HORIZON EUROPE PROGRAMME
TOPIC HORIZON-CL5-2023-D2-05-01

GA No. 101137975

Situationally Aware Innovative Battery Management System for Next Generation Vehicles

InnoBMS - Deliverable report

D3.1 - Embedded edge software for BMS

Funded by the European Union

Deliverable No.	D3.1	
Related WP	WP3	
Deliverable Title	Embedded edge software for BMS	
Deliverable Date	2026-04-30	
Deliverable Type	REPORT	
Dissemination level	Public (PU)	
Author(s)	Leonardo Chiacchiararelli (FMF) Plugaru Tudor Stefan (BOSCH) Nagy Sandor (BOSCH) Halauca Ionut Emilian (BOSCH) Mireia Perna Vila (IDIADA) Tomaž Katrašnik (UL) Igor Mele (UL) Bernhard Stanje (AVL) Pankaj Ganeshrao Pallewar (AVL) Jon Perez (CIDETEC) Joanes Lorente (CIDETEC) Robert Alfie Peña (VUB) Sajib Chakraborty (VUB)	2026/08/07
Checked by	Deborah Lyall (FMF)	2026/08/11
Reviewed by	Markus Faltermeier (AVL-SFR)	2026/08/11
	Guta Paul (BOSCH)	2026/08/11
Coordinator	Prof. dr. ir. Omar Hegazy (VUB)	2026/04/24

Document History

Version	Date	Editing done by	Remarks
V1.0	2026/04/15	L. Chiacchiararelli, A. Foster, D.Lyall	
V2.0 FINAL	2026/08/11	L. Chiacchiararelli, A. Foster, S. Chakraborty, G. Paul	Submitted

Project summary

The core objective of InnoBMS is to develop and demonstrate (TRL6) a future-ready best-in-class BMS hard- and software solution that maximizes battery utilization and performance for the user without negatively affecting battery life, even in extreme conditions, whilst continuously maintaining safety. Concretely, the InnoBMS proposal will deliver a 12% higher effective battery pack volumetric density, a 33% longer battery lifetime and a demonstrated lifetime of 15 years. The results will be demonstrated using novel testing methods that give a 36% reduction in the testing time of a BMS. The results will be demonstrated in two use cases, one light commercial vehicle (Fiat Doblo Electric) and one medium-duty van (IVECO eDaily). The key outcomes will enable a cost reduction of 12% and 9.7% for passenger cars and light-duty vehicles, respectively. The core objective will be achieved through five technical objectives. 1) advanced hybrid physical and data-driven models and algorithms to enable a flexible and modular BMS suitable for a wide range of batteries. 2) Software for a fully connected and fully wireless BMS that acts as a communication server inside the vehicle E/E-architecture, the center of connection, on-board diagnostics and decision-taking for all battery-related information. 3) A scalable, fully wireless and self-tested BMS hardware that enables using different battery sizes at different operating voltage levels, and smart sensor integration. 4) Better battery utilization and exploitation using cloud-informed strategies and procedures. 5) A heterogeneous simulation toolchain and automated test methods.

Publishable summary

A streamlined harmonization approach is established to create a scalable and modular software architecture capable of integrating the advanced InnoBMS algorithms developed by multiple project partners in T2.2 into a common execution framework within T3.2. This includes the definition of software interfaces, communication mechanisms, deployment concepts, and integration guidelines required to deploy advanced battery functions on the Edge ECU platform.

This deliverable covers two complementary software streams: (a) advanced functionality software for the embedded Edge system, and (b) core software for the sub-Battery Management System (sBMS).

(a) Advanced Functionality Software for the Embedded Edge System

This software stream addresses the deployment of the advanced functionalities developed in T2.2 on the Edge ECU platform. The functionalities are compiled and prepared for deployment on the InnoBMS Edge ECU, based on the dSPACE MicroAutoBox III. Each functionality is structured into an edge-executable format, including its interfaces and data definitions, programming language, computational requirements, and integration complexity. With support from VUB and BOSCH, functionality developers generate edge-executable software, including Functional Mock-up Units (FMUs) compliant with FMI 2.0/3.0, using tools such as the Simulink Compiler FMU Builder. Python-based functionalities are packaged in an FMI 2.0-compliant format, while C++ implementations are similarly integrated as FMI-compliant units, enabling flexible deployment and interoperability on the InnoBMS Edge ECU.

(b) sBMS Core Software (PTE, FMF)

The sub-Battery Management System (sBMS) provides the core battery-management functionality and interfaces with the other system units, including the Edge ECU and Vehicle Control Unit (VCU). Its main functions include managing battery sensor data, communicating cell-level information, and supporting the safety and cybersecurity of the battery system.

The sBMS directly controls the HV contactors to ensure safe battery-pack operation. CAN communication with the safety-certified Cell Monitoring Controllers (CMCs) provided by AVL enables real-time monitoring of cell parameters, including voltage and temperature. The sBMS hardware and software are developed following an ISO 26262-compliant development process, supporting the required functional-safety objectives of the system.

Overall, the software development activities and timelines presented in this deliverable are aligned with the planned availability of the hardware components, ensuring coordinated and timely system integration.

Contents

List of Figures

List of Tables

Abbreviations & Definitions

Abbreviation	Explanation
AVL-SFR	AVL Software and Functions
BCU	Battery Control Unit
BMS	Battery Management System
CAN	Control Area Network
CMB	Cell Monitoring Board
CMC	Cell Monitoring Controllers
CySec	Cyber Security
DBC	Data Base CAN
Edge ECU	Edge Electronic Control Unit
FMF	FPT Motorenforschung AG
Gen5m	Production-ready BMS product by Potenza, used as baseline for InnoBMS
Li-P	Li-Plating
LLSW	Low Level Software
PTE	Potenza Technology
SBMS	Sub-Pack Battery Management System
TRA	Thermal Run-Away
VCU	Vehicle Control Unit
vHIL	Virtual Hardware-In-the-Loop
WNM	Wireless Network Manager

1 Introduction

This deliverable report lays out the scope of the software requisites to design embedded software for the project's Battery Management System (BMS). This deliverable comprises two software components: (a) Edge SW: advanced functionalities software for the embedded edge system, which is referred to from hereon as Edge SW, and (b) sBMS SW: software for the sub–Battery Management System (sBMS), referred to as core BMS SW. Both are shown in figure 1-1 below.

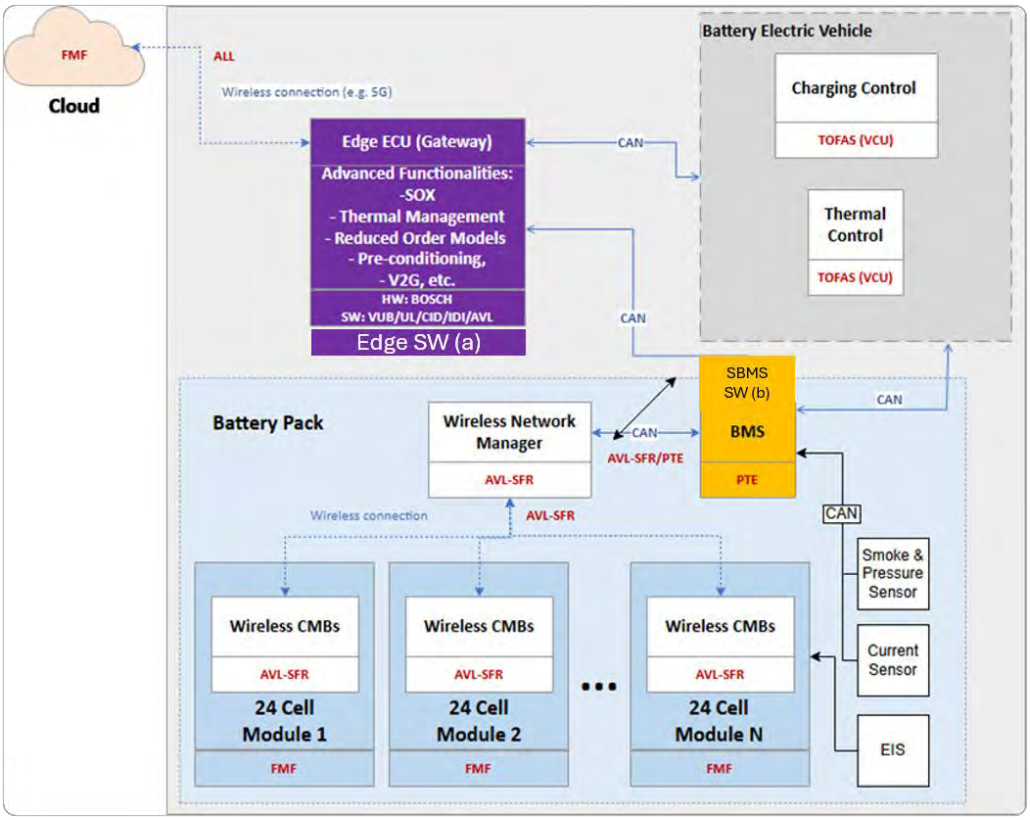

Figure 1-1. InnoBMS: embedded SW streams.

Both embedded software have been created to monitor, manage and protect battery systems on two different architectures: 400V and 800V. The software will monitor the battery behaviour, performance and health.

The target of this task is the design of the sBMS software including safety layer, drivers and diagnostics following ISO 26262 (Functional Safety) and ISO 21434 (Cyber Security) starting with the requirements set out at the beginning of the project. FMF provided the production-ready Gen5m sBMS as the project baseline, as discussed in section 3. The Gen5m, certified to ISO 26262 ASIL-C and in production since May 2024, is integrated with FMF Cell Monitoring Controllers (CMCs) to form the V1 hardware platform. In parallel, AVL-SFR developed dedicated Cell Monitoring Boards (CMBs) for the project, incorporating the Texas Instruments Wireless Network Manager (WNM) with support from PTE. The primary hardware adaptations focus on enabling communication between the sBMS and the wireless battery monitoring system.

The next step is to consider the work led by AVL on the Advanced Functionalities for the Edge device from WP2. The BMS communicates with the models in the Edge device for a bidirectional communication of data for better battery performance and diagnostics. Following on from the creation of these algorithms, the testing phase will take place in Task 3.2 to integrate these into the Edge device and see a seamless communication stream between the hardware devices and InnoBMS cloud.

The output from this task will be useful in the integration of advanced functionalities in the Edge-Electronics Control Unit (ECU) for the 400V/800V architecture in Task 3.4 and subsequently into Work Package 5, on the virtual and physical testing of the complete InnoBMS solution.

All the advanced functionalities developers contributed through the development of edge-executable SW models, including FMU-based implementations compliant with FMI standards. These contributions span physics-based models, data-driven algorithms, and control strategies, all aligned with the computational and integration constraints of the InnoBMS edge platform. The coordinated effort ensures interoperability and efficient deployment across the embedded system.

- **UL:** Generated FMUs from C++-based reduced-order physics models and provided RC-model parameters for SoX estimation;
- **IDI:** Though IDI is not part of WP3, IDI still supported the FMU generation for the SoX model in an edge-executable format;
- **AVL:** Provided data-driven algorithms for BMS edge devices in FMU format;
- **CID:** Generated FMUs from MATLAB/Simulink algorithms for embedded edge BMS functionalities, including (a) thermal runaway detection (TRA), (b) lithium plating detection, and (c) BTMS control;
- **VUB:** Provided FMU blocks for (a) battery pre-conditioning and (b) smart fast-charging functionalities on the embedded edge BMS; VUB supported the preparation and development of the advanced functionalities FMUs through testing and feedback sessions with individual partners where initial versions of their FMUs were tested on the Edge device and modified until the final versions were validated; VUB also supported the development and integration of the Edge SW, in particular the development of the Raspberry Pi as a gateway and memory device between the Edge and the cloud and the corresponding communication links;
- **BOSCH:** BOSCH contributes to the development and integration of the Edge Software platform that enables the deployment of advanced battery management functionalities in automotive environments. This activity was not initially covered in the project proposal and represents a deviation. The need for this work emerged during project execution, driven by the requirement to provide a common software framework for integrating and deploying advanced battery management functionalities developed by multiple project partners.
  BOSCH activities focus on establishing a scalable and modular software architecture capable of integrating battery algorithms developed by multiple project partners into a common execution framework. This includes the definition of software interfaces, communication mechanisms, deployment concepts, and integration guidelines required for the execution of advanced functions on the Edge ECU platform. A key objective of BOSCH's work is to ensure interoperability between the various functional components developed throughout the project. To achieve this, BOSCH leads the definition and harmonization of software interfaces, coordinates integration activities across partners, and develops the communication infrastructure required for data exchange between the Edge ECU, the Vehicle Control Unit (VCU), and the Battery Management System (BMS). Furthermore, BOSCH supports the deployment and validation of Functional Mock-up Units (FMUs), enabling the integration of diverse battery algorithms within a standardized execution environment.
- **AVL-SFR & THIL:** Monitor and ensure seamless integration and transition of edge software functions within CMBs;

The integration activities provided the backbone of the InnoBMS Edge platform, ensuring that advanced battery management functionalities can be efficiently deployed, evaluated, and operated on the embedded automotive hardware while supporting the project's objective of developing next-generation intelligent battery management solutions.

2 SW Functionalities for Edge ECU

2.1 Edge Hardware selection

The edge hardware selection was carried out by VUB to identify a suitable hardware platform for hosting the advanced functionalities of the InnoBMS project. The selected edge hardware is intended to support local processing, data acquisition from the sBMS, data compression, data buffering, and cloud communication.

The edge device is expected to interface with the vehicle communication network and the sBMS system, support the required software environment for deploying advanced InnoBMS functionalities, and provide sufficient local computational capability for on-board processing. In addition, the platform should enable reliable communication with the cloud while ensuring compatibility with the project's data acquisition, monitoring, and digital-twin-oriented requirements. The main requirements were defined based on one-to-one discussions with the developers of the advanced functionalities and the project OEMs. These requirements are listed below:

- Support for the required software package and MATLAB/Simulink-based development environment, and compatibility with C/C++ and/or Python-based implementation.
- Capability to host and execute multiple advanced functionalities, including SOX estimation, thermal management, reduced-order models, preconditioning, and V2G-related functions.
- Support for CAN 2.0 and/or CAN FD communication.
- Possibility to connect with wireless modules and gateway interfaces.
- Support for cloud communication through suitable communication protocols, such as MQTT.
- Availability of GSM, Wi-Fi, Ethernet, or external modem connectivity for cloud data transfer.
- Sufficient local computational resources for edge-level processing and data compression.
- Local data buffering capability in case of temporary communication loss.
- Possibility to store buffered data locally as snapshots instead of continuous intermediate 1 Hz signal transmission.
- Compliance with the required electrical specifications for vehicle or test-bench operation.
- Feasibility of authentication and certification in case GSM-based communication is used.

2.1.1 Hardware options comparison

Two edge hardware options were evaluated for hosting the advanced InnoBMS functionalities: Flea4+ CarMedia and MicroAutoBox III/II from dSPACE. The comparison focused on software compatibility, CAN communication capability, cloud connectivity, local data buffering, model integration effort, debugging capability, and cost.

The Flea4+ CarMedia platform supports C/C++ and Python-based software development through its SDK and CarMedia API. Although root access is not available, compiled binaries can be deployed through the provided software interface. The platform operates on Linux and can be configured to support either CAN 2.0 or CAN FD. It also provides GSM and Wi-Fi connectivity, making it suitable for direct cloud communication (but only 1-way, e.g., only vehicle-cloud). Local data buffering is supported, including storage in MDF format and snapshot-based data logging. Its electrical specifications are acceptable for the intended use case, and the estimated unit price is around €8000. However, the main limitation of the Flea4+ CarMedia platform is the limited debugging support. There is no dedicated feedback interface or GUI available to debug compiled binaries. Therefore, if the advanced functionality binaries do not compile or execute correctly, root-cause analysis may be difficult. In addition, developers may need to share open models for integration, which could raise IP-related concerns. The platform also does not directly support MATLAB/Simulink-based workflows.

