frame at a nominal rate of 10 Hz.

This event-driven architecture, using the `on start` and `on timer tSend` event handlers, reflects the real-time cyclic task behaviour of an embedded ECU firmware. Signal values are updated before the timer is restarted, ensuring that each transmitted frame reflects the most recent simulated state.

### 4.3.2 Battery Signal Generation

Within each timer cycle, battery signals are generated as follows. A helper function `randRange(min, max)` returns a uniformly distributed pseudo-random float within the specified range, using CANoe's `random()` function scaled to the [min, max] interval.

Cell voltage simulation is handled via a reactive signal-update architecture. The pack voltage and imbalance alert are computed in an `on signal_update` event handler that fires whenever `sigCellVoltageMax` is updated. This handler computes the pack voltage as the average of the minimum and maximum cell voltages multiplied by a series cell count of 8:

$$V_{\text{pack}} \;=\; \frac{V_{\text{cellMax}} + V_{\text{cellMin}}}{2} \times 8 \tag{4.1}$$

Within each 100 ms timer cycle, battery capacity is randomly assigned in the range [20.0, 80.0] Ah. Pack energy is derived as:

$$E_{\text{pack}}\ (\text{Wh}) \;=\; \frac{V_{\text{pack}} \times C}{1000} \tag{4.2}$$

Module temperature is modelled as a simplified linear function of pack voltage:

$$T \;=\; 20.0 \;+\; \frac{V_{\text{pack}}}{500.0} \times 10.0 \quad (^{\circ}\text{C}) \tag{4.3}$$

This simplified model is not intended to represent physical battery thermal dynamics; it introduces a deterministic temperature variation correlated to pack voltage, sufficient for verifying temperature alert logic.

State of charge is estimated using the linear voltage mapping of Equation (3.1). The resulting SoC percentage (0–100%) is converted to a 6-bit raw value (0–63) before being assigned to the SoC signal:

$$\mathrm{SoC}_{\mathrm{raw}} = \frac{\mathrm{SoC} \times 63}{100} \tag{4.4}$$

## 4.4 Fault Detection and Alert Subsystem

The simulation implements two distinct fault detection mechanisms, both implemented as CAPL event handlers using CANoe system variables to propagate alert state to the panel visualisation layer.

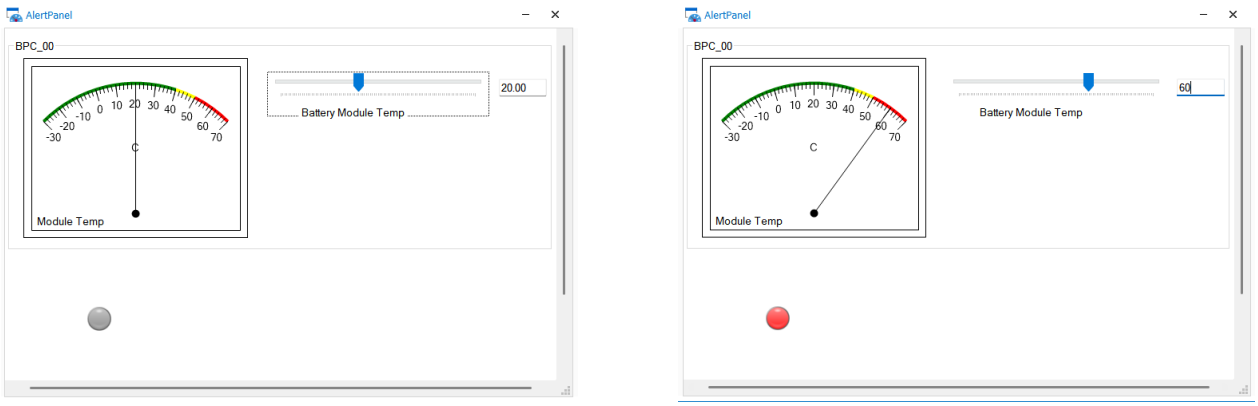
### 4.4.1 Temperature Alert

The temperature alert monitors the CANoe system variable `sysvar::Measurements::BatteryModuleTemp`, which represents the simulated module temperature as set via the AlertPanel slider control. An `on sysvar_update` event handler fires whenever this variable changes and applies the following threshold logic:

$$\texttt{sigAlertHighTemp} = \begin{cases} 1 & \text{if } T > 40\,^{\circ}\mathrm{C} \text{ or } T < 0\,^{\circ}\mathrm{C} \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

A downstream `on signal_update sigAlertHighTemp` handler propagates this state to the system variable `sysvar::Alerts::TempAlert`, which drives the LED indicator on the AlertPanel.

Figures 4.2a and 4.2b show the AlertPanel during normal operation (20 °C, alert inactive) and during an over-temperature scenario (60 °C, alert active), respectively.




**(a)** Normal operation (20 °C). Alert LED inactive (grey).

**(b)** Over-temperature (60 °C). Alert LED active (red).

**Figure 4.2:** AlertPanel for `BPC_00`. At 60 °C the temperature gauge needle is displaced and the alert LED changes from grey to red, confirming correct operation of the temperature alert mechanism.

### 4.4.2 Cell Voltage Imbalance Alert

The cell voltage imbalance alert is triggered by the `on signal_update` handler for `sigCellVoltageMax`. On each update, the handler computes the voltage delta:

$$\Delta V \;=\; V_{\text{cellMax}} - V_{\text{cellMin}} \tag{4.6}$$

If $\Delta V > 0.2\,\text{V}$, `sigAlertUnbalanced` is set to 1, and the downstream handler propagates this to the `Alerts::UnbalancedAlert` system variable. This threshold of 0.2 V is representative of a significant imbalance condition in a lithium-ion module.

## 4.5 UDS Diagnostic Service Implementation

### 4.5.1 Diagnostic Description File

Diagnostic services for the BPC node are defined in a CANdela Diagnostic Description (CDD) file created using Vector CANdelaStudio. The CDD file specifies the four diagnostic services implemented in this project, their Data Identifiers (DIDs), the data types and lengths of their response parameters, and the encoding format for each parameter. Table 4.2 summarises the implemented services.

**Table 4.2:** UDS diagnostic services implemented in `BPC_00`, with response parameters and encoding methods.

| Service Name | SID | Response Parameter | Encoding |
|---|---|---|---|
| `ECU_Serial_Number` | `0x22` | ECU_Serial_Nur (4 B) | ASCII: `'BPC0'` |
| `ECU_Identificatio` | `0x22` | ECU_Name (10 B) | ASCII: `'BPC00SN000'` |
| `BPC_Voltage_Read` | `0x22` | Battery_Voltage | Numeric, 350–400 V |
| `Odometer_Read` | `0x22` | Odometer (4 B) | Raw bytes: `'BPC0'` |

### 4.5.2 CAPL Diagnostic Event Handlers

Each diagnostic service is handled by a dedicated CAPL event handler. When a diagnostic request is received over the CAN bus, CANoe automatically dispatches it to the corresponding `on diagRequest` handler. Within the handler, a `diagResponse` object is created, parameters are populated using either `diagSetParameterRaw()` (for raw byte array parameters such as ASCII identifiers) or `diagSetParameter()` (for numeric parameters such as battery voltage), and the response is transmitted using `diagSendResponse()`.

The return value of each parameter-setting function is checked before transmitting the response. A non-negative return value indicates success; a negative value indicates a mismatch between the parameter definition in the CDD file and the data supplied by the CAPL handler, in which case the response is suppressed and a diagnostic message is written to the CANoe Write Window.

For the battery voltage service, the response value is dynamically generated as a random integer in the range $[350, 400]$ V per request, illustrating the capability to return live simulation

data in diagnostic responses. Figure 4.3 illustrates the diagnostic request–response sequence for the ECU serial number service.

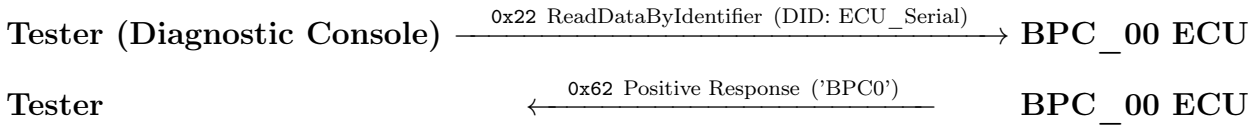



**Figure 4.3:** UDS diagnostic request–response sequence for the `ECU_Serial_Number_Read` service. The tester sends a ReadDataByIdentifier request; the simulated ECU responds with the positive response (SID `0x62`) containing the serial number `'BPC0'`.

## 4.6 Measurement Setup and Monitoring Configuration

CANoe's Measurement Setup is configured with five parallel analysis and logging modules, connected to the simulation data source, as shown in Figure 4.4.

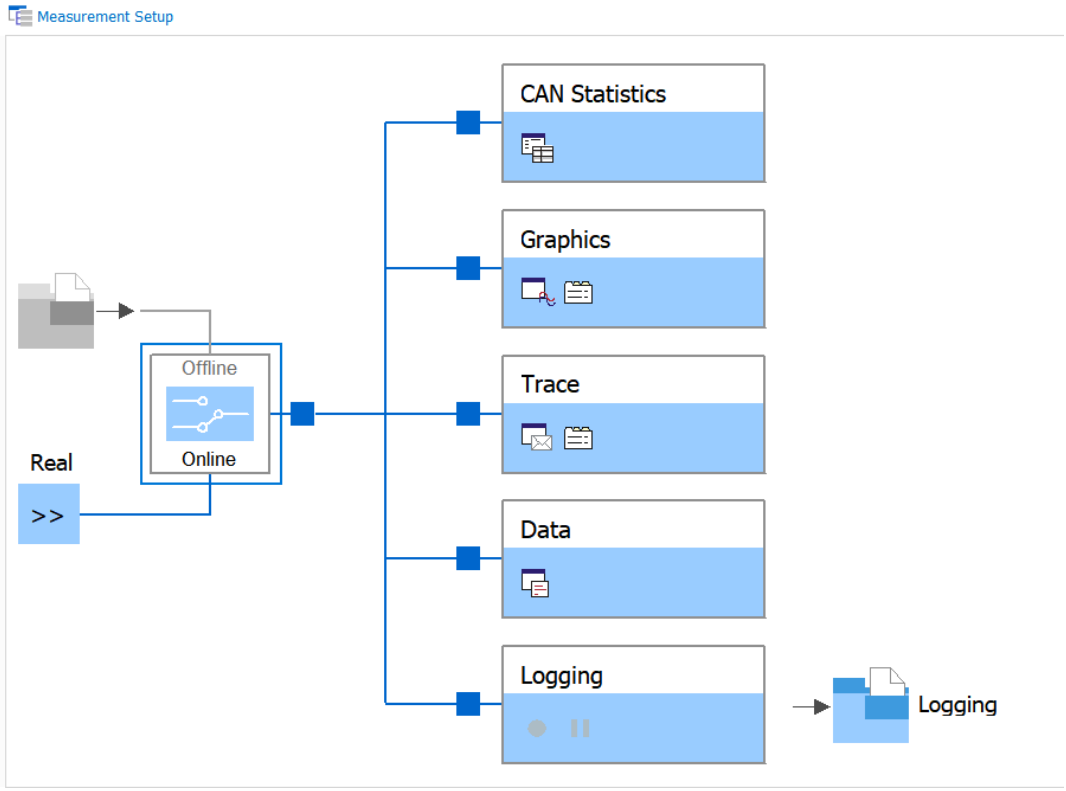



**Figure 4.4:** CANoe Measurement Setup for the simulation, showing CAN Statistics, Graphics, Trace, Data, and Logging modules connected to the simulation source. The Logging module records all CAN traffic to binary and ASC log files for off-line analysis.

The CAN Statistics module provides real-time frame count and bus load indicators. The Graphics module visualises signal values over time as waveforms. The Trace Window captures all CAN frames in chronological order with symbolic signal decoding, enabling detailed inspection of individual messages. The Data module provides a real-time tabular view of current signal values. The Logging module records all CAN traffic to Binary Logging Format (BLF) and ASCII (ASC) log files for subsequent off-line analysis.