The MicroAutoBox III/II dSPACE platform provides a mature rapid-prototyping environment and integrates well with dSPACE ConfigurationDesk, MATLAB, C/C++, and Python workflows. It supports standards such as AUTOSAR and is suitable for advanced model-based development. CAN 2.0

communication is supported through the DS1521 bus and network board, while CAN FD capability depends on the selected configuration. The platform does not include native GSM connectivity; therefore, an external GSM module or router is required for cloud communication. MQTT-based cloud transfer and local buffering are possible, but the overall integration effort is higher. The estimated unit price is around €18,000.

2.1.2 Hardware selection decision

Based on the hardware comparison and with the agreement of all model developers and InnoBMS OEMs, **MicroAutoBox III from dSPACE** was selected as the preferred edge hardware platform for the InnoBMS project.

Although the Flea4+ CarMedia platform is more cost-effective and provides integrated GSM/Wi-Fi connectivity, its limited debugging capability, lack of GUI-based feedback, absence of direct MATLAB/Simulink support, possible IP concerns related to sharing open models and uni-directional data flow from vehicle-to-cloud make it less suitable for hosting and validating advanced InnoBMS functionalities.

The dSPACE MicroAutoBox platform was selected because it provides a mature and flexible rapid-prototyping environment, supports MATLAB/Simulink, C/C++, and Python workflows, and enables easier integration and validation of advanced models. It also provides better support for model-based development, debugging, and iterative testing, which are essential during the development phase. Although an external GSM module or router is required for cloud communication and the cost is higher, the platform offers lower technical risk and better compatibility with the project's advanced functionality development needs.

2.2 Edge SW architecture – BOSCH

2.2.1 Introduction to software architecture

The Edge ECU software architecture defines the structural organization of software components within the InnoBMS system. It establishes how application functions, communication modules, and runtime elements are arranged and interact on the Edge platform.
The architecture supports a modular approach, where components such as FMUs and CAN communication modules are integrated via standardized interfaces. This allows consistent data exchange between Edge ECU, VCU, SBMS, and Cloud systems. Additionally, it defines how software components are scheduled and executed, ensuring coordinated operation within the available system resources.
Furthermore, the architecture ensures traceability of all software interfaces by explicitly defining the source, destination, and usage of each signal within the system. This enables a clear mapping between components and their interactions, supporting consistency between the architecture, the Software Interface List, and the integrated FMUs. As a result, data flow across the system can be transparently followed, facilitating integration, validation, and debugging activities.

2.2.2 Architecture for software integration

The software architecture provides the foundation for integrating functionalities developed by different partners into a single, coherent system. It establishes common interfaces, data formats, and execution behaviour, enabling seamless interaction between independently developed software components. This significantly reduces integration effort, minimizes inconsistencies during system assembly, and ensures reliable interoperability across the complete software stack.

The overall InnoBMS software architecture comprises two execution environments: the Edge ECU and the Cloud. Effective communication and coordination between these environments are essential for ensuring the correct operation of the overall system. As part of this task, the focus is on integrating all Edge ECU functionalities into a unified software framework. Table 1 summarizes the advanced functionalities contributed by each project partner and identifies whether they are intended for deployment on the Edge ECU or in the Cloud. This deliverable describes the development of Edge ECU-executable functions only; the implementation of cloud-executable functions is outside the scope of this document.

Table 1. Advanced functionalities and contributions per partner

Partner	Advanced functionality
AVL	Edge ECU • Data-driven algorithms
	Cloud • Data-driven algorithms
CID	Edge ECU • TRA early detection, Lithium plating detection and edge BTM control
	Cloud • BTMS control / MPC algorithm
IDIADA	Edge ECU • SOX estimator
UL	Edge ECU • Reduced order PBM
	Cloud • Full order PBM
VUB	Edge ECU • Degradation aware fast charging and V2G algorithm • Battery pre-conditioning function
Bosch	Edge ECU (not BMS functionalities) • Edge ECU-VCU CAN Manager • Edge ECU- sBMS CAN Manager (VUB & BOSCH)

The architecture becomes particularly critical in collaborative projects where software development is distributed across five partners, suppliers, or companies. In such environments, it serves as a common technical framework that defines integration rules, responsibilities, and communication mechanisms, ensuring that independently developed software modules can be integrated efficiently and predictably. Furthermore, the architecture supports integration activities by structuring dependencies and communication paths, which facilitates validation, debugging, and system-level verification. By providing a common integration framework, it enables reliable deployment of all software components within the Edge ECU.

2.2.3 Architecture views

From a SW architecture perspective, the internal logic of each FMU is not considered relevant. The focus is placed exclusively on the externally visible interfaces, namely inputs, outputs, and parameters. This approach ensures that all required inputs are clearly defined and can be correctly provided by the

surrounding system components, enabling proper integration without dependency on the internal implementation details of the FMUs.

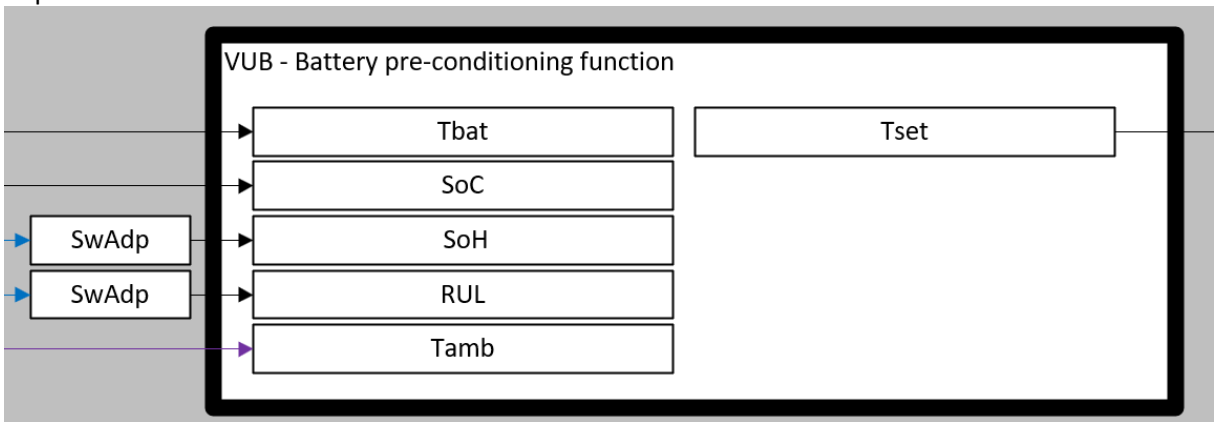

Figure 2-1: Example of FMU interfaces illustrating inputs, outputs, and parameters (BPC function)

Another important aspect is the definition of the boundaries of the Edge ECU software architecture. In this context, only software components deployed on the Edge ECU are within scope. As a result, the architecture includes exclusively the software elements executed on the Edge platform.
The system boundaries are explicitly represented by the communication managers, which handle the interaction between the Edge ECU and external systems such as other ECUs or controllers. These managers define the interfaces between internal application software and external communication channels.

2.2.4 Description of component interconnections

As most signals within the Edge ECU are represented as float or double, no distinction between data types is made at the architectural level. Data type handling becomes relevant only at the communication interfaces, specifically within the CAN and LAN managers, where signal encoding and decoding are required. The following legend defines the signal types used within the architecture:

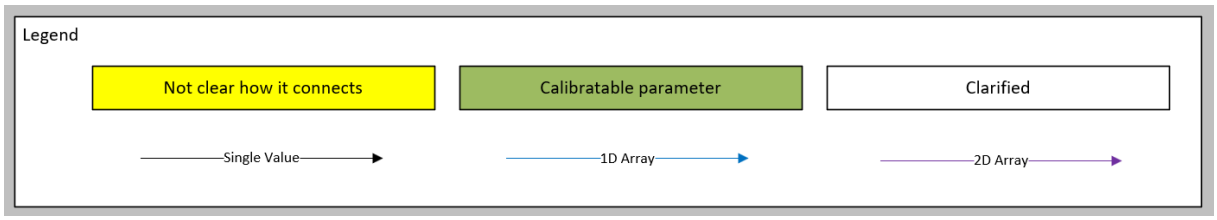

Figure 2-2: Legend of Edge ECU SW Architecture

Each software interface can be defined in one of three states: input, output, or parameter. For inputs and outputs, the clarity of their connections within the architecture may vary depending on the level of definition available. Parameters represent fixed values, typically hard-coded and, in most cases, provided by the respective FMU developer.

As previously defined, signal connections are distinguished based on their structure: single-value signals are represented with black arrows, while array-type signals are represented with blue arrows. This distinction is sufficient, as most signals are implemented using float or double data types.

In addition to the signal classification, each interface is associated with a clearly defined source component (provider). Where the provider is not yet specified or under discussion, the signal is marked accordingly in the architecture. This ensures traceability and highlights open integration points requiring alignment between partners.

From a conceptual perspective, the developed software architecture follows principles similar to the AUTOSAR layered approach. In particular, a separation is introduced between application-level components and basic software elements related to communication services. This separation ensures that communication-related functionalities (e.g., CAN and LAN managers) are decoupled from the application software. As a result, application components can operate independently of changes in the communication stack, supporting portability and reducing dependency on the underlying hardware platform.

2.2.5 Definition of priorities

The highest priority is assigned to the Thermal Runaway Detection functionality, as it directly indicates critical safety conditions. Upon detection, the corresponding signal is immediately transmitted to the VCU via the CAN interface to trigger appropriate vehicle-level actions.
From the Edge ECU software perspective, execution priorities are defined in a sequential manner. The communication managers responsible for receiving data are executed first, ensuring that all required inputs are available. Following this, the FMUs are executed using the updated input values. Finally, the transmission managers are executed to send the computed results to external systems. A detailed description of the execution timing and ordering of each FMU is provided in the Scheduling chapter.

2.2.6 Description of model scheduling

All FMUs and managers are executed cyclically with a fixed step size. It is essential that the FMUs operate using the most up-to-date information; therefore, the Rx managers must be scheduled for execution beforehand.

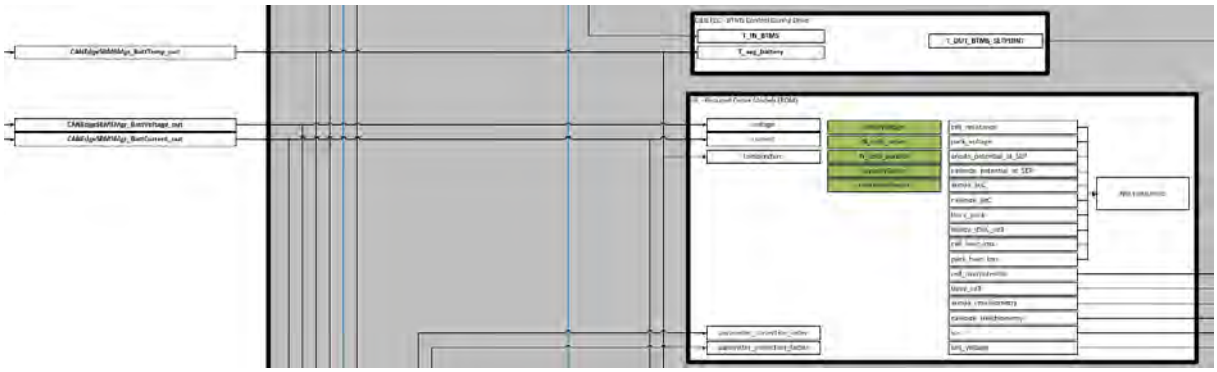

Figure 2-3: Example of FMU interconnections and data flow, including SwAdp handling of interface inconsistencies

An example of the connection is illustrated in the figure above. Alongside the FMUs, a SwAdp is employed to handle any inconsistencies that may arise between the FMUs during integration.

2.3 Edge global SW interfaces - BOSCH

- Description of the **global interface clarification process**
- Explanation of the **importance of alignment between all project partners**
- Description of the **interface table structure**, including the meaning of each column
- Complete **interface list provided as an annex**

This section describes the software interface information collected for each model within the InnoBMS project prior to integration activities. The objective is to establish a consistent overview of the interfaces provided by the different project partners and to organize the specifications required for future integration activities.

In projects involving multiple external partners and independently developed software models, early alignment of software interfaces and connectivity requirements is essential to ensure compatibility between software components. Since stakeholders may follow different implementation approaches and interface definitions, a common understanding of signal structures, data types, dimensions, and communication expectations is required.
The interface specification phase establishes a shared baseline for all participating parties during development and preparation activities. By consolidating and reviewing interface definitions before integration, potential inconsistencies and incompatibilities can be identified early, reducing integration risks and minimizing rework in later project phases.
This process supports the integration of all software models into a unified executable/FMU environment, ensuring that the individual components can communicate and operate together as intended.

2.3.1 Interface Definition Process Flow

The interface definition process flow is required to ensure transparency and facilitate any necessary analyses prior to the start of the integration phase. Collecting interface information from all partners enables early identification of mismatches, inconsistencies, and integration risks related to signals, data representation, and interface behavior.
This chapter provides the interface-related information currently available from the different model providers and serves as the baseline for upcoming integration activities.

- **Interface Template Preparation**: A standardized Excel-based interface template was prepared in order to establish a common structure for collecting and documenting interface-related information across all participating partners and models.
- **Collection of Existing Interface Information**: Available interface specifications, technical documents, model descriptions, and existing signal definitions were reviewed in order to gather the relevant interface information required for integration activities.
- **Interface Information Consolidation**: The collected information was consolidated into the common interface template, including details such as signal names, directions, data types, dimensions, units, descriptions, dependencies, and model-specific assumptions.
- **Partner Alignment Activities**: Alignment discussions were conducted with the involved partners to clarify missing information, validate interface expectations, and ensure a common understanding of signal behavior and model interactions.
- **Interface Review Sessions**: Joint review sessions were performed together with the partners in order to validate the documented interfaces, identify inconsistencies or mismatches, and confirm the correctness and completeness of the interface specifications.
- **Issue Resolution and Specification Updates**: Identified interface discrepancies, dependency conflicts, or unclear definitions were addressed collaboratively, followed by updates to the interface documentation where necessary.
- **Interface Specification Release**: After successful review and alignment, the finalized interface specification package was released and used as the baseline for the subsequent activities.
- **Interface required info:** The collected interface information includes:
    - Model Name
    - SW signal description: Short summary of what the interface will provide
    - SW signal name proposal: proposed name of the interface
    - Data types: Data type of the interface (e.g. uint8)
    - Type of signal: Direction of the interface (Input/Output)
    - Physical Unit: physical measurement unit used to represent the interface (e.g. Amperes, Celsius degrees, Ohm)
    - Physical resolution: step size
    - Physical lower/upper limit: The lowest and the highest value the signal can have

- Periodicity: how many times the signal will be calculated in a given amount of time (e.g. twice / year, once every 0.1 seconds)
- Provider: Source of the interface

2.3.2 Project partners alignment

As part of the interface specification phase, communication and alignment sessions were conducted with each external partner in order to clarify the expected software interfaces and integration requirements for their respective models. These discussions were necessary to obtain a complete understanding of the provided inputs and outputs, signal behavior, and execution expectations.

The interaction with the different stakeholders helped ensure that all interface definitions were consistently understood and sufficiently documented prior to integration activities. This approach also supported the identification of missing information, ambiguities, and potential incompatibilities at an early stage, allowing interface-related concerns to be addressed before the models are combined into a final SW product.

Due to the involvement of multiple external partners across different locations and time zones, coordination activities also presented organizational challenges related to scheduling, availability, and synchronization between teams. Additional effort was therefore required to align discussions and review interface information within the project timelines. Despite these challenges, maintaining continuous communication with all stakeholders was essential for establishing a common integration baseline and ensuring readiness for the upcoming integration phase.