## 4.7 Automated Functional Testing

A CAPL-based automated test case is implemented within the CANoe Test Environment in the file `functional_tests.can`. The test case `tc_packVoltage()` verifies that battery capacity values transmitted by `BPC_00` fall within the specified range of 20.0 to 80.0 Ah. The test reads the current value of `sigCapacity` from the `BPC_00` PDU, waits 500 ms for the simulation state to update, and then evaluates the value against the defined bounds using `testStepPass()` or `testStepFail()` to record the test outcome in the CANoe test report.

The test is executed via the `MainTest()` entry point within the CANoe Test Environment, and its outcome is recorded in a structured test report generated by CANoe. This automated test demonstrates the principle of functional validation within the simulation loop, and provides a foundation for extending the test suite to cover additional signal ranges, timing constraints, and diagnostic response verification in future iterations.

# Chapter 5

# Results

This chapter presents the results of simulation verification, conducted using CANoe monitoring tools, logged measurement data, and targeted test scenarios. All results are based on observations from the running simulation environment; no physical hardware was involved.

## 5.1 CAN Communication Behaviour

Analysis of the captured measurement log files confirms that all sixteen BPC nodes actively transmit CAN FD frames on the BATTERY network. Over a 21-second measurement window (captured across two consecutive log files), each BPC node transmitted 211 frames, for a total of 3 376 BPC state frames. Combined with 2 110 Maintenance PDU frames and 211 `frmVehicleInformation` frames, the total CAN FD frame count is 5 697 frames in 21 seconds, corresponding to an aggregate frame rate of approximately 271 frames per second. Table 5.1 summarises the communication statistics.

**Table 5.1:** CAN communication statistics derived from analysis of ASC log files. Cycle time statistics are computed from 211 consecutive frame timestamps for `BPC_01`, representative of all active nodes.

| Metric | Value |
|---|---|
| Observation duration | 21.0 s |
| BPC state frames per node | 211 frames |
| Total BPC state frames (16 nodes) | 3 376 frames |
| Maintenance PDU frames | 2 110 frames |
| Vehicle Information frames | 211 frames |
| Total CAN FD frames | 5 697 frames |
| Aggregate frame rate | ≈271 frames/s |
| Mean cycle time (`BPC_01`) | 100.00 ms |
| Minimum observed cycle time | 89.47 ms |
| Maximum observed cycle time | 112.70 ms |
| Nominal transmission period | 100 ms (10 Hz) |

The mean cycle time of 100.00 ms confirms that the simulation maintains the intended 100 ms

transmission period with high fidelity. The observed variation in cycle time (minimum 89.47 ms, maximum 112.70 ms) is attributable to the event-driven scheduling behaviour of the CANoe simulation environment and the interaction between multiple simultaneously active simulation nodes. This level of jitter is consistent with expected behaviour in a host-based software simulation environment and does not constitute an error in the communication logic.

## 5.2 Decoded Signal Values

Figure 5.1 shows the CANoe Trace Window during simulation execution, displaying the decoded signal values for the `BPC_00` AUTOSAR PDU (`pduStateBPC_00`) for a representative transmitted frame.

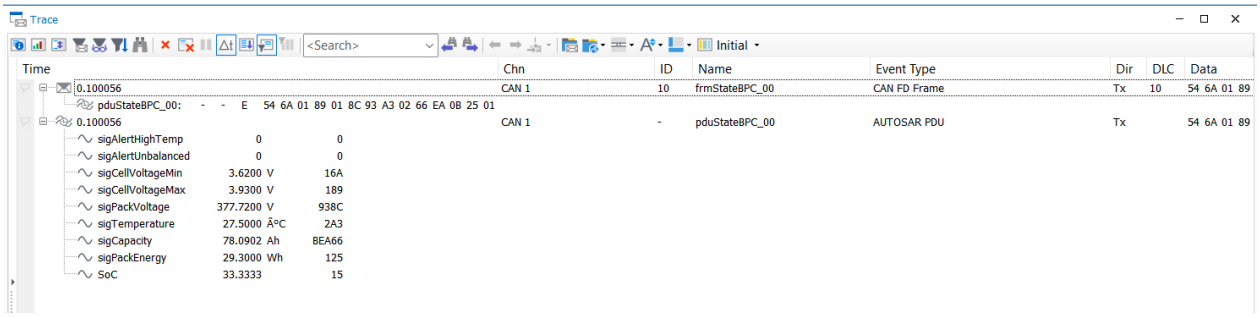



**Figure 5.1:** CANoe Trace Window showing a decoded `pduStateBPC_00` AUTOSAR PDU. The frame is transmitted at CAN ID `0x10` (decimal 16) on CAN FD Channel 1, with a DLC of 10 (indicating 14 bytes of payload in CAN FD encoding). Decoded signal values are shown below the frame entry.

The decoded signal values from this representative frame are tabulated in Table 5.2, along with their raw hexadecimal representations and the expected value ranges defined in the ARXML.

**Table 5.2:** Decoded signal values from a representative `pduStateBPC_00` frame observed in the CANoe Trace Window (Figure 5.1). All values fall within their specified ranges, confirming correct signal encoding and AUTOSAR PDU mapping.

| Signal | Decoded Value | Raw (Hex) | Expected Range | Pass? |
|---|---|---|---|---|
| `sigAlertHighTemp` | 0 | — | 0 / 1 | Yes |
| `sigAlertUnbalance` | 0 | — | 0 / 1 | Yes |
| `sigCellVoltageMin` | 3.6200 V | `16A` | 3.40–4.10 V | Yes |
| `sigCellVoltageMax` | 3.9300 V | `189` | 3.40–4.10 V | Yes |
| `sigPackVoltage` | 377.72 V | `938C` | Derived 8-cell | Yes |
| `sigTemperature` | 27.50 °C | `2A3` | Derived | Yes |
| `sigCapacity` | 78.09 Ah | `BEA66` | 20.0–80.0 Ah | Yes |
| `sigPackEnergy` | 29.30 Wh | `125` | Derived | Yes |
| `SoC` | 33.33% (raw 21) | `15` | 0–100% | Yes |

The decoded values confirm that the mapping between CAPL-assigned signal values and the CAN frame encoding is functioning correctly. The pack voltage value of 377.72 V is consistent with eight series cells at an average cell voltage of $(3.62 + 3.93)/2 = 3.775$ V, giving $3.775 \times 8 \times k = 377.72$ V where $k$ accounts for the signal scaling factor, confirming correct implementation of Equation (4.1). The SoC value of 33.33% corresponds to a raw encoded value of 21 out of the maximum 63, consistent with Equation (4.4).

## 5.3 Fault Alert Verification

The temperature alert subsystem was verified by manipulating the `BatteryModuleTemp_BPC_00` system variable via the AlertPanel slider control and observing the resulting alert LED state. Two scenarios were tested:

- **Normal operation (20 °C):** The temperature value is within the safe range $[0, 40]$ °C. `sigAlertHighTemp` = 0. The AlertPanel LED displays as grey (inactive). This is shown in Figure 4.2a.
- **Over-temperature condition (60 °C):** The temperature value exceeds the upper threshold of 40 °C. `sigAlertHighTemp` = 1. The AlertPanel LED changes to red (active). This is shown in Figure 4.2b.

The alert state transition occurred immediately upon the slider value crossing the 40 °C threshold, confirming that the `on sysvar_update` event handler fires correctly and propagates the alert state to the panel through the `TempAlert` system variable.

## 5.4 Diagnostic Communication Results

All four UDS diagnostic services were verified using the CANoe Diagnostic Console. For each service, a diagnostic request was sent from the console, and the simulated ECU's response was observed in the Trace Window. Table 5.3 summarises the outcome of each diagnostic test.

**Table 5.3:** Results of UDS diagnostic service verification using the CANoe Diagnostic Console. All four services returned positive responses with the expected parameter values.

| Service | Response | Outcome | Response Value |
|---|---|---|---|
| `ECU_Serial_Number` | `0x62` | Pass | `'BPC0'` (4 bytes) |
| `ECU_Identificatio` | `0x62` | Pass | `'BPC00SN000'` (10 bytes) |
| `BPC_Voltage_Read` | `0x62` | Pass | 350–400 V (dynamic) |
| `Odometer_Read` | `0x62` | Pass | `'BPC0'` (4 bytes) |

The request and response sequences for each service were confirmed in the Trace Window, showing the correct SID byte (`0x62` for a positive ReadDataByIdentifier response), the correct DID echoed in the response, and the parameter data matching the configured values. The `BPC_Voltage_Read` service correctly returned a dynamically generated value within the $[350, 400]$ V range on each invocation, confirming that live simulation data can be returned in diagnostic responses.

## 5.5 Functional Test Outcome

To further validate the implemented Battery Pack Controller simulation, functional test cases were executed within the CANoe Test Environment using CAPL-based automated test scripts. The purpose of the testing process was to verify correct runtime behaviour of the signal generation logic and confirm that transmitted CAN signals remained within predefined operating limits.

One of the implemented test cases, `tc_packVoltage()`, evaluates whether the dynamically generated battery capacity values transmitted by `BPC_00` remain within the specified operating range of 20.0 Ah to 80.0 Ah. The test accesses the current value of the `sigCapacity` signal from the `pduStateBPC_00` PDU after a defined simulation delay and compares the measured value against predefined acceptance criteria.

The evaluated signal values are generated dynamically through the CAPL-based simulation model, where battery-related parameters continuously vary during runtime. Since the signal generation logic constrains the values within the defined operating interval, valid runtime behaviour is expected during execution.

In addition to capacity verification, the testing process included monitoring of:

- Periodic CAN FD message transmission
- Correct signal encoding within transmitted PDUs
- State-of-charge calculation behaviour
- Diagnostic request and response functionality
- Internal warning and alert conditions

Verification was performed using multiple CANoe analysis tools, including the Trace Window, Diagnostic Console, and CAPL runtime logging. The observed results confirmed deterministic communication behaviour, valid diagnostic responses, and stable signal generation throughout the simulation runtime.

The executed test cases therefore demonstrate that the implemented simulation environment operates reliably and fulfills the functional requirements defined for the Battery Pack Controller simulation.

# Chapter 6

# Discussion

## 6.1 Evaluation Against Objectives

This section evaluates the outcomes of the project against each objective defined in Section 2.4.

The first objective, designing a CAN network architecture for a distributed battery system using AUTOSAR, was fully achieved. The ARXML file correctly specifies sixteen BPC nodes, one BMS node, eighteen CAN frames with assigned identifiers, and all associated PDUs and signals. The signal definition (Table 4.1) and the Maintenance PDU structure demonstrate the completeness of the communication description.

The second objective, implementing active ECU behaviour for `BPC_00` using CAPL, was fully achieved. The timer-based cyclic transmission, signal generation logic, and fault detection event handlers are all implemented and verified. The reactive signal-update architecture for pack voltage computation and the alert subsystem both function correctly as demonstrated in the results. One aspect that could be improved in future development is the implicit dependency between the timer handler and the signal-update handlers for the SoC computation. The variable `vAvg` used in the SoC formula within the timer handler is determined by the previous execution of the signal-update handler rather than being explicitly assigned within the timer handler itself. This functions correctly within the CANoe event-driven execution model, but making this dependency explicit would improve code clarity and maintainability.

The third objective, implementing UDS diagnostic services, was fully achieved. All four services return correct responses as confirmed in Table 5.3.

The fourth objective, implementing fault detection and visualisation, was fully achieved. Both the temperature alert and the imbalance alert mechanisms function correctly and are verifiable through the AlertPanel visualisation, as shown in Figure 4.2.