2.3.3 Interface review together with partners

In collaboration with each partner, the interface specifications were defined and consolidated using a common table template to ensure consistency across all models. This standardized format is used to capture key interface attributes such as signal names, directions, data types, dimensions, and functional descriptions, enabling a uniform and comparable view of all model interfaces ahead of integration.

E.g.:

Model Name	SW signal description	SW signal name proposal	Data Type	Type of signal	Physical unit	Physical resolution	Physical lower limit	Physical upper limit	Periodicity	Provider	Obser-vations
Reduced order Model	Predicted cathode SoC	RPBM_cathode_soc_out		Output	%	0.1	0	100	0.1s	RPBM	

An example demonstrating the effectiveness of this activity was the early identification of an interface mismatch between two models. One model provided an output in the form of an array, while the receiving model expected scalar input values instead. Detecting this discrepancy during the interface specification review phase helped prevent potential incompatibilities during the integration stage and reduced the risk of delays caused by late interface corrections.

Dependencies between the different models were analyzed in order to identify the required interactions, signal exchanges, execution relationships, and functional dependencies between the involved software components. This activity provided improved visibility into how the individual models are expected to communicate and interact once integrated into a common environment.

Based on the identified dependencies, alignment discussions were conducted with the corresponding partners to clarify interface expectations, signal usage assumptions, timing considerations, and model

interaction behavior. These discussions were important to ensure that all stakeholders shared a common understanding of the dependent interfaces and their intended functionality prior to the integration phase. The dependency review and alignment activities also supported the early identification of potential integration risks, such as incompatible signal dimensions, inconsistent assumptions regarding data usage, or missing interface definitions. Addressing these topics during the specification phase helped reduce the likelihood of integration issues and minimize the need for late-stage modifications during model integration activities.

2.4 FMU Generation and Cross-Compilation for ARM Linux Targets (BOSCH)

2.4.1 Objective

To support the deployment of simulation models on embedded Linux platforms, a methodology was established for generating Functional Mock-up Units (FMUs) from Simulink models and adapting them for execution on 32-bit ARM architectures. Since the standard FMU export functionality of Simulink mainly supports x86-based targets, an additional cross-compilation stage is required to produce binaries compatible with ARM-based systems. The proposed workflow enables the transformation of a Simulink model into a deployable FMU package that can be executed on embedded Linux devices while maintaining FMI 2.0 compliance.

2.4.2 FMU Generation Process

The FMU generation process begins with the preparation of the Simulink model and the export of a source-code FMU. The workflow requires Simulink Coder, Simulink Compiler, and the FMU Builder add-on. Depending on the operating system used for development, a Linux environment is required either through a native Linux installation or through Windows Subsystem for Linux (WSL2)

During the export phase, the model is configured as a Co-Simulation FMU compliant with FMI 2.0. To enable recompilation for alternative architectures, the generated FMU must include the source code. The resulting package contains the model sources, FMI interface files, and binaries generated for supported desktop platforms.

2.4.3 Cross-Compilation for ARM Architecture

Following FMU generation, the source code contained within the FMU is extracted and rebuilt using an ARM cross-compilation toolchain. The compilation process targets ARMv7 Linux systems and generates a shared library compatible with 32-bit ARM processors. The cross-compilation environment relies on GNU ARM compilers, associated development libraries, and a CMake-based build system.

The extracted FMU sources are compiled using architecture-specific compiler and linker settings. These settings define the target instruction set, floating-point support, and binary format required by the embedded platform. After compilation, the resulting shared library is verified to ensure that the generated binary corresponds to an ARM executable format.

2.4.4 FMU Packaging and Deployment

Once the ARM-compatible shared library has been generated, the FMU package is restructured to include only the files required for deployment. Development artifacts such as source code, build directories, documentation, and desktop-platform binaries are removed from the package. The final FMU is then repackaged and renamed according to the FMI specification.

This approach produces a compact deployment-ready FMU containing only the binaries required for execution on the target ARM system. The resulting package can be integrated into embedded software environments, virtual ECUs, or edge computing platforms that support FMI-based model execution.

2.5 Edge-executable SW models development

This section describes the development of the software (SW) models intended for execution on the Edge ECU (as discussed in Table 1). To ensure consistency across partner contributions, a common documentation template has been established for all executable software models. Each partner shall document its model using the same structure, providing information on model parametrization, FMU functionality, code compilation, validation methodology, and dSPACE testing. This harmonized approach facilitates software development, integration, verification, and deployment within the InnoBMS Edge ECU architecture.

2.5.1 [Model name] – each partner to fill

Each model shall be documented in a **dedicated subchapter** following a common structure.

2.5.1.1 Parametrization

This section describes the role of model-specific parametrization within the project.
It explains which vehicle-specific characteristics influence the parametrization and how parameters are used to adapt the model to a given vehicle configuration.
Key differences in parametrization between the IVECO and Tofaș vehicles shall be clearly identified and described.

2.5.1.2 FMU functional description

This section describes the FMU as a blackbox functional component within the overall system architecture. It shall include:

- The functional role and purpose of the FMU within the system
- A short description of the expected functional behavior, limited to observable input/output relationships

2.5.1.3 Code compilation description

This section describes the code compilation process for the FMU.
It shall outline the compilation workflow, relevant tools, target platforms, and any model specific compilation settings required to generate the FMU

2.5.1.4 DSpace Environment testing

This section documents the testing performed in the DSpace environment. It shall describe:

- The specific dSPACE configuration used
- Integration setup
- Test scenarios executed
- Any relevant constraints or deviations from ideal conditions

2.5.2 Edge ECU – VCU CAN Manager – BOSCH

2.5.2.1 Parametrization

From a functional perspective, both TOFAS and IVECO configurations are supported without requiring any parameterization at the manager level. This is primarily enabled by the fact that the current underlying communication matrix is identical for both configurations. While certain signals may not be relevant for a specific configuration, they remain defined within the system and can be considered as available but unused, without impacting the overall functionality.

The adaptation to each vehicle integration is handled at the Basic Software (BSW) level, where the communication configuration is defined. In this phase, the communication matrix (CAN .dbc) is linked to the Tx and Rx signals exposed by the manager. The configuration activities are performed within ConfigurationDesk and are described in detail in the "CAN Setup" sub-chapter, where the mapping approach and implementation aspects are further elaborated.

Overall, the manager development and integration are aligned with the latest SW Interface List, SW Architecture, and Edge ECU – VCU CAN database, ensuring consistency and traceability across all components involved in the system.

2.5.2.2 FMU functional description

The functionality of the model is intentionally kept simple, as its primary role is to act as a decoupling layer between the application software and the communication configuration. By introducing this abstraction, the application SW—implemented as FMUs—remains independent of the underlying platform and communication setup, allowing for a consistent behavior regardless of the integration context.

In addition to its role in data transmission, the model also provides a consolidated representation of the LPD (Lithium Plating Detection) status. Given that 168 LPD FMU prototypes are instantiated—one for each cell—the model aggregates their outputs before forwarding information to the VCU. Specifically, the overall LPD status is determined using a logical OR approach, such that the detection of a fault in any individual cell results in a positive (true) status being transmitted.

Alongside this aggregated status, the model also ensures traceability by transmitting the position of the affected cell. This is achieved based on the known correspondence between the cell index and its physical position within the battery pack, enabling the VCU to identify the exact location of the detected issue.

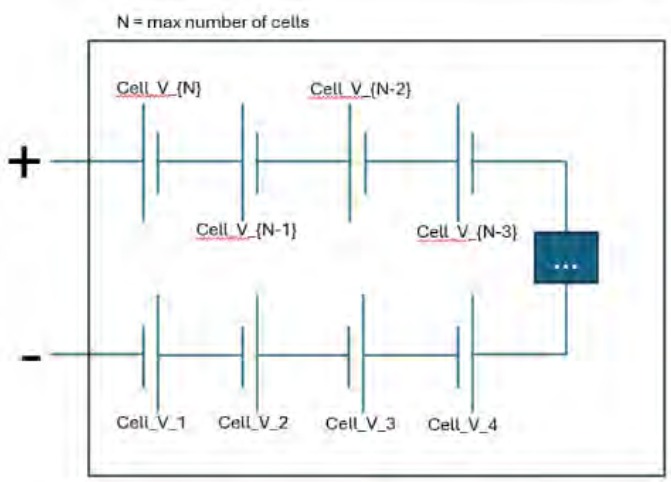

Figure 2-4: Cell location corresponding to the cell index as defined in the communication matrix (CAN .dbc).

For debugging and informational purposes, a dedicated flag is transmitted to indicate that the LPD FMU has not received the necessary environmental conditions required to perform its function. While the interpretation and handling of this information are left to the VCU, the model ensures that prolonged periods during which the LPD is unable to provide a valid confirmation/information are explicitly communicated.

2.5.2.3 Code compilation description

Since the model is developed as a Simulink model rather than an FMU, the compilation and integration are handled directly within ConfigurationDesk. As a result, the model can be imported and used without any additional pre-processing steps or specific prerequisites prior to this activity.

2.5.2.4 Validation method of the FMU

For validation purposes, a CAN setup consisting of two nodes implemented on the MicroAutoBox III was used. In this configuration, the first node represents the model, which transmits signals over the CAN bus, while the second node acts as a virtual “VCU” responsible for receiving these signals.

2.5.2.5 DSpace Environment testing

As the content of this chapter aligns with that presented in the “CAN Setup” chapter, the reader is referred directly to the “CAN Setup” sub-chapter, where the ConfigurationDesk setup is described in detail together with the corresponding hardware configuration.

2.5.3 Edge ECU – SBMS CAN Manager – VUB & BOSCH

2.5.3.1 Parametrization

The manager was initially provided by VUB with support for a single communication matrix, establishing the baseline communication concept. Bosch subsequently extended the implementation by adding support for two additional communication matrices and implementing End-to-End (E2E) protection mechanisms, including CRC and counter handling. To ensure compliance with industry practices and interoperability requirements, the implementation follows the AUTOSAR R20-11 E2E standard.

2.5.3.2 FMU functional description

The manager is responsible for receiving and processing CAN communication signals required by the Edge ECU FMUs. Incoming CAN frames are decoded and subjected to consistency checks, including the verification of End-to-End (E2E) protection mechanisms. In the event of a communication inconsistency or invalid data, the FMU signals the error condition and provides predefined default values to ensure predictable system behaviour. In addition, the FMU performs data structuring and formatting to align the communication data with the expectations of the different project partners. For example, individual cell voltage and temperature signals received via CAN are combined into voltage and temperature arrays before being provided to the corresponding FMU interfaces.

2.5.3.3 Code compilation description

The compilation and integration process is identical to that of the Edge ECU – VCU CAN Manager. As the functionality is delivered as a Simulink model rather than an FMU, it can be imported directly into ConfigurationDesk without requiring any additional pre-processing, or specific prerequisites.

2.5.3.4 Validation method of the FMU

For validation purposes, a dummy SBMS node was implemented to transmit CAN messages representative of a real SBMS. The transmitted signals were received by the SBMS CAN Manager and verified using non-zero test values to confirm correct communication and signal processing.

2.5.3.5 DSpace Environment testing

As the content of this chapter aligns with that presented in the “CAN Setup” chapter, the reader is referred directly to the “CAN Setup” sub-chapter, where the ConfigurationDesk setup is described in detail together with the corresponding hardware configuration.

2.5.4 Reduced order electrochemical model – UL

2.5.4.1 Parametrization

The parametrisation of the electrochemical models requires defining several physics-based parameters. They are usually obtained from electrochemical characterisation measurements on the cell, advanced microscopy techniques from cell teardown, literature-based measurements of the specific material properties, and finally with the optimisation algorithms. The two cells used in the InnoBMS project, i.e. from the IVECO and Tofaș vehicles require a separate parametrisation, since the cells are not identical, however the parametrisation methodology remained the same. After defining the geometry and material properties of the cell, the electrode balancing routine was used to determine the volume fractions of different components of the cell. The unknown parameters were extracted with the optimisation routine coupled with the full-order electrochemical model on the Cloud. Since the both models on the Cloud and Edge share the same physico-chemical basis, the parameters can be directly transferred. More details on the parametrisation methodology can be found in the InnoBMS deliverable D2.3, which covers the advanced functionalities on the Cloud.

2.5.4.2 FMU functional description

The main role of the reduced order electrochemical model on the Edge ECU is to serve as a virtual sensor for internal states of the battery cell, the most important being the cathode and anode potential. It is known that during fast charging, the anode potential can drop below 0 V below Li/Li-ion reference where Li-plating becomes thermodynamically more favourable than intercalation into the active material in the anode. Similarly, high cathode potentials during fast charging can cause degradation of the active material and decomposition of the electrolyte. The developed model calculates these potentials in real time and can support the existing BMS with such advanced functionality to reduce the risk of unwanted side reactions and improve safety and longevity of the cells.

2.5.4.3 Code compilation description

The reduced-order battery model was implemented in the C programming language due to its ability to generate highly optimised machine code with minimal runtime overhead, while also providing excellent portability across different microcontrollers and processor architectures with only minor modifications. These characteristics make C particularly suitable for embedded real-time applications, such as battery management systems and automotive control units.

The code compilation process was managed using a modified *Makefile* supporting multiple target platforms, including Windows, Linux, and ARM-based architectures. For deployment of the Functional Mock-up Unit (FMU) to the dSPACE MicroAutoBox III platform, the source code was compiled under the Linux operating system using the *arm-linux-gnueabihf-gcc* compiler, i.e. a GCC-based cross-compiler intended for 32-bit ARM architectures.

During the development of the model, the *Valgrind* software was extensively used to detect potential memory leaks and invalid memory accesses within the code to ensure flawless online operation of the model

2.5.4.4 dSPACE environment testing

The developed model was finally tested also on the target MicroAutoBox III platform. The model is designed to run in a real-time with the time step of 100 ms or frequency of 10 Hz, so the 100 ms time-stepping of the RTOS system was set, and the initial real-time factor of the model was around 0.02. Test scenarios were the same as they were already presented in the previous subsections with the

main goal to check, if the model compiles, takes the correct inputs, calculates the expected outputs and executes each time step faster than 100 ms. The only deviation from the offline tests in MATLAB Simulink is the fact that apparently (not fully confirmed) the MicroAutoBox III platform does not call the *fmi2SetReal* function inside the FMU during the initialisation, which meant that some minor changes in the code were needed.

2.5.5 SoX Edge Estimator- IDIADA

2.5.5.1 Parametrization

The SoX Edge estimator uses a closed loop algorithm to estimate key battery indicators as SoC, SoH and SoP based on inputs measurement such as current, voltage and temperature. This means that it needs to compute a model representing the battery behaviour so the model output (voltage) can be compared with the measurements received in real time to correct the estimated values. Consequently, the battery model must be adapted according to the specific vehicle where the software will run. Because of this reason, the precision of the estimation results is dependent on the accuracy of the used model. As the model needs to be computed in real time in the Edge device, there is a trade-off between accuracy and computational complexity. In this case, an electrical equivalent RC circuit model provided by UL has been integrated into the estimation algorithm.

The battery model is adjusted based on test results of the cell installed in each battery to adapt the resistances and capacitors in the equivalent circuit. Moreover, the battery capacity, open circuit voltage and cell configuration is also adapted depending on the target vehicle (Iveco or Tofas). The whole structure of the model and the algorithm is maintained for any of the vehicles so only an update of the parameters is required to configure the software.

2.5.5.2 FMU functional description

A black-box Functional Mock-up Unit (FMU) has been generated from the Simulink model of the SoX Edge estimator. The FMU provides a standardized, tool-independent component that can be deployed into the target processor. The SoX Edge estimator provides at runtime an estimate of key battery states that are not directly measurables, supporting the diagnostic, control, protection and decision-making functions of higher-level algorithms running in the Edge BMS. The FMU provides a standalone block which can be integrated in the overall system architecture, interacting through its inputs and outputs ports without disclosing the internal logic and implementation details. Figure 2.5 shows the Simulink block which has been used to generate the FMU.