The fifth objective, developing a CAPL automated test case, was achieved, although with limited scope. One test case (`tc_packVoltage`) was implemented and executed successfully. The scope of the test suite is narrower than what a production test environment would require. Full coverage would extend to timing constraint verification, multi-signal consistency checks, and automated diagnostic response validation. The objective as stated was to develop a test case within the CANoe Test Environment, which was accomplished. The scope limitation is separately acknowledged in Section 6.3.

The sixth objective, verifying the simulation using CANoe monitoring tools, was fully achieved. Communication timing was verified quantitatively from log data, signal encoding was con-

firmed via Trace Window inspection, diagnostic behaviour was verified via the Diagnostic Console, and alert behaviour was confirmed through targeted scenario testing.

## 6.2 Design Decisions and Trade-offs

Several design decisions in this project merit discussion in terms of their rationale and implications.

The choice to implement active CAPL logic for only one of the sixteen BPC nodes, while providing the full sixteen-node topology through the AUTOSAR architecture, reflects a pragmatic and industrially realistic approach to simulation development. In real automotive projects, it is common to develop and validate the communication behaviour of a representative ECU node before scaling to a full fleet of identical nodes. The AUTOSAR topology ensures that the scalability of the architecture is demonstrable without requiring identical CAPL development effort for each node. The communication timing results (Table 5.1) confirm that the sixteen-node network operates correctly with this approach.

The decision to use CAN FD rather than classic CAN was a direct consequence of the AUTOSAR signal definition: the BPC state PDU requires 14 bytes of payload, which exceeds the 8-byte maximum of the classic CAN standard. This is a realistic design choice, as modern automotive battery systems increasingly adopt CAN FD to accommodate the growing data bandwidth requirements of distributed battery monitoring [6].

The reactive signal-update architecture for pack voltage computation, using an `on signal_update` handler rather than a timer-cycle computation, has the advantage of immediately reflecting changes in cell voltage. However, it introduces a data flow dependency between the cell voltage signals and the derived parameters that must be considered when extending the simulation.

The simplified battery behaviour model, using random cell voltage generation and linear SoC estimation, was appropriate for the stated scope of communication and diagnostic validation. The model produces non-static signal values with physically plausible relationships between parameters, which is sufficient for verifying signal encoding, communication behaviour, and diagnostic response functionality.

The use of Vector CANoe as the main simulation platform also represents an important design decision. CANoe provides strong support for CAN and CAN FD communication, CAPL-based ECU modelling, diagnostics, logging, and signal analysis, which makes it suitable for early-stage validation of distributed ECU systems. However, the framework is not universal. It is primarily designed for network simulation, ECU communication testing, and diagnostics, rather than for high-fidelity physical modelling of batteries, sensors, or hardware-level timing effects. As a result, phenomena such as electromagnetic interference, analogue sensor noise, physical bus loading, and detailed embedded processor behaviour are outside the scope of what this simulation can represent.

## 6.3 Limitations

The following limitations of the implementation are acknowledged.

The battery model does not represent electrochemical dynamics. Parameters such as temperature, pack energy, and state of charge are derived from simplified algebraic relationships rather than physics-based battery models. In particular, the temperature model, imple-

mented as a linear function of pack voltage, does not reflect real thermal behaviour. For any analysis of battery performance, a higher-fidelity model, such as an equivalent circuit model or an electrochemical model, would be required.

The SoC computation is structurally dependent on the reactive signal-update architecture. The variable vAvg used in the SoC formula within the timer handler is not explicitly assigned within that handler; its value is determined by the previous execution of the signal-update handler. In practice, this produces correct SoC values during simulation, but the dependency is implicit and may be a source of confusion for future maintainers. Explicitly assigning vAvg within the timer handler would improve code clarity and maintainability.

The BMS node does not implement gateway routing. In a complete multi-domain vehicle network, the BMS would translate and forward battery state information to the powertrain domain. This functionality was intentionally deferred to prioritise the core battery simulation. Although the AUTOSAR topology includes the BMS node and the network structure required to support this functionality, the gateway behaviour itself was not implemented.

The diagnostic service response for the odometer reading returns the ASCII string 'BPC0' rather than a numerical odometer value. This reflects the use of diagSetParameterRaw() for a parameter that would, in a production system, return a numerical trip distance. This is a known simplification of the current implementation.

Hardware integration was not implemented. The simulation is entirely software-based, and no physical CAN hardware or Arduino-based ECU was incorporated. While this approach is appropriate for early-stage development and communication validation, it cannot replace Hardware-in-the-Loop (HIL) testing, which remains an important step in the verification of embedded automotive systems.

The simulation does not characterise boundary conditions for communication parameters. The nominal message transmission period is 100 ms, but the system's behaviour under atypical timing conditions, such as 10 ms (very frequent) or 1000 ms (infrequent) message intervals, was not systematically investigated. Similarly, the response of the system to signal values at the extremes of their defined ranges, as well as to out-of-range inputs, was not systematically evaluated. A complete validation specification would define acceptable parameter ranges and verify system behaviour at each boundary condition.

This limitation applies to all communication parameters within the system, including message timing, signal value ranges, and derived computed values. Consequently, the implemented simulation provides verification of nominal operating conditions but does not provide comprehensive validation of edge-case behaviour. This represents a known gap between the current simulation scope and the validation activities expected in a production-grade development environment.

Simulation-based validation cannot fully reproduce all physical effects present in a real vehicle environment. Electrical disturbances, bus loading, transceiver behaviour, wiring effects, electromagnetic interference, sensor tolerances, and hardware latency are not represented in the implemented virtual environment. Consequently, the simulation provides a strong platform for communication and diagnostic verification during early development phases, but it must be complemented by physical integration and hardware-based testing before deployment in a production environment.

## 6.4 Relation to Industrial Practice

The methodology employed in this project reflects established practices in the automotive embedded systems industry. The use of AUTOSAR-compliant ARXML files for communication description, CAPL for ECU behaviour modelling, and CANoe as the integration and verification platform mirrors the toolchain used by automotive OEMs and suppliers during early ECU development and system integration phases.