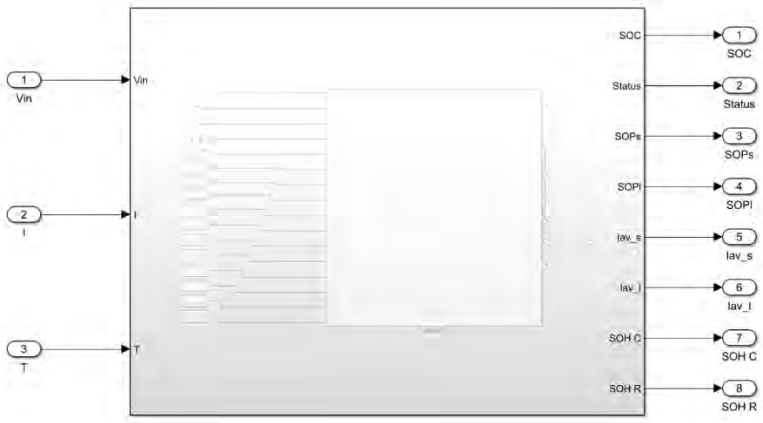

Figure 2-5: Simulink subsystem for FMU generation of SoX Edge estimator

The required inputs of the system are the current, voltage and temperature of the battery measured by the BMS. The SoX Edge estimator process all these data in real time to calculate the outputs representing the main indicators for State of Charge, State of Health and State of Power. These values are shown as a number varying between 0 and 1, being 0 the state where the minimum required

performance cannot be achieved and 1 the state where the maximum performance is delivered for each parameter.
The SoH is represented by two different parameters: SoH_C, which represents the state of health degradation estimated based on the capacity reduction and SoH_R, which represents the state of health estimated according to the internal resistance increase.
Similarly, the SoP is represented by two parameters: SoP_S represents the state of power at short term (pulse performance), while SoP_L represents the state of power at long term. The available current calculated in both cases is also reported.

2.5.5.3 Code compilation description

The SoX Edge estimator has been developed in the Matlab/Simulink environment. The Simulink model has been enclosed in a subsystem with input and output ports which is exported as a standalone FMU. The inputs gather all the information provided by the BMS (current, voltage and temperature) and the outputs give the beforementioned state indicators.
As Simulink runs in a 64 bits environment, it does not allow to generate a FMU for a 32 bits environment as it is the case of the target processor. Because of this reason, the FMU needs to be recompiled in a second phase out of Matlab/Simulink. Consequently, the first FMU has to be generated from Simulink with access to the source code. The second FMU is compiled from the first FMU to adjust the target processor and protect the source code.

#	Target processor	Black box
1st FMU	64-bit ARM	No
2nd FMU	32-bit ARM Linux	Yes

The FMU generation process is summarized in the following steps, according to guidelines provided by Bosch:

1. Generation of 1st FMU in Simulink

Requirements:
- Matlab Coder, Matlab Compiler, Simulink Coder and Simulink Compiler licenses.
- FMU Builder Add-on for Simulink

It has been seen that the generation for ARM processor works properly from 2025 Matlab/Simulink version. When generating from 2024 version, code includes file references for Intel processors as “emmintrin.h”, which causes recompilation failure in the second phase of the process.

The FMU generation in Simulink has 2 main steps:

1.1 Select target hardware
1.2 Set Fixed-step solver

After these steps are done, a 64 bits FMU for an ARM target has been created.

2. Recompilation of FMU in Linux environment

Requirements:
- Python 3 and pip
- FMPy library
- Enable sudo from settings

The FMU recompilation has 2 main steps:

2.1 Environment Setup
- Install ARM Cross-Compilation Toolchain

2.2 Compile 32-bit ARM Shared Library
- Extract FMU source code
- Prepare build directory
- Set cross-compilation environment variables
- Build
- Packaging final FMU

After the recompilation, the resulting FMU file is ready for the deployment on the 32-bit ARM target system.

2.5.5.4 dSPACE Environment testing

Once the FMU has been generated and validated, the model is tested in the dSPACE environment. For this purpose, a Simulink model is prepared with the inputs and outputs which are assigned to the respective ports in the FMU. The Simulink model also is responsible for loading the profile of the voltage, current and temperature inputs by means of a look-up table. This look-up table represents the samples that are measured by the BMS at each sample time and that the SoX estimator requires for its calculations. Again, the current is defined as positive in discharge.

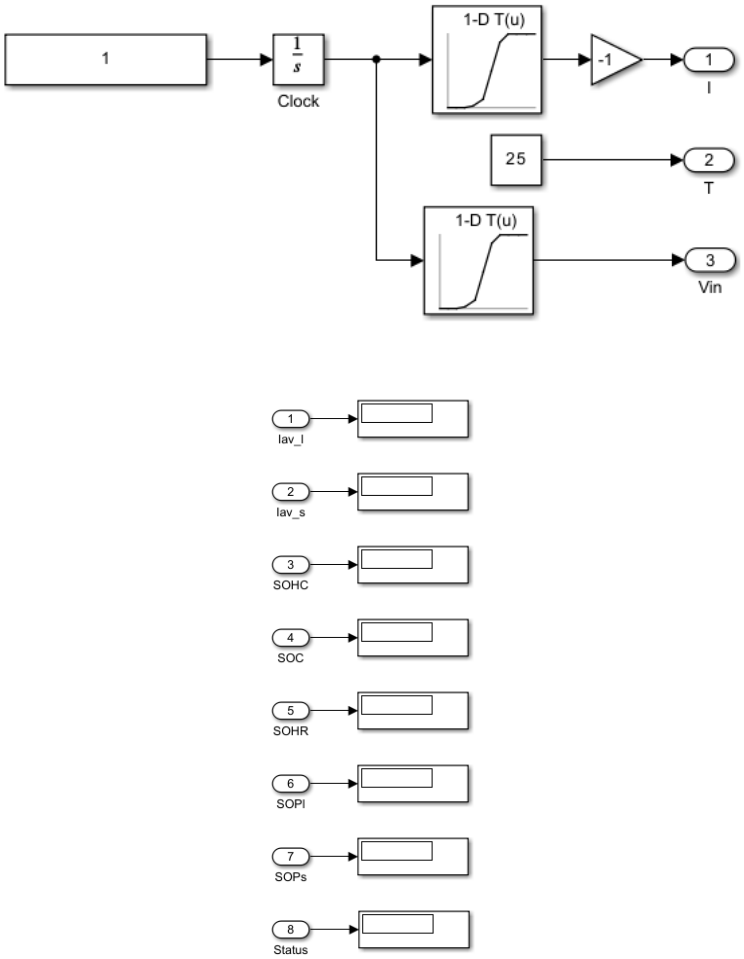

Figure 2-6: Inputs and outputs configuration in Simulink

In the dSPACE configuration software, the FMU is loaded and the execution time is configured according to the defined sampling time of 0.1s. Finally, the configuration is loaded and run into the dSPACE.

The test scenario executed uses the same input profile that the one simulated in Matlab/Simulink. This allows to validate that the behaviour of the FMU deployed into the dSPACE is the same seen in the previous simulations. It is important to take into account that the testing on the dSPACE occurs at real

time, because it receives the input values according to the established sampling frequency, which is considered inside the FMU for the corresponding calculations. Therefore, it is required a long testing time to obtain results from long duration profiles.
As it is known, the SoC and SoP vary slowly according to the current demanded to the battery and it can take several hours to have a significant variation. In addition, the complete validation of SoC estimation can need several charge and discharge cycles, which increases even more the testing time. This issue is more important in the SoH, as this parameter can need a great number of cycles before an important change can be appreciated. However, thanks to the usage of the same test scenario as the one used in Simulink, it is possible to infer that the long-term behaviour of the estimator will be the same already studied as long as the results from the short-term testing are the same as in the previous simulations.

2.5.6 Data Driven Model – AVL

2.5.6.1 Parametrization

Our data-driven model is used to predict the State of Health (SoH) of the battery and Remaining useful Life (RUL). Parametrization serves to adapt this shared, universal machine learning model to both vehicle platforms without altering the foundational software logic. Cell specific physical characteristic such as the Nominal energy capacity, and operating Voltage, Current, Temperature boundaries dictate how the functional mock-up unit (FMU) scales and normalizes sensor data prior to feeding it as input to the ML model for inference. The key differences in parametrization between the IVECO and TOFAS configurations stem directly from these fundamental physical disparities. By updating these specific configuration, the system recalibrates the mathematical data-binning histograms to accurately match the unique chemistry of each specific battery pack. Consequently, we also need to change the input and output structure of the FMU based on the battery configurations.

2.5.6.2 FMU functional description

The FMU operates as an edge-based, data-driven estimator. Its primary role within the system is to predict the real-time State of Health (SoH) and project the Remaining Useful Life (RuL) of the battery pack. To achieve this, it aggregates battery signals into structured mathematical formats (voxel bins) and processes this data through an embedded machine learning inference engine. The FMU is designed with a strictly stateless architecture, meaning it does not store any data locally. Instead, it acts purely as a computational engine.
Functioning as a black-box component, the FMU's expected behavior is defined by a strict and observable input-output contract. At every execution step, the FMU ingests continuous raw sensor measurement, specifically total pack current, individual cell voltages, and individual cell temperatures alongside a discrete Real-Time Clock (RTC) Unix timestamp. At every Key-On initialization, it requires the master system to inject discrete state arrays that represent the battery's historical degradation and accumulated energy. In response to these inputs, the FMU outputs the real-time predicted SoH for each individual cell, a scalar prediction for the overall pack RuL, and the live, updated internal state arrays (energy integrators and histogram bins). While the FMU continuously absorbs and integrates sensor inputs at every computational tick, its primary machine learning predictions are time gated. The observable SoH and RuL outputs will dynamically update only when the incoming Unix timestamp crosses a predefined 24-hour boundary. The master system must continuously observe the output state arrays and capture them precisely at Key-Off to successfully persist the vehicle's degradation history.

2.5.6.3 Code compilation description

The compilation workflow for the FMU is a multi-stage process designed to seamlessly bridge the machine learning environment with the embedded C architecture. First, the neural network is converted into the Open Neural Network Exchange (ONNX) format using the tf2onnx library. The ONNX format is utilized because it completely decouples the model from heavy Python training environments, allowing the neural network to be executed highly efficiently in C on embedded hardware. Next, a Python-based pre-processing pipeline analyzes this ONNX model to extract

structural metadata. This step automatically generates dynamic C-headers—which define the exact memory maps for the neural network weights—alongside the modelDescription.xml file required to define the FMI 3.0 Co-Simulation interfaces. Once the interfaces and memory maps are generated, the core C source files, which contain the stateless mathematical integrators and execution logic, are compiled into the final executable binary. To ensure isolated and stable execution without causing integration issues on the target platform, the system employs a zero-dependency architecture. The pre-compiled ONNX Runtime shared library (onnxruntime.so or .dll) is explicitly bundled alongside this executable. Finally, to create the actual FMU, these components are packaged into a standardized directory structure. The compiled binary, the dynamically loaded ONNX Runtime library, and the modelDescription.xml file are organized into specific system folders and compressed into a standard ZIP archive with a .fmu extension, delivering a completely self-contained, deployable artifact.
The relevant tools utilized in this workflow include Python for automated interface generation, a standard C compiler for the core functional logic, and the ONNX Runtime C-API to facilitate machine learning inference. The code is initially developed and compiled for Windows x86_64 architectures to undergo foundational testing. Following successful verification, the source code is then cross-compiled for Linux/ARM architectures for deployment on the dSPACE MicroAutoBox III using the arm-linux-gnueabihf-gcc compiler.

2.5.6.4 dSPACE environment testing

This section documents the testing performed in the dSPACE environment. Testing in the dSPACE environment focused on hardware integration and architectural compatibility. The specific dSPACE configuration utilized was the MicroAutoBox III, which operates on an ARM32 architecture. The integration setup involved deploying the cross-compiled `.fmu` archive—containing the ARM32 binaries and the bundled `onnxruntime.so` library—directly onto the hardware. The test scenario executed involved stimulating the FMU with constant, artificial Voltage, Current, and Temperature values. The successful generation of inference results from these inputs served as the definitive passing criteria, confirming that the standalone C-code and the ONNX Runtime library were correctly cross-compiled, dynamically linked, and executing stably on the ARM32 processor.

2.5.7 TRA early detection, Lithium plating detection and edge BTM control – CID

2.5.7.1 Parametrization

None of the three algorithms has parametrization. All the algorithms are based on data received as inputs.

2.5.7.2 FMU functional description

2.5.7.2.1 TRA detection algorithm

The Thermal Runaway (TRA) Detection FMU is designed to identify the presence of abnormal gas generation events that may indicate the onset of thermal runaway conditions within a battery system. The algorithm continuously monitors gas concentration measurements and evaluates them against predefined detection criteria to determine whether a potential thermal event is occurring. These predefined detection criteria have been set as per the Thermal Runaway testing campaign carried out. The FMU mainly exchanges the following signals:

Signal	Type	Description
Gas	Input	Gas concentration measurement acquired from the battery monitoring system.

TRA detection flag	Output	Detection flag indicating the presence of gas levels consistent with a potential thermal runaway event.

Detection Principle
During abnormal battery operation, electrochemical and thermal degradation mechanisms can generate gaseous by-products before a thermal runaway event fully develops. By continuously monitoring gas concentration levels, the algorithm can identify early indications of cell failure and thermal instability.
When the measured gas concentration exceeds the predefined detection criteria, the FMU activates the detection flag, providing an early warning signal that can be used by higher-level safety functions to initiate mitigation actions.

Safety Function
The generated detection flag can be used to:

- Trigger thermal runaway warning mechanisms.
- Activate safety procedures.
- Support fault diagnosis functions.
- Enable protective actions at system level.
- Improve battery safety through early event detection.

Edge Deployment
The algorithm is packaged as a Functional Mock-up Unit (FMU) and can be deployed on embedded edge-computing platforms using standardized FMI interfaces.

2.5.7.2.2 Lithium plating detection

The Lithium Plating Detection FMU is intended to identify charging conditions that may lead to lithium plating within lithium-ion battery cells. The algorithm analyses the evolution of cell voltage during operation and determines whether the observed behaviour is consistent with the onset of lithium plating phenomena.

The FMU mainly exchanges the following signals:

Signal	Type	Description
Cell voltage	Input	Measured cell voltage used to evaluate potential lithium plating conditions.
Li plating detection flag	Output	Detection flag indicating that voltage behaviour associated with lithium plating has been identified.

Lithium plating can occur when metallic lithium deposits on the anode surface instead of being properly intercalated into the electrode structure. This phenomenon is typically associated with adverse charging conditions and can accelerate battery degradation while increasing safety risks.

The algorithm continuously monitors the cell voltage evolution after a charging process and evaluates characteristic patterns that may indicate the presence of lithium plating. When the detection criteria are fulfilled, the FMU activates the detection flag to notify the Battery Management System of the potential event.

Battery Protection Function
The generated detection flag can be used to:

- Reduce charging current.
- Adapt charging strategies.
- Modify thermal management operation.
- Prevent accelerated battery ageing.
- Improve battery safety and lifetime.

Edge Deployment
The algorithm is packaged as a Functional Mock-up Unit (FMU) and can be deployed on embedded edge-computing platforms using standardized FMI interfaces.

2.5.7.2.3 BTMS edge control

The BTMS Edge Control FMU is designed to determine the optimal thermal management system temperature setpoint for battery conditioning. The algorithm continuously evaluates the thermal state of the battery pack and the current operating conditions of the Battery Thermal Management System (BTMS) in order to generate an appropriate temperature target for the cooling or heating circuit.

The FMU exchanges the following signals:

Signal	Type	Description
T_avg_battery	Input	Average battery temperature, representing the overall thermal condition of the battery pack.
T_IN_BTMS	Input	Temperature of the thermal fluid entering the Battery Thermal Management System (BTMS).
T_OUT_BTMS_SETPOINT	Output	Temperature setpoint generated by the controller to regulate the BTMS operation.

Thermal Control Strategy

The primary objective of the controller is to maintain the battery temperature within its optimal operating window. Lithium-ion batteries exhibit strong temperature dependency, with performance, efficiency, charging capability and ageing mechanisms being significantly influenced by cell temperature.

When the battery temperature exceeds the desired operating range, the controller can reduce the BTMS setpoint to increase cooling effectiveness and remove excess heat from the battery pack. Conversely, when the battery temperature is below the target range, the controller can increase the setpoint to promote battery warming and improve electrochemical performance.

The use of both battery temperature and coolant inlet temperature allows the controller to consider not only the battery thermal state but also the thermal conditions available in the cooling circuit. This enables a more efficient thermal regulation strategy by adapting the requested setpoint to the actual heat transfer capability of the BTMS.

Battery Protection and Efficiency

By continuously adjusting the BTMS temperature reference, the FMU contributes to:

- Prevention of battery overheating during high-power operation.
- Reduction of temperature gradients within the battery pack.
- Improvement of charging and discharging efficiency.
- Mitigation of accelerated ageing mechanisms associated with prolonged exposure to high temperatures.
- Maintenance of battery performance under low-temperature conditions.
- Enhancement of overall battery lifetime and system reliability.

Edge Deployment

The controller is packaged as a Functional Mock-up Unit (FMU), allowing deployment on embedded edge-computing platforms. The FMU executes locally on the target processor and exchanges data with the Battery Management System and thermal management controller through standardized FMI interfaces. This architecture facilitates integration into vehicle control systems while preserving algorithm portability and intellectual property protection.

2.5.7.3 Code compilation description

Deployment of the Li plating detection, TRA early warning and BTMS edge control algorithms on the target platform requires the generation of a Functional Mock-up Unit (FMU) compatible with a 32-bit ARM Linux architecture. Since Matlab/Simulink only supports FMU generation in a 64-bit environment, an intermediate FMU containing the source code is first created and subsequently recompiled for the target processor.

The algorithms themselves were developed in Matlab/Simulink and organized within a dedicated subsystem exposing the required input and output interfaces. The subsystem processes measurements received from the BMS together with ambient temperature information provided by the thermal management system.

2.5.7.3.1 TRA detection

Experimental TRA testing has been carried out in order to generate Gas sensor data with each of the battery cells provided by IVECO and TOFAS.

Based on those results and taking into account the temperature sensing data of the experimental tests, CID has ensured that the TRA detection flag is activated when the gas concentration data has reached to the level defined.

2.5.7.3.2 Lithium plating detection

Experimental testing has been carried out in order to generate Lithium plating in the cells provided by IVECO and TOFAS.
Several cell voltage measurements have been done in order to ensure, that the voltage drop caused by lithium plating during relaxation period is identified.

2.5.7.4 Dspace environment testing

As the three FMUs have been validated previously in MATLAB simulink, the same validation has been carried out in dSpace environment. CID has ensured that the response of the FMUs is the same in dSpace and in Simulink.

2.5.8 Preconditioning, Fast Charging, and V2G – VUB

2.5.8.1 Parametrization

Model-specific parametrization was implemented through vehicle parameter sets while keeping the same preconditioning and fast charging/V2G FMU algorithms for both Iveco and Tofas vehicles. The parameters account for differences in cell chemistry and capacity, pack series/parallel configuration, OCV and resistance maps, voltage/current/temperature limits, charger/V2G capability, and BTMS dynamics. Common FMU runtime inputs are Tamb_forecast_C, Tpack_raw_C, SoC_raw_pu, SoH_raw_pu, RUL_raw_s, and t_to_fastcharge_s, with Pgrid_req_raw_W additionally used by the fast charging/V2G FMU. Outputs are Tpack_ref_C, Ipack_ref_A, and Vpack_ref_V. Pack-average temperature and SoC are appropriate supervisory values, while SoH and RUL use a conservative pack-level estimate representative of the weakest cells or modules rather than a simple arithmetic average.

2.5.8.2 FMU functional description

2.5.8.2.1 Preconditioning FMU

The preconditioning FMU acts as a supervisory thermal-reference generator that prepares the battery pack for a scheduled fast-charging event. Its purpose is to determine a degradation-aware pack-temperature reference that places the battery in a safe and lifetime-favourable thermal condition when fast charging begins, while accounting for the expected ambient conditions, present battery state, battery health and available preparation time. The FMU does not directly control the heater, coolant pump or thermal chamber, it provides the temperature reference to the downstream thermal-control system. This role is consistent with the project definition of preconditioning as preparing the battery for fast charging under thermal, timing and battery-life constraints.

Signal	Type	Definition
Tamb_forecast_C	Input	24-hour ambient-temperature forecast for the following day, downloaded once per day
Tpack_raw_C	Input	Representative pack temperature measured by the SBMS
SoC_raw_pu	Input	Pack SoC supplied by IDIADA's SoX FMU
SoH_raw_pu	Input	Pack SoH supplied by AVL's data-driven model
RUL_raw_s	Input	Pack remaining useful life supplied by AVL's data-driven model
t_to_fastcharge_s	Input	Remaining time before the scheduled fast-charging event, configurable through the Raspberry Pi
Tpack_ref_C	Output	Degradation-aware reference temperature for the battery pack

From the external system perspective, the FMU converts the ambient forecast, present battery condition and remaining time before charging into a single pack-temperature reference. As the scheduled charging time approaches, Tpack_ref_C is adjusted so that the measured pack temperature is driven toward a suitable fast-charging condition. The reference may request heating, cooling or no significant temperature change, depending on the initial pack temperature, forecast ambient temperature, SoC, SoH and RUL. The output remains bounded by the valid battery and thermal-system operating range. When no preconditioning action is required, the reference remains close to the current or lifetime-favourable pack temperature.

2.5.8.2.1 Fast Charging and V2G FMU

The fast charging and V2G FMU is a degradation-aware supervisory controller whose function depends on the demonstrator. In the IVECO case, it coordinates fast charging and bidirectional V2G operation by generating battery-pack current, voltage and temperature references according to the requested grid power and the battery's present state and health. In the Tofaș case, fast charging and V2G power control are not implemented, the FMU is used only to generate a degradation-aware battery-temperature reference during charging. The FMU provides references to downstream charger, converter and thermal-control systems and does not implement their internal high-frequency control loops.

Signal	Type	Definition
Tamb_forecast_C	Input	24-hour ambient-temperature forecast for the following day, downloaded once per day
Tpack_raw_C	Input	Representative pack temperature measured by the SBMS
SoC_raw_pu	Input	Pack SoC supplied by IDIADA's SoX FMU
SoH_raw_pu	Input	Pack SoH supplied by AVL's data-driven model
RUL_raw_s	Input	Pack remaining useful life supplied by AVL's data-driven model
t_to_fastcharge_s	Input	Remaining time before the scheduled fast-charging event, configurable through the Raspberry Pi
Pgrid_req_raw_W	Input	signed grid-side power request, configurable from the Raspberry Pi
Ipack_ref_A	Output	Battery-pack fast charging/V2G current reference
Vpack_ref_V	Output	Battery-pack fast charging/V2G voltage reference
Tpack_ref_C	Output	Degradation-aware reference temperature for the battery pack

For the IVECO case, the FMU converts the requested grid power and battery condition into bounded current, voltage and temperature references. During charging, it produces a negative current

reference and adjusts the voltage and temperature references to complete charging while limiting battery degradation. During V2G operation, it produces a positive current reference and reduces or rejects the requested power when SoC reserve, temperature, voltage, SoH, RUL or lifetime constraints become limiting. When no power exchange is requested, the current reference returns to zero.

For the Tofaș demonstrator, the observable function is limited to degradation-aware temperature control during charging. The FMU evaluates the ambient conditions, measured pack temperature, SoC, SoH and RUL and generates Tpack_ref_C for the vehicle BTMS. The reference may request heating, cooling or temperature maintenance to reduce resistance, avoid harmful charging conditions and limit ageing. The FMU does not determine the charging current or voltage and does not perform V2G control in the Tofaș configuration.

2.5.8.3 Code compilation description

Both the preconditioning FMU and the fast charging/V2G FMU were first exported from MATLAB/Simulink as FMI 2.0 co-simulation FMUs using a fixed-step solver, with source-code access enabled so that the generated C/C++ code and build metadata could be cross-compiled. The FMUs were then processed in a Windows subsystem for Linux environment using FMPy, CMake, and the ARM GNU cross-compilation toolchain. FMPy generated the CMake project, after which the model code was compiled for the 32-bit ARM Linux hard-float platform used by the dSPACE MicroAutoBox III, with the resulting shared library placed in binaries/linux32. The generated binary was verified as an ARM 32-bit ELF shared object before packaging.

For deployment, the default simulation stopTime was removed from modelDescription.xml so that the FMUs could run continuously on the MicroAutoBox III. The generated source-code directory, source-file declarations, temporary build files, documentation and non-target binaries were then removed, leaving only the compiled ARM library and the required FMI metadata. The cleaned FMU contents were repackaged as Precond_FMU_edge.fmu and FastCharge_V2G_FMU_edge.fmu. The same compilation workflow was used for both models. The main model-specific settings were the Simulink model identifier, binary name, fixed-step execution settings and corresponding FMU input/output definitions. The binary name was retained according to the original model identifier, even when the final FMU archive used the _edge suffix. Final compatibility was checked against the GCC and glibc requirements of the installed dSPACE release.

2.5.8.4 DSpace Environment testing

The FMUs were integrated in ConfigurationDesk 2025-B for execution on the dSPACE MicroAutoBox III. Each FMU was added as a model implementation, assigned to a periodic task under an application process, built, and downloaded to the MABIII. Testing was performed in ControlDesk, where relevant inputs were tuned and the FMU outputs were displayed and monitored. Test cases covered nominal, cold and hot battery conditions, different SoC/SoH/RUL values, scheduled preconditioning, and (only for IVECO) fast-charging and V2G power requests. For Tofaș, only the degradation-aware temperature output was used during charging. The main limitation was that several inputs were manually varied in ControlDesk instead of being supplied by the final SBMS, SoX, AVL and Raspberry Pi interfaces. In addition, the IVECO setup used a thermal chamber and limited battery hardware rather than the complete vehicle BTMS and full pack.

3 SW for sub-Pack Battery Management System (sBMS)

To ensure compliance with the Grant Agreement details and requirements, the sub-Pack Battery Management System (sBMS) unit is required to:

1. Ensure safe cell data readings
2. Ensure required Cyber Security level, as per InnoBMS-L1-R046 as described in D1.1 Requirements and Interface Definition
3. Ensure safety of the battery operations (monitoring cell voltages, temperatures, current and acting on the main contactors)
4. Dispatch data related to battery operations to the Edge and VCU
5. Use data coming from Edge to enhance battery operations.

It's required that the SBMS unit complies with the ISO26262 standard and, in particular, that the HW and SW development process follows its guidelines, i.e. the V-Model (see Figure 3-1).

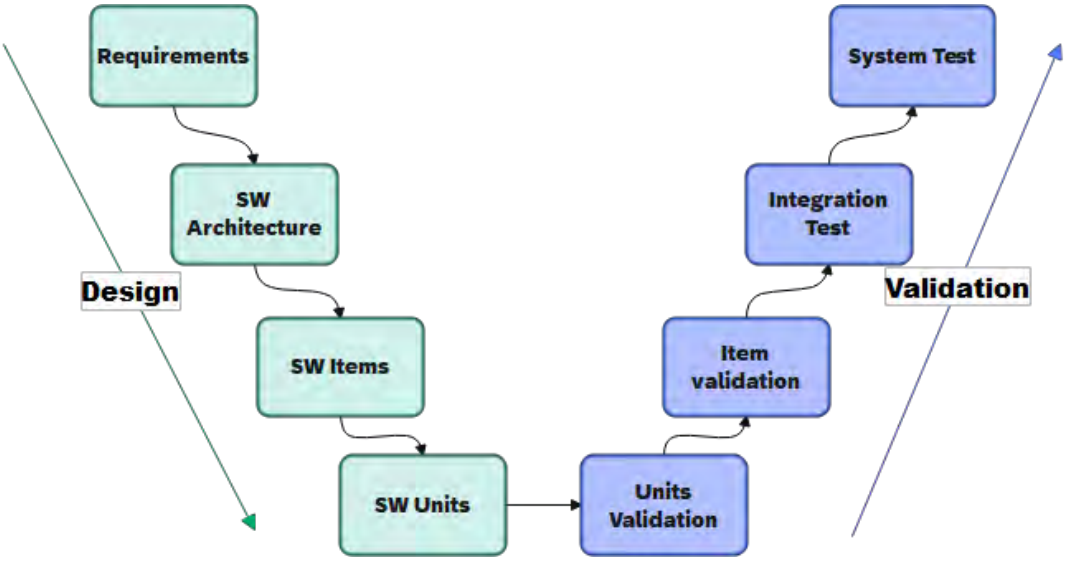

Figure 3-1: V-model process, SW-focused for sBMS.

This process has been followed and certified for the realization of the Gen5m unit and followed for all the modifications needed for Gen5mPLUS, to ensure standard automotive production level functional safety.
The process was followed during the InnoBMS SW release, in fact the requirements concerning the SW functions (as in Figure 3-1) have been linked to specific SW functions that implement them. Consequently, the development process followed the V-model guidelines, resulting in traced links between the requirements and the related tested SW units.

3.1 sBMS SW architecture-FMF

3.1.1 Ensure Safe Cell Data Readings

The sBMS that FMF provided as baseline, referred to from here on as Gen5m unit, is coupled with specific CMCs delivered by FMF. These two pieces of hardware make up the V1 board referred to later on in this report. This SBMS is already in production since May 2024 and is certified ASIL-C automotive standard with ISO26262.

The CMBs for this project have been designed and manufactured by the consortium partner AVL-SFR and the WNM from Texas Instruments has been selected by AVL-SFR, with support from Potenza Technology. Most of the needed adaptation for the specifics of this project are related to the communication with the WNM (that, together with CMBs, falls under AVL-SFR responsibility as highlighted in Figure 3-2).

For better understanding of the WNM-CMBs communication and pack architecture, we refer to deliverable D4.3 Advanced pack-level assembling for ultimate battery pack performance.

Initially, to fulfil the wireless communication requirement, multiple possible solutions were evaluated. They were narrowed down to two possible strategies:

1. WNM as simple Wireless to CAN Gateway, agnostic about CMB management and data being transmitted
2. WNM as manager of the CMBs other than wireless to CAN gateway (see Figure).

Solution N.2 was chosen, to:

- Minimize the risk: choosing an off-the-shelf product makes it possible to rely on the supplier's tests and ensured reliability
- Clearly define partners areas of interest: partners were able to define a clear separation between FMF and AVL responsibility and agree on an interface between the two.
  As shown in Figure 3-2, WNM to CMCs communication is responsibility of AVL, together with tools to make the CAN communication possible (i.e. the DBC file). It's under FMF responsibility to adapt BMS SW to make sure that the CAN communication works. Collaboration is needed for a correct CAN interface setup.