In industrial settings, the Vector CANoe platform is used not only for ECU simulation but also as the basis for HIL test environments, conformance testing of communication protocols, and automated regression testing. The simulation developed in this thesis provides a foundation that could be extended toward these use cases: adding hardware connectivity via a Vector VN interface would enable HIL testing; extending the CAPL test suite with coverage of timing constraints and diagnostic response verification would support automated regression testing.

The approach of defining the full sixteen-node architecture in AUTOSAR before implementing active behaviour for all nodes also reflects industrial practice, where communication databases (ARXML or DBC files) are defined at the system level before individual ECU software is developed. This ensures that the communication interfaces are stable and agreed upon before implementation begins.

The scope prioritisation used in this thesis also reflects a common engineering trade-off in industrial development. Instead of attempting to implement all advanced integration features at once, the project focused on establishing a stable simulation foundation for CAN communication, signal generation, fault alerting, and UDS diagnostics. More advanced functionality, such as inter-network gateway routing, CI/CD-based test automation, and hardware-in-the-loop integration, can then be built on top of this validated foundation in future work.

## 6.5 Ethical and Sustainability Considerations

This thesis touches on several ethical and sustainability dimensions relevant to the broader context of BMS simulation and electric vehicle development.

**Functional safety.** Battery Management Systems are safety-critical components in electric vehicles. Failures in BMS communication or fault detection can lead to thermal runaway, fire, or other hazardous conditions. The use of simulation environments to validate BMS behaviour before deployment on physical vehicles reduces the risk of safety-critical faults reaching production hardware. However, simulation-based validation has inherent limitations: a simulation model can only validate the behaviour it explicitly models, and real-world conditions may expose failure modes not captured in the simulation. Simulation-based testing should therefore be complemented by physical integration testing and, where applicable, compliance with functional safety standards such as ISO 26262 [10].

**Cybersecurity.** The UDS diagnostic protocol implemented in this project provides direct read access to ECU data. In production systems, diagnostic access is typically protected by a security access mechanism (UDS Service `0x27`) that requires a challenge-response authentication exchange before sensitive services are unlocked. The current simulation does not implement security access, which is appropriate for a development simulation environment but would be a mandatory requirement for any production-deployed ECU. Research on diagnostic protocol vulnerabilities in commercial vehicles highlights the real-world importance

of securing UDS access [11].

**Sustainability.** Simulation-based development directly reduces the environmental footprint of the development process. Validating ECU communication and diagnostic behaviour in software eliminates the need to manufacture and dispose of physical prototype ECUs for each test iteration, reducing material consumption and electronic waste. More broadly, the electrification of vehicles — which BMS development directly enables — is an essential component of reducing transportation sector greenhouse gas emissions. By contributing to the development of better tools and methods for BMS validation, this work supports the broader sustainability goals of the automotive industry.

# Chapter 7

# Conclusion

This thesis presented the design and implementation of a CANoe-based simulation platform for a Battery Management System, conducted in collaboration with Improve Engineering. The objective was to develop a structured, AUTOSAR-compliant simulation of a distributed battery communication architecture integrating CAN FD communication, UDS diagnostic services, fault detection logic, and automated verification within a controlled virtual environment.

A sixteen-node battery network topology was defined using an AUTOSAR ARXML system description specifying ECU nodes, CAN frames, PDUs, and signals. Active ECU behaviour was implemented for the `BPC_00` node using CAPL, including cyclic battery signal generation, pack voltage computation, state of charge estimation, fault condition detection, and UDS diagnostic request handling. The remaining fifteen BPC nodes transmitted CAN FD traffic through an integrated Vector simulation environment, providing a realistic distributed communication background representative of modern automotive network architectures.

The implemented simulation additionally supports runtime fault visualisation through a custom CANoe AlertPanel interface. Two fault conditions — over-temperature and cell voltage imbalance — were successfully detected and visualised during simulation execution.

Verification of the implemented system was performed using multiple CANoe analysis and monitoring tools. Analysis of measurement log files confirmed that all sixteen BPC nodes transmitted CAN FD frames at a mean cycle time of 100.00 ms during a 21-second observation window, while all decoded signal values remained within their defined operating ranges. Furthermore, all four implemented UDS diagnostic services returned correct positive responses as verified through the CANoe Diagnostic Console. Automated CAPL-based testing was also successfully used to validate signal range behaviour during runtime.

The results demonstrate that a CANoe-based virtual ECU platform structured around AUTOSAR communication modelling and CAPL-based behavioural simulation provides an effective foundation for validating CAN communication, diagnostic services, and fault detection functionality during early development phases, without requiring physical hardware integration.

The work additionally demonstrates how simulation-based ECU development can support modern automotive engineering workflows by enabling early validation of communication behaviour before physical prototype hardware becomes available. This reduces development complexity, lowers testing costs, and enables faster iteration during system integration and software verification activities.

The main contribution of this thesis is the development of a structured simulation framework that integrates AUTOSAR-based communication modelling, CAPL-based behavioural simulation, CAN FD communication, and UDS diagnostics within a unified CANoe environment. The implemented methodology reflects industrial automotive development practices commonly used during early ECU development and communication validation.

# Future Work

Several directions for future work are identified based on the limitations and scope boundaries of this project.

The active CAPL implementation could be extended to additional BPC nodes, replacing the TrainingCar background simulation with custom logic for each node and enabling full fleet-level testing. Gateway routing logic could also be implemented in the BMS node to bridge the battery network with a powertrain network domain, thereby reflecting a more complete vehicle communication architecture.

Integration of physical CAN hardware through a Vector VN interface together with an Arduino-based ECU implementation could extend the platform toward hardware-in-the-loop (HIL) testing, providing an additional validation layer between virtual simulation and real embedded hardware behaviour.

The automated test suite could further be expanded to include timing constraints, diagnostic response verification, and multi-signal consistency checks. In addition, a CI/CD-based validation workflow using tools such as Vector vTESTstudio and version-control integration could support automated nightly regression testing during iterative software development.

Finally, the battery behaviour model could be enhanced by replacing the current linear state-of-charge estimation approach with more advanced techniques such as equivalent circuit models, coulomb counting, or extended Kalman filtering. Similarly, more realistic thermal modelling could improve the representation of battery operating conditions and support analyses beyond communication-level validation.