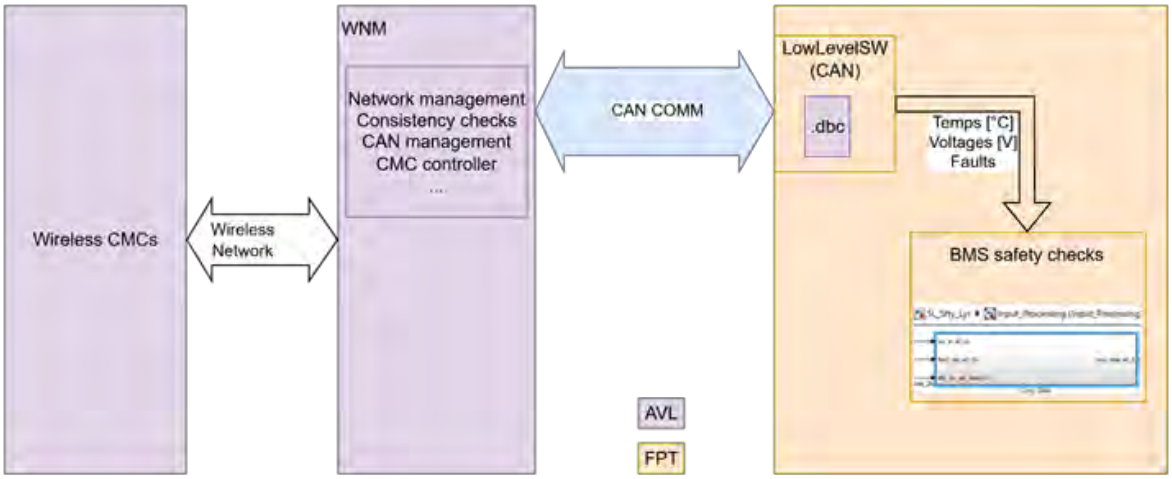

Figure 3-2: Detailed architecture and workflow of SBMS and WNM

The scouting process for the supplier of the WNM and the CMBs presented some difficulties that led to a delay. Initially a supplier was picked, but it was impossible to find an ongoing support agreement. Following a review of the available options on the market, a new supplier, Texas Instruments was selected based on their compatibility, function, availability and support for this research project.

This delay in the supplier choice during months 19-22 cascaded into a delay of the SW preparation since, without a well-defined CAN communication strategy, the final SW could not be released.

3.1.2 Ensure required Cyber Security level

The Gen5m BMS produced by Potenza Technology does not meet the required CySec level for InnoBMS, that's why, as described in deliverable D4.3, section 2.1.3.2, the level of CySec of the BMS are enhanced integrating a specific CAN transceiver. From a SW perspective, this change results in LLSW changes. The only affected unit will be the CAN communication handler inside the SW, highlighted in Figure 3-3.

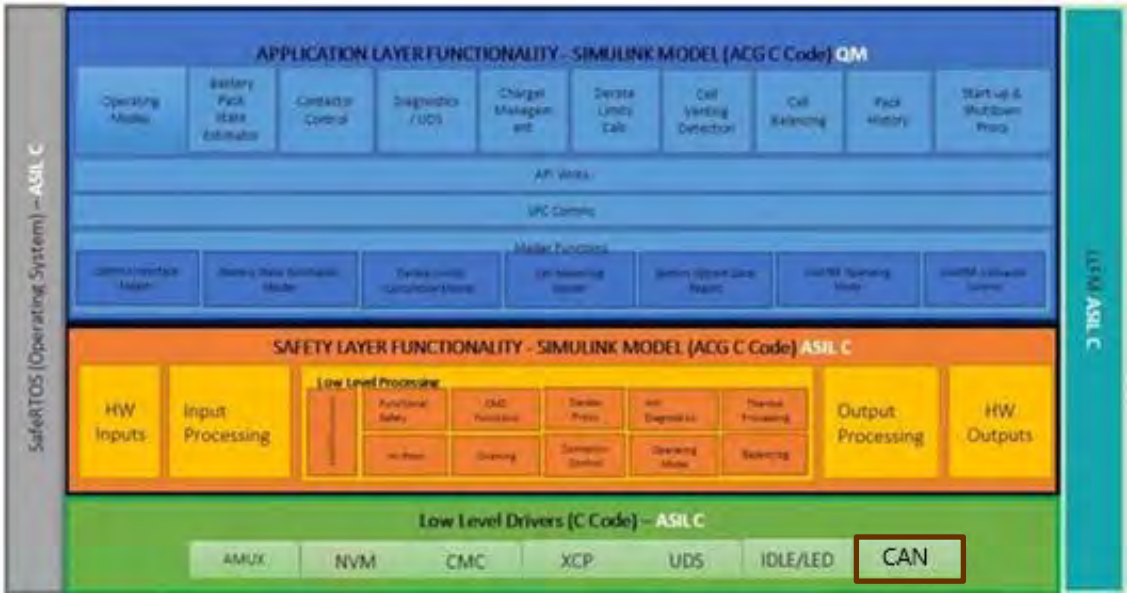

Figure 3-3: Gen5m SW architecture. CAN SWC highlighted.

3.1.3 Ensure safety of the battery operations

The Gen5m SBMS that FMF provided as baseline is, as described in D4.3: a battery management controller designed to monitor, manage, and protect 400V and 800V battery systems at a pack level. Cell level data is provided by an analogue front end, such as the Cell Monitoring Board (CMB). The SBMS is designed to support single or multi pack battery systems, is built on top of a single safety critical microcontroller with lockstep architecture and is certified integrity level of ISO:26262 ASIL C.

Therefore, the battery safety functions are ensured by the non-modified version of the SBMS provided by Potenza Technology.

3.1.4 Dispatch data related to battery operations to the Edge-ECU and VCU

The data output from the CMBs (i.e. cell voltages, temperatures), plus the pack current and voltage, are needed by the EdgeECU to run the advanced algorithms and by the VCU to properly monitor and control the vehicle-related functions.

This data streams on the CAN network internal to the battery pack, connecting WNM and SBMS (See Figure 3-4), so the SBMS is required to relay this information outside the battery pack, guaranteeing data integrity. In Section 3.2.2 the implementation of this function is further discussed.

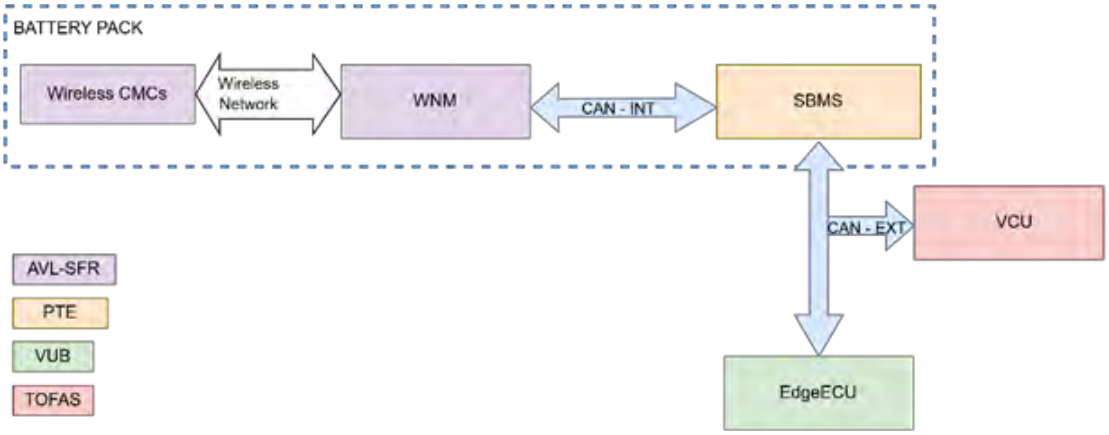

Figure 3-4: InnoBMS architecture, internal and external CAN interfaces highlighted.

3.1.5 Use data coming from Edge to enhance battery operations.

Advanced algorithms running on the Edge device provide outputs that, to be proved to be effective and to enhance the battery management process, must be used during tests on the battery pack.

Among the advanced algorithms, the ones related to **(a) thermal runaway**, the ones related to lithium plate detection and the ones related to **(b) fast charging**, need their output to be processed by the sBMS. In Sections: 3.2.2, 3.2.2.2, 3.2.3 the implementation of this function is further discussed.

3.2 Procedures

3.2.1 Multiple steps plan

3.2.1.1 V1 board

The first step is needed to test the HIL testbed available in THIL facilities.

HW	GEN5m
SW	IB1.0
Calib	Gotion Cells

The scope is to have a well-known configuration, tested at the Arbon HiL, that can be used as a benchmark to test the HiL connections.

3.2.1.2 V2 Board

In this step we implement and test the needed SW changes.

HW	GEN5m
SW	IB1.0 with BCU and Edge Communication enabled
Calib	Gotion Cells

The SW modifications performed in this step concern:
Compatibility with WNM
Communication and added functionality regarding EdgeECU
Those modifications are described and summarized, respectively, in 3.2.2 and in 3.2.3.

3.2.1.3 V3 board

This step is needed to comply with the company development timeline. It can be implemented after Step 2.

It will feature Gen5m SW and a modified Gen5m hardware, called Gen5mPLUS.

The main difference between this newer hardware version and the older will be the CAN transceiver, now capable to comply with ISO21434.

HW	GEN5mPLUS
SW	IB1.0, AL migrated to Orpheus the InnoBMS SW architecture, with InnoBMS-specific modifications
Calib	Gotion Cells

Only the application layer will be migrated to the InnoBMS architecture, the SW will then result in a Hybrid architecture described in Figure 3-5.

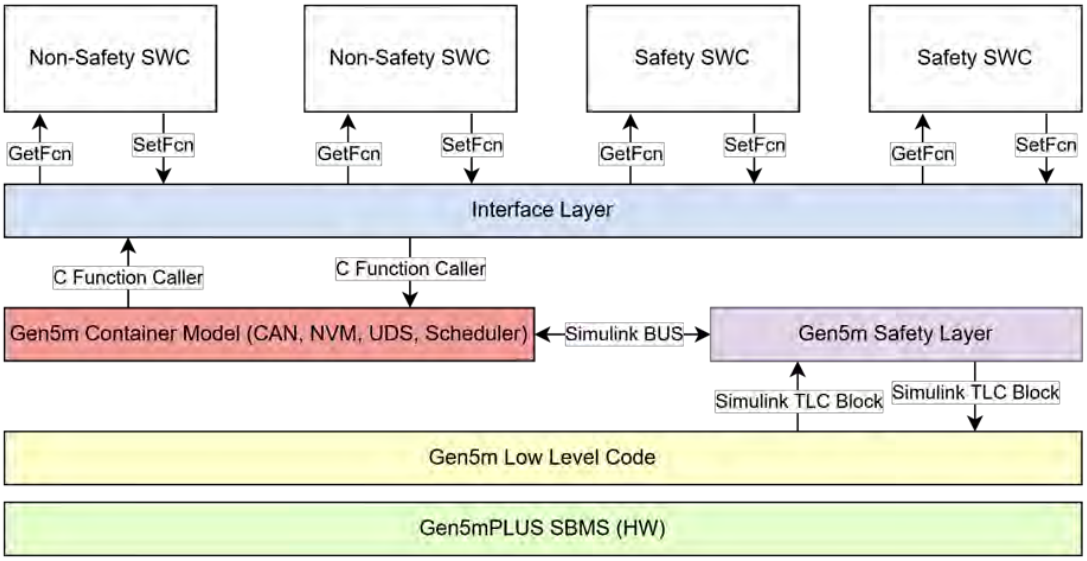

Figure 3-5 : Hybrid SW architecture: AL migrated to InnoBMS architecture and Gen5m SL

3.2.2 EdgeECU Algorithms output

As stated previously, one of the tasks of the SBMS is to let the EdgeECU algorithms control the battery, to prove the effectiveness of the algorithms themselves.

3.2.2.1 Advanced Functions Interfaces Document

To align the interested partners on the shared interfaces between the EdgeECU and the SBMS, a shared document was created. The scope of the document was to list all the advanced functions running on the EdgeECU and collect their Inputs and outputs.

Thanks to this approach it was possible to optimize the information sharing, for example, as shown in 3.2.2.3, multiple units share one CAN interface.

The document can be found here.

3.2.2.2 Advanced Fast Charge Algorithm

The advanced fast charge algorithm running on the EdgeECU requires the sBMS to control the maximum allowed current during fast charge operations. For this purpose, the sBMS SW has been created specifically for InnoBMS, in order to accommodate an external (via CAN) fast charge current signal.

As shown in Figure 3-6, the output of the advanced algorithm, in the form of a maximum allowed current, will be communicated via CAN to the SBMS, which will then use it in place of the internally specified one. The signal will go through the SBMS' safety layer, to always ensure ISO26262 compliant safety.

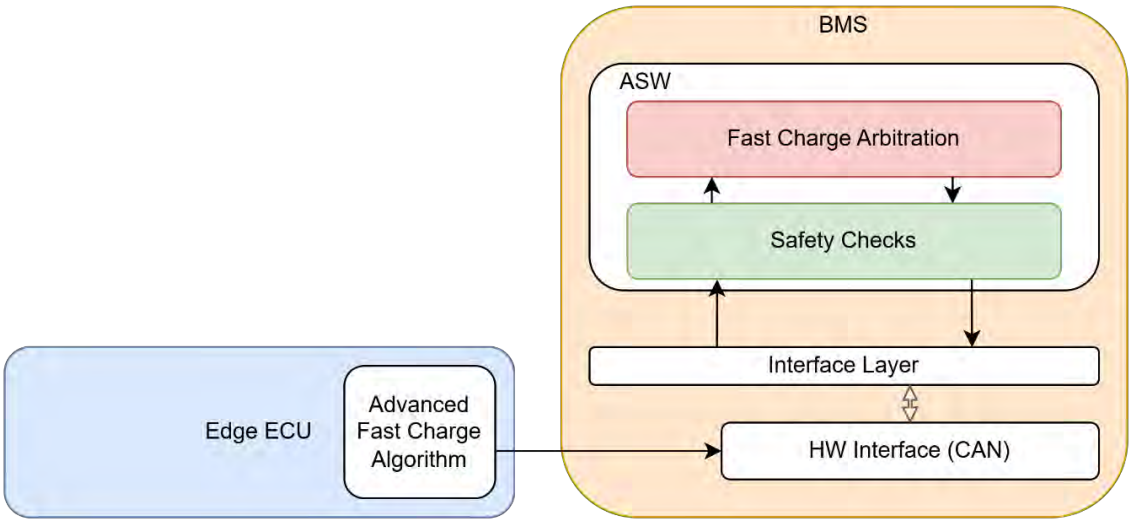

Figure 3-6: Fast charge signal path, from Edge-ECU to sBMS.

3.2.2.3 Emergency Disconnect signals

Multiple advanced algorithms running on EdgeECU, that monitor safety parameters, to effectively operate the battery pack operations, need the SBMS contactors control functionality.
The SBMS Gen5mPLUS can receive an "Emergency Disconnect" signal from CAN and open contactors (Figure 3-7).
In this case the SW of the SBMS was not modified ad hoc, the work done by FMF consisted in sharing this signal arbitration proposal to the other partners, to optimize the use of the CAN communication.

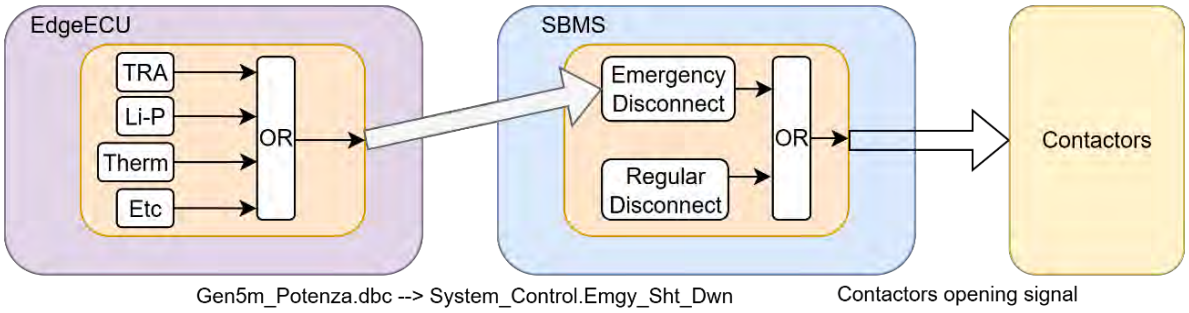

Figure 3-7: Emergency Disconnect signal, EdgeECU internal arbitration

3.2.3 WNM integration

As described in detail in Deliverable D4.3, the CMBs communicate wirelessly to the WNM, which in turn gateways all the relevant data to the SBMS via CAN.
On Gen5m the implementation of the CMC communication starts from the LLSW, the communication is based on the isoSPI protocol.
Specifically, for InnoBMS, the SW has been adapted to read CMB data (cell voltages, temperatures) via CAN.
As shown in Figure 3.2, thanks to the info contained into the DBC file, the CAN SW component in the SBMS is capable of reading data sent by the BCU. The delays related to the BCU choice first, then availability, slowed down work towards completing the SW but did not stop fully the SW development

and test of the SW modifications just described. It was possible to preliminarily test the setup thanks to a vHIL setup, based on CAN communication created by FMF (Figure 3-8).

With a 12V generator, a CAN device and a specifically programmed rapid prototyping tool (Pinnovo ECU, manufactured by DANA), it was possible to emulate the presence of the BCU and the rest of the pack. The SW reacted as expected to the emulated environment, this leads to the assumption that at most a few test loops will be needed to verify the compatibility between the BCU unit and the SBMS.

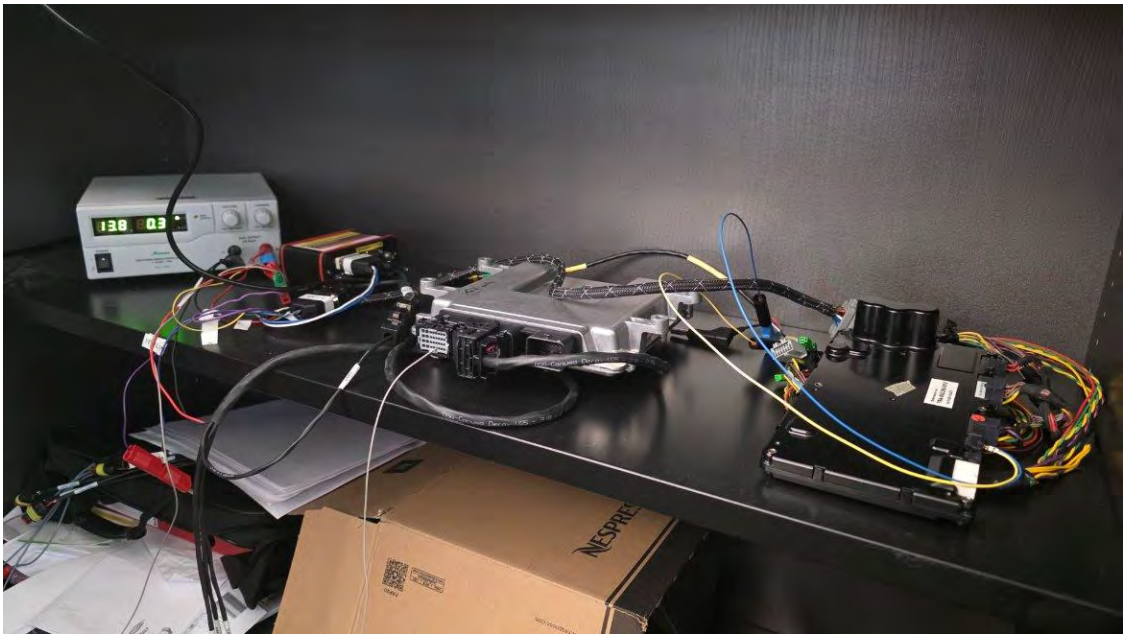

Figure 3-8: vHIL setup: starting from the left: 12V power supply, CAN device, DANA Pinnovo, SBMS.

4 Results & discussion

4.1 Results of Edge-executable SW functions

The validation of the Edge-executable software (SW) functions aims to verify that each Functional Mock-up Unit (FMU) operates correctly after deployment and satisfies its functional and integration requirements. Validation is performed individually for each software component before system-level integration (will be discussed in D3.3) to ensure that the implemented algorithms produce the expected outputs under representative operating conditions.

- The validation process typically includes verification of input and output interfaces, assessment of the functional behaviour against the original model, evaluation under nominal and boundary operating conditions, and confirmation that the FMU executes consistently across the target development environments. Depending on the functionality, dedicated test cases are defined together with acceptance criteria to demonstrate correct operation.
- In addition, selected FMUs are validated within the dSPACE environment to confirm their suitability for real-time execution and integration into the Edge ECU software architecture.

The following subsections present representative validation results for the individual Edge-executable software functions developed by the project partners.

4.1.1 Validation Results of the Reduced-Order Electrochemical Model FMU – UL

The reduced-order battery model was extensively tested already in the development phase, prior to the FMU creation. The tests included validation of different functions that return material properties dependent on e.g. temperature and concentration of Li-ions. Furthermore, the inputs were tested that they are correctly used and applied to the internal structure of the model, outputs were tested for values to be in the expected range and correctly signed.

The model was then parametrised and validated against measured experimental data from the both Li-ion cells used in the InnoBMS project. Applied current profile following the conventional HPPC procedure was used, in which the current pulses within a single HPPC sequence alternate between charging and discharging conditions while their amplitudes are progressively increased. Typically, HPPC measurements are conducted over the entire SoC operating range and repeated at different temperatures to characterise the battery behaviour under various operating conditions (state of charge, current) and thermal states (temperature).

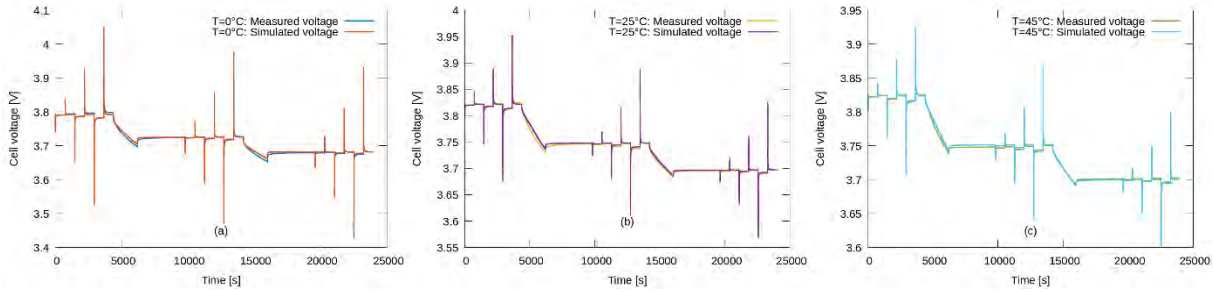

Figure 4-1: Validation of the electrochemical model on the experimental data from the Gotion cell at three different temperatures.

Figure 4-1 presents the validation results of the cloud-based electrochemical model against experimental HPPC measurements performed on the Gotion cell at three different temperatures,

namely 0 °C, 25 °C, and 45 °C. The final RMSD values obtained from the optimisation procedure were around 4 mV for the three temperatures. Furthermore, Figure 4-2 present also the validation results of the electrochemical against experimental HPPC measurements performed on the TOFAS cell again at three different temperatures, namely 0 °C, 25 °C, and 45 °C with similar average RMSD value around 4mV.

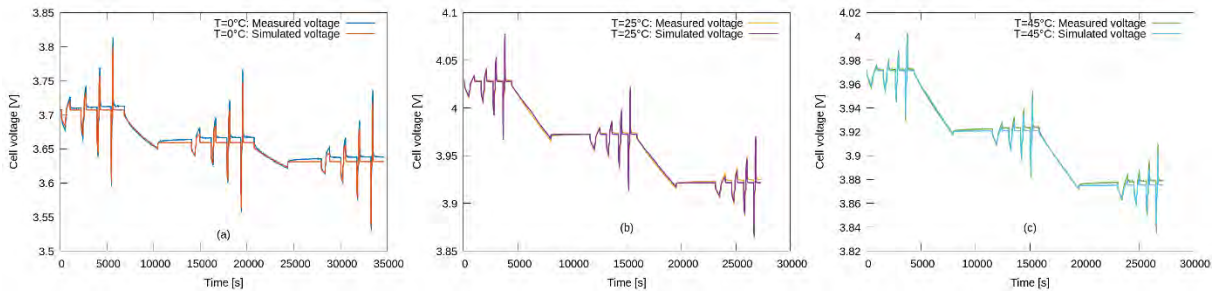

Figure 4-2: Validation of the electrochemical model on the experimental data from the TOFAS cell at three different temperatures.

These results demonstrate a good agreement between the simulated and experimental voltage responses across a wide range of operating temperatures for both cells used in the project. The model was finally exported to the FMU and tested also in the MATLAB Simulink environment with different inputs, i.e. current profiles and temperatures. Passing criteria was that the model behaves identically as the FMU in Windows or Linux operating system as it did during while running the compiled model directly in the PC. Exported FMU is dependent also on some parameters that define battery pack configuration, and these parameters were tested in combination with the inputs to work on a cell level parameter as well as on the pack level parameters. Finally, the re-parametrisation of the model during runtime was tested with changing of the different parameters.

4.1.2 Validation Results of the SoX Edge Estimator FMU – IDIADA

The algorithm has been validated first in Matlab/Simulink. That allows checking all the internal signals and perform accelerated simulations of hours in some seconds. As the SoX estimator requires real time measured variables from the battery operation, some current and voltage profiles are loaded to simulate that the algorithm is receiving the values according to the expected sample time (0.1 s).
Available profiles from HPPC with the Iveco cell have been used for the algorithm validation. The criteria used for the validation are the following:

- Current is defined as positive in discharge.
- Estimated voltage from the model follows the input voltage.
- SoC and voltage increases with charging and decreases during discharging.
- SoC estimation has been compared with short term coulomb counting estimation to ensure that the general trend and the results are similar.
- SoP decreases with SoC as there is a reduction in the available current when the voltage approaches to the minimum limit.
- SoH follows the internal resistance and capacity variations estimated by the algorithm.

As the available information from testing is limited, is it not possible to validate the algorithm performance over long term recordings including a high number of cycles and battery degradation. For this reason, additional simulations have been performed considering wrong initialization of variables to check that the algorithm can properly adjust the estimated values to reduce the mismatch.

- Initial SoC has been incorrectly set in purpose. It has been seen how the algorithm can correct the initial SoC and converge to the real value shown by the reference coulomb counting curve calculated with a correct starting point.
- Battery capacity has been incorrectly set in purpose. The algorithm can recalculate the capacity and consequently the SoH to approach the real value while calculating correctly the rest of indicators.

- Internal resistance has been set lower than the real one to check that the algorithm can track the resistance increase that occurs when the battery ages.

Figure below shows the results of one of the simulations. Graph of the measured and estimated voltage, measured current, estimated and reference SoC, estimated SoP, calculated available current at short and long term and estimated SoH based on capacity and resistance are shown.

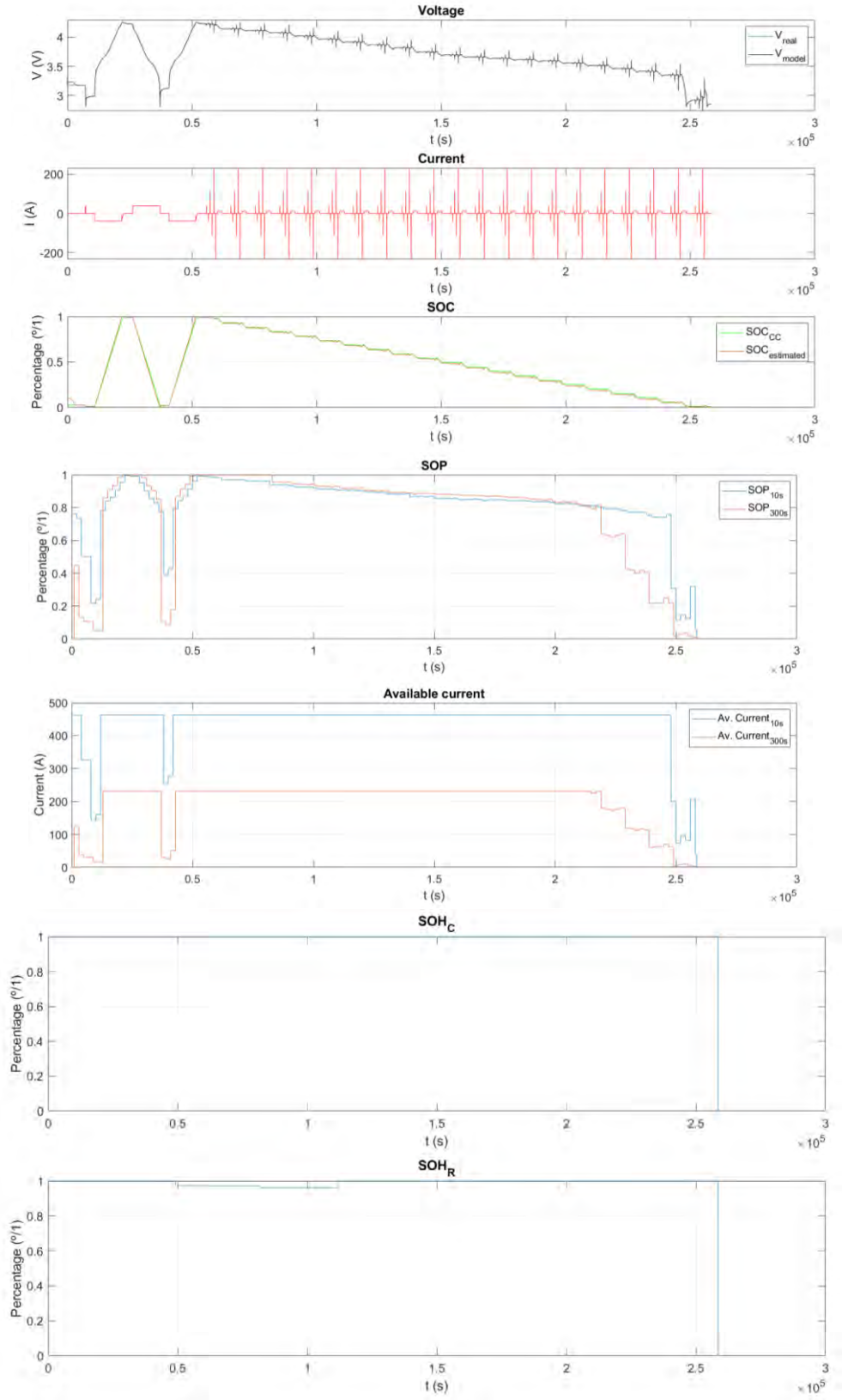

Figure 4-3: Simulation of SoX estimations with HPPC profile

In the first graph, it can be seen how the SoC converges to the reference curve even though it has been incorrectly initialized. The same occurs, with the voltage estimation from the model compared with the real voltage measured.
The estimated SoP is according to the available current, which is reduced when the SoC decreases. As all the parameters have influence on each other, SoP estimation depends on the current SoC and SoH. In this case, the capacity and internal resistance are correctly set up, so there is no relevant SoH variation because the profile duration is not long enough to produce a significant aging in the battery.

4.1.3 Validation Results of the Data Driven Model FMU – AVL

This section explains how the Functional Mock-up Unit (FMU) was validated prior to hardware deployment. The core machine learning model for battery State of Health (SoH) and Remaining Useful Life (RUL) estimation was initially developed and rigorously validated against actual battery data using Python. Because the model's predictive accuracy was already established prior to compilation, this phase of validation focused strictly on software execution.