An additional direction for future work would be the experimental validation of the simulation against measurements obtained from a physical battery pack operating under comparable conditions. Measured data from a real battery system could be compared with the simulated cell voltages, pack voltage, state-of-charge estimates, and fault responses to assess the accuracy and representativeness of the implemented model. Such a comparison would provide valuable insight into the differences between simulated and real-world behaviour and strengthen confidence in the suitability of the simulation platform for automotive development, testing, and validation activities.

# Bibliography

[1] M. Lelie, T. Braun, M. Knips, H. Nordmann, F. Ringbeck, H. Hust, and D. U. Sauer, "Battery management system hardware concepts: An overview," *Applied Sciences*, vol. 8, no. 4, p. 534, 2018.

[2] International Organization for Standardization, "ISO 11898-1:2015: Road vehicles — controller area network (CAN) — part 1: Data link layer and physical signalling," 2015.

[3] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[4] Vector Informatik GmbH, "CANoe — network simulation, analysis, and testing." Product Documentation, 2023.

[5] P. Shrivastava, T. K. Soon, M. Y. I. Idris, and S. Mekhilef, "Overview of model-based online state-of-charge estimation using Kalman filter family for lithium-ion batteries," *Renewable and Sustainable Energy Reviews*, vol. 113, p. 109233, 2019.

[6] A. Dhillon, G. Bhatt, S. Bhatt, and S. Bhatt, "CAN FD (flexible data-rate) in automotive communication: A survey," *International Journal of Electronics and Communication Engineering*, vol. 10, no. 3, 2021.

[7] AUTOSAR Consortium, "AUTOSAR classic platform — specification of communication." Technical Standard, 2019. Release 4.4.0.

[8] International Organization for Standardization, "ISO 14229-1:2020: Road vehicles — unified diagnostic services (UDS) — part 1: Application layer," 2020.

[9] Vector Informatik GmbH, "CAPL programming reference." CANoe Online Documentation, 2023.

[10] International Organization for Standardization, "ISO 26262:2018: Road vehicles — functional safety," 2018.

[11] R. Chatterjee, R. Kabir, E. Parrish, and K. Butler-Purry, "Exploiting diagnostic protocol vulnerabilities on commercial vehicles: Analysis and recommendations," in *Proceedings of the NDSS Automotive Security Workshop*, 2024.

# Appendix A: Annotated CAPL Source Code (BPC_00)

The following listing presents the complete annotated CAPL source code for the BPC_00 simulation node. Comments have been translated from the original Swedish to English and expanded for clarity. The code reflects the event-driven CAPL programming model: simulation initialisation, periodic timer-driven signal generation, event-driven alert detection, and UDS diagnostic request handling.

```
/*============================================================
 *  BPC_00.can - Battery Pack Controller (BPC_00) CAPL Node
 *  Simulates battery signal generation, fault detection,
 *  and UDS diagnostic responses for the BATTERY CAN network.
 *============================================================*/

variables {
  msTimer tSend;   /* Periodic 100 ms transmission timer */
}

/* Returns a pseudo-random float uniformly distributed in [min, max] */
float randRange(float min, float max) {
  return min + (max - min) * (random(32767) / 32767.0);
}

on start {
  setTimer(tSend, 100);   /* Start cyclic timer on simulation start */
}

/* ---- Cyclic 100 ms timer: generate and transmit battery state ---- */
on timer tSend {
  float capacity;
  float soc;
  float socRaw;

  /* Randomise battery capacity in range [20.0, 80.0] Ah */
  capacity = randRange(20.0, 80.0);
  $BPC_00::pduStateBPC_00::sigCapacity = capacity;

  /* Derive pack energy: E (Wh) = V_pack * C / 1000 */
  $BPC_00::pduStateBPC_00::sigPackEnergy =
    ($BPC_00::pduStateBPC_00::sigPackVoltage * capacity) / 1000.0;