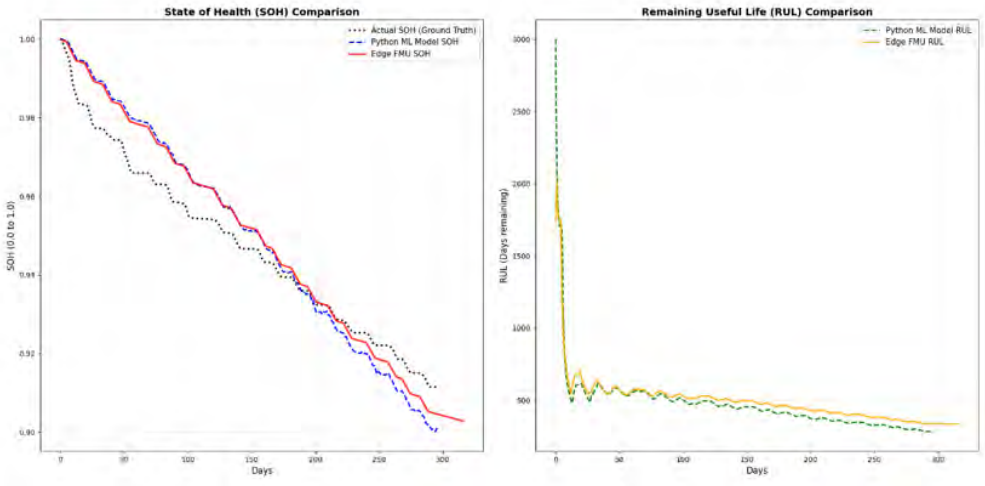

Figure 4-4: 300-Day Comparative Analysis of ML Model vs FMU Results (SoH & RUL)

The primary assumption was that the C-compiled FMU must closely replicate the behavior of the pre-validated Python model. The test objective was to confirm the stability of the stateless C-architecture and the 250ms integrators (I, V, T). To achieve this, a Software-in-the-Loop (SIL) test was conducted using MATLAB and the FMPy Python library to stream approximately 300 days of continuous battery profile data into the FMU. The FMU successfully processed all daily boundaries without memory faults. An extensive 300-day comparative analysis yielded SoH and RUL outputs that were highly consistent with the native ML model, exhibiting only negligible variance and confirming a successful translation from Python to the standalone C-executable

Following this baseline SIL verification, the FMU was deployed to the MicroAutoBox-III(MABX) hardware environment to validate its real-time computational stability, memory retention, and functional logics. The following key scenarios were successfully tested:

- Ageing Cycles and SoH Consistency: To ensure the FMU performs consistently on target edge hardware, an accelerated ageing test compared the MABX execution directly against the MATLAB desktop environment. Both setups were fed the exact same input data (Voltage, Current, Temperature) and simulation conditions over a 12-day period. While the FMU and the inputs were identical, perfectly replicating the exact testing scenarios and execution dynamics between a desktop environment and real-time edge hardware is inherently difficult.

As a result of these environmental differences, a minor variance is expected. The results confirmed that the State of Health (SoH) outputs remained highly consistent, showing a maximum deviation of only 0.0001% between the two systems across the entire simulation window.

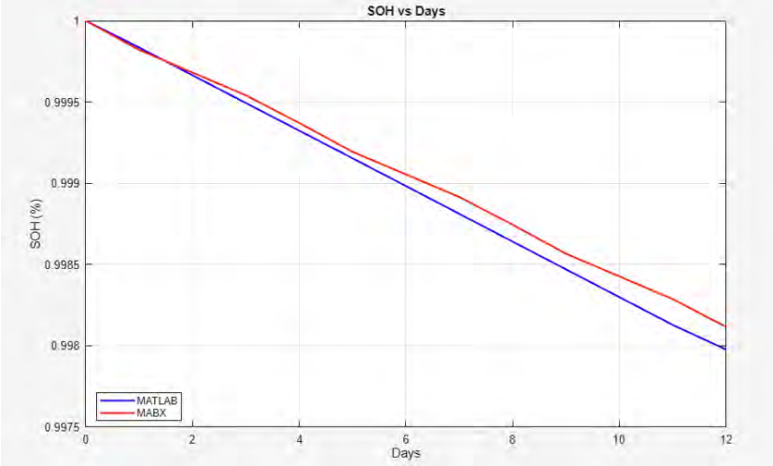

Figure 4-5: SoH Comparison – Edge (MicroAutoBox III) vs Developer Machine (MATLAB)

- State Retention Across Power Cycles (Key-Off Survival): The system's ability to retain historical state variables during vehicle shutdowns was verified. The FMU was run to generate active state variables, which were stored in Non-Volatile Memory (NVM) before the MABX was turned off. Upon the subsequent key-on, the system successfully ingested the stored variables instead of resetting to default values. The model seamlessly resumed from its prior **state (SoH = 0.997957468032836, RUL = 3000, Days = 16)** and correctly updated the current UNIX timestamp to continue the simulation.
- 7-Day Zero-Energy Catch-Up Logic: The model's handling of prolonged vehicle inactivity was tested by advancing the current UNIX timestamp by 15 days past the last recorded key-off timestamp. Because the system is computationally constrained to back-calculate a maximum of 7 consecutive "zero-energy" days, it successfully updated the overall day counter to reflect the full rest period (e.g., jumping from 2 previous days to 17 total days). Simultaneously, it instantly calculated and updated the SoH and RUL to account for exactly 7 days of inactivity, intentionally bypassing the remaining 8 days as designed.
- External Model Weights Update: Finally, the FMU was tested for its capacity to dynamically load external parameters rather than relying solely on its built-in architecture. An array of 5,119 model weights extracted from the ONNX configuration was passed to the FMU externally. Over a one-day simulation, the system processing the external ONNX weights produced an **identical high-precision SoH estimation (0.999861359596252)** as the simulation utilizing the default internal model, verifying that the FMU correctly ingests and applies external weight configurations.

4.1.4 Validation Results of the TRA Early Detection, Lithium Plating Detection, and Edge BTMS Control FMUs – CIDETEC

Three main working modes have been defined. And the validation has been done based on those three main groups:

- Cooling state: The battery cell temperature is higher than the optimum temperature defined by CIDETEC. The BTMS edge control needs to define a lower temperature setpoint.
- Heating state: The battery cell temperature is lower than the optimum temperature defined by CIDETEC. The BTMS edge control needs to define a lower temperature setpoint.
- Maintain state: The battery cell temperature is inside the optimum temperature defined by CIDETEC. The edge BTMS control needs to maintain the temperature of the battery.

In order to validate the BTMS edge control, different temperature profiles have been added as input to the FMU and CID has checked that the control is done based on the three main groups explained above. Validation results is shown below.

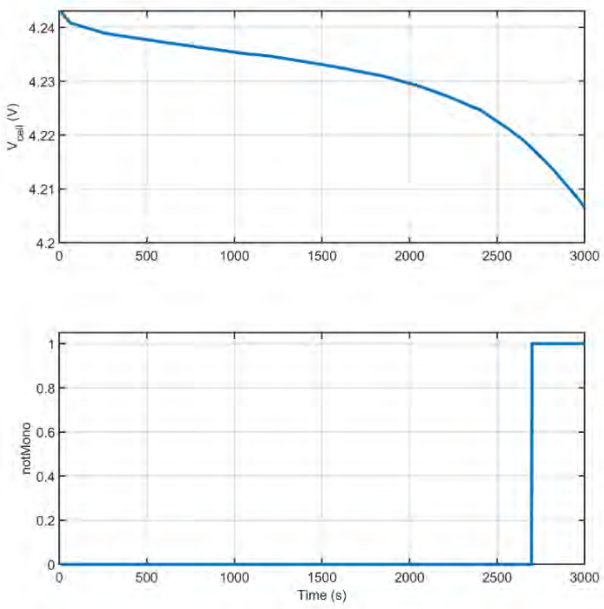

Figure 4-6: Results from the run of the Li-plating detection FMU on the MicroAutoBox III.

4.1.5 Validation Results of the Preconditioning and the Fast Charging and V2G FMUs – VUB

Both FMUs were validated by comparing their compiled FMU outputs with the corresponding Simulink reference models under identical inputs and initial conditions, followed by execution tests on the MicroAutoBox III. Validation results can be seen in Figure 4-7. Validation assumed valid SoC, SoH, RUL, temperature and forecast inputs, correct vehicle parametrization and independent enforcement of hard safety limits by the SBMS and downstream controllers. The normal-operation scenario covered scheduled preconditioning, fast charging, V2G operation for IVECO, and charge-phase temperature control for Tofaș. Test cases included nominal, cold and hot pack conditions, different SoC/SoH/RUL levels, charging and V2G power requests, zero-demand operation, input-limit conditions and communication or invalid-input fallbacks. Passing required numerical agreement with the Simulink model within defined tolerances, correct sign and saturation of current and power references, bounded voltage and temperature references, correct charging/V2G mode transitions, continuous execution without an unintended stop time, deterministic real-time operation and safe fallback outputs when inputs were invalid. Correct FMU behaviour was confirmed when all outputs remained physically plausible, vehicle limits were respected, degradation-aware derating occurred as expected, and no runtime, timing or interface errors were observed.

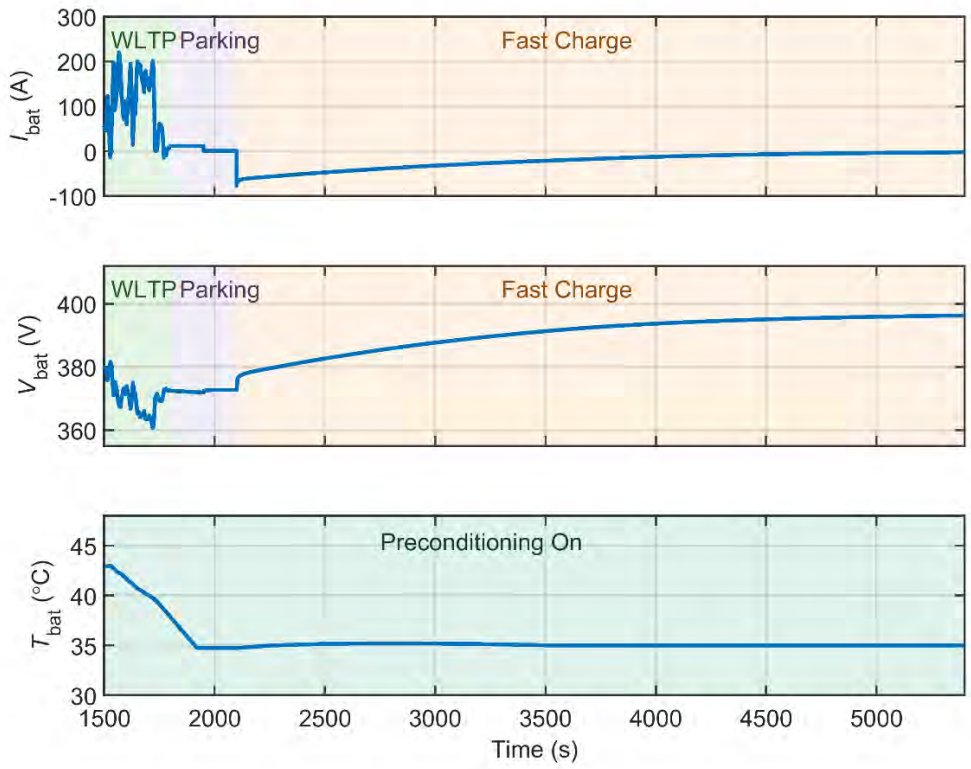

Figure 4-7 : Results from the run of the preconditioning and fast charging and V2G FMUs on the MicroAutoBox III.

4.2 Results of the sBMS SW function

As the current development status is V2 board (see 3.2.1 - Multiple steps plan), only integrated yet with BCU, we can only demonstrate:

- Outcome and results with each component, as:
  - Correct communication of the CMB data on CAN, via to WNM (BCU)
  - Correct reception of the CMB data on CAN by the SBMS
  - Correct interpretation of the data from SBMS

In fact, the SBMS can correctly perform all the initialization steps that lead to a safe contactor closing event (Figure 4-7). During the current project phase, a current cannot be drawn or fed to the battery pack yet, so charge-discharge cycles have not been tested yet.

As the current development status is V2 board (see 3.2.1 - Multiple steps plan), not integrated yet with Edge-ECU, we can only anticipate:

- Outcome and results with each component, as:
  - Correct getaway of the data to the EdgeECU

- Risks and obstacles foreseen
  - Correct communication between SBMS and EdgeECU has not been tested yet, it could fail for:
    - Incorrect CAN communication setting
    - Incorrect DBC file exchange
    - Other communication related incompatibilities

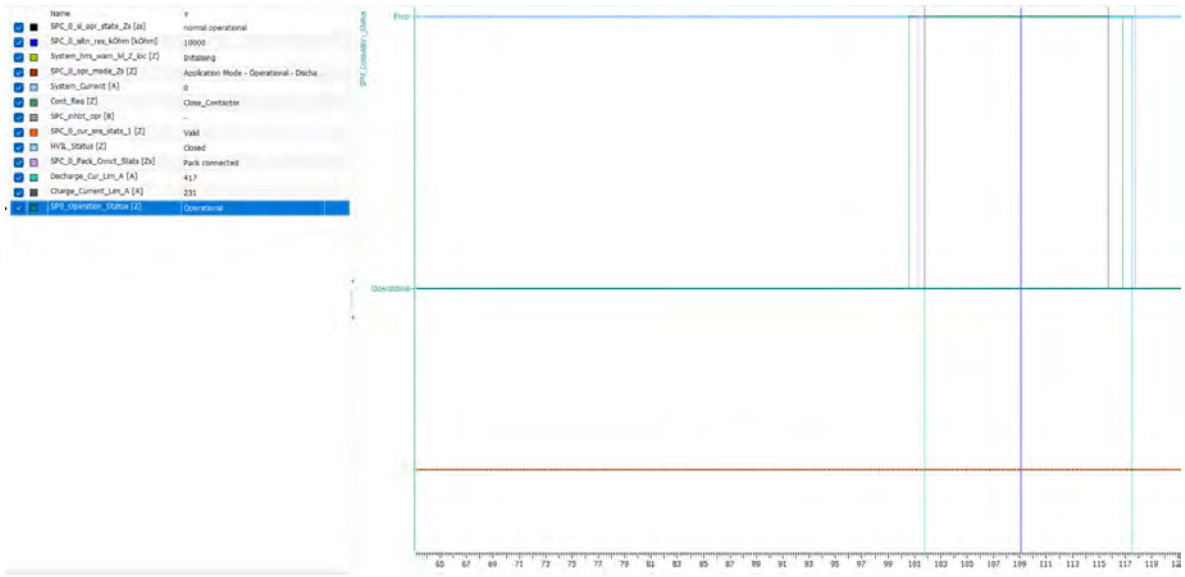

Figure 4-8: Logged safe-startup procedure leading to contactor close. All the safety checks are performed and successful

Until now SW components and SBMS CAN communication has been tested (see 3.2.3).

The next steps are:
V2 board integration of other HW components (CMB, WNM, EDGE)
V3 new board, running final software

4.3 Contribution to project (linked) Objectives

Technical objective 1: Models and algorithms to enable a flexible BMS that is suitable for a range of batteries, and that is fully informed to take balanced decisions for safe and optimal battery use in every driving condition

Outcomes: A novel embedded software architecture has been developed to integrate multiple Edge ECU software functions contributed by different project partners into a unified and executable framework. The architecture establishes standardized interfaces, communication mechanisms, and integration rules, enabling seamless interoperability between independently developed software modules. This common framework significantly reduces integration effort, improves software scalability and maintainability, and ensures predictable deployment of advanced BMS functionalities. By providing a flexible and modular software platform, the architecture supports the realization of a next-generation BMS capable of incorporating diverse models and algorithms for enhanced battery monitoring, diagnostics, control, and optimization across different battery technologies and vehicle platforms.

Technical objective 2: Software for a wireless BMS with reliable and secure connections between all actors in the battery system

Outcomes:

- An efficient and reliable communication architecture of the connected devices, in line with the methodology