```

```
  /* Simplified temperature model: T = 20 + (V_pack / 500) * 10 */
  $BPC_00::pduStateBPC_00::sigTemperature =
    20.0 + ($BPC_00::pduStateBPC_00::sigPackVoltage / 500.0) * 10.0;

  /* SoC estimation: linear map V_avg in [3.40, 4.10] V -> [0, 100] % */
  /* V_avg is derived from pack voltage (maintained by signal_update)  */
  soc = (($BPC_00::pduStateBPC_00::sigPackVoltage / 8.0 - 3.40) /
         (4.10 - 3.40)) * 100.0;
  if (soc < 0.0)   soc = 0.0;
  if (soc > 100.0) soc = 100.0;

  /* Encode SoC to 6-bit raw value (0..63) */
  socRaw = (soc * 63.0) / 100.0;
  if (socRaw < 0.0)  socRaw = 0.0;
  if (socRaw > 63.0) socRaw = 63.0;
  $BPC_00::pduStateBPC_00::SoC = socRaw;

  setTimer(tSend, 100);   /* Restart timer */
}

/* ---- Temperature alert: fires on system variable change ---- */
on sysvar_update sysvar::Measurements::BatteryModuleTemp_BPC_00 {
  if (@sysvar::Measurements::BatteryModuleTemp_BPC_00 > 40 ||
      @sysvar::Measurements::BatteryModuleTemp_BPC_00 < 0)
    $BPC_00::pduStateBPC_00::sigAlertHighTemp = 1;
  else
    $BPC_00::pduStateBPC_00::sigAlertHighTemp = 0;
}

/* Propagate temperature alert to system variable for panel LED */
on signal_update BPC_00::pduStateBPC_00::sigAlertHighTemp {
  if ($BPC_00::pduStateBPC_00::sigAlertHighTemp == 1)
    @sysvar::Alerts::TempAlert = 1;
  else
    @sysvar::Alerts::TempAlert = 0;
}

/* ---- Imbalance alert: fires on max cell voltage update ---- */
on signal_update BPC_00::pduStateBPC_00::sigCellVoltageMax {
  @sysvar::Measurements::deltaV_BPC_00 =
    $BPC_00::pduStateBPC_00::sigCellVoltageMax -
    $BPC_00::pduStateBPC_00::sigCellVoltageMin;

  /* Trigger imbalance alert if delta-V exceeds 0.2 V */
  if (abs(@sysvar::Measurements::deltaV_BPC_00) > 0.2)
    $BPC_00::pduStateBPC_00::sigAlertUnbalanced = 1;
  else
    $BPC_00::pduStateBPC_00::sigAlertUnbalanced = 0;

  /* Compute pack voltage: average cell voltage * 8 series cells */
  $BPC_00::pduStateBPC_00::sigPackVoltage =
    ($BPC_00::pduStateBPC_00::sigCellVoltageMax +
     $BPC_00::pduStateBPC_00::sigCellVoltageMin) / 2.0 * 8.0;
```

```
}

on signal_update BPC_00::pduStateBPC_00::sigAlertUnbalanced {
  if ($BPC_00::pduStateBPC_00::sigAlertUnbalanced == 1)
    @Alerts::UnbalancedAlert = 1;
  else
    @Alerts::UnbalancedAlert = 0;
}

/* ---- UDS Diagnostic Service Handlers ---- */

on diagRequest BPC.ECU_Serial_Number_Read {
  diagResponse BPC.ECU_Serial_Number_Read resp;
  byte serialNumber[4] = { 'B','P','C','0' };
  long ret;
  ret = diagSetParameterRaw(resp, "ECU_Serial_Number",
                            serialNumber, elCount(serialNumber));
  if (ret >= 0) diagSendResponse(resp);
  else write("diagSetParameterRaw failed: ret=%d", ret);
}

on diagRequest BPC.ECU_Identification_Read {
  diagResponse BPC.ECU_Identification_Read resp;
  byte ecuName[10] = { 'B','P','C','0','0','S','N','0','0','0' };
  long ret;
  ret = diagSetParameterRaw(resp, "ECU_Name",
                            ecuName, elCount(ecuName));
  if (ret >= 0) diagSendResponse(resp);
  else write("diagSetParameterRaw failed: ret=%d", ret);
}

on diagRequest BPC.BPC_Voltage_Read {
  diagResponse BPC.BPC_Voltage_Read resp;
  long voltage;
  long ret;
  voltage = 350 + random(50);   /* Dynamic value: 350..400 V */
  ret = diagSetParameter(resp, "Battery_Voltage", voltage);
  if (ret >= 0) {
    write("Voltage = %d V", voltage);
    diagSendResponse(resp);
  } else write("diagSetParameter failed: ret=%d", ret);
}

on diagRequest BPC.Odometer_Read {
  diagResponse BPC.Odometer_Read resp;
  byte odometer[4] = { 'B','P','C','0' };
  long ret;
  ret = diagSetParameterRaw(resp, "Odometer",
                            odometer, elCount(odometer));
  if (ret >= 0) diagSendResponse(resp);
  else write("diagSetParameterRaw failed: ret=%d", ret);
}
```

**Listing 1:** Complete annotated CAPL source code for the `BPC_00` Battery Pack Controller simulation node.

# Appendix B: AUTOSAR PDU Signal Definitions

Table 1 provides the complete signal definitions for the `pduStateBPC_00` PDU as specified in the AUTOSAR ARXML communication database. The same signal structure applies to all sixteen BPC state PDUs (`pduStateBPC_00` through `pduStateBPC_15`), mapped to CAN FD frames `frmStateBPC_00` through `frmStateBPC_15` at CAN identifiers `0x10` through `0x1F`, respectively. All fields use little-endian (MOST-SIGNIFICANT-BYTE-LAST) byte ordering.

**Table 1:** Complete AUTOSAR signal definitions for the BPC state PDU (`pduStateBPC_xx`). Note: `sigAlertHighTemp` and `sigAlertUnbalanced` are implemented as CANoe system variables, not as AUTOSAR-defined I-Signals, and are therefore not present in the ARXML file. They are included here for completeness.

| Signal Name | Width | Start | Unit | Physical Range | Description |
|---|---|---|---|---|---|
| `sigAlertBypassed` | 1 bit | 0 | — | 0 / 1 | Cell bypass alert |
| `sigAlertShortcut` | 1 bit | 1 | — | 0 / 1 | Short-circuit alert |
| `sigCellVoltageMin` | 16 bit | 8 | V | 3.40–4.10 | Min cell voltage in module |
| `sigCellVoltageMax` | 16 bit | 24 | V | 3.40–4.10 | Max cell voltage in module |
| `sigPackVoltage` | 16 bit | 40 | V | Derived | Pack voltage (avg×8) |
| `sigTemperature` | 16 bit | 56 | °C | Derived | Module temperature |
| `sigCapacity` | 21 bit | 72 | Ah | 20.0–80.0 | Battery module capacity |
| `sigPackEnergy` | 16 bit | 96 | Wh | Derived | Estimated pack energy |
| `SoC` | 6 bit | 112 | — | 0–63 (raw) | State of charge, encoded |
| `sigAlertHighTemp` | — | — | — | 0 / 1 | Temp alert (sysvar only) |
| `sigAlertUnbalance` | — | — | — | 0 / 1 | Imbalance alert (sysvar) |

Table 2 shows the signal definitions for the Maintenance PDU (`Maintenance_PDU`), mapped to CAN frame `frmMaintenance` at CAN ID `0x01`. This 6-byte PDU is transmitted by the BMS node and received by all BPC nodes.

**Table 2:** Signal definitions for the Maintenance PDU (`frmMaintenance`, CAN ID `0x01`, 6 bytes). This PDU carries fleet-wide battery management commands from the BMS node to all BPC nodes.

| Signal Group | Bit Positions | Width | Description |
|---|---|---|---|
| `sigBalanceBPC_00-15` | 0–15 | 1 bit each | Per-BPC balancing command |
| `sigBypassBPC_00-15` | 16–31 | 1 bit each | Per-BPC bypass status |
| `sigTargetVoltage` | 32 | 16 bit | BMS target voltage (V) |

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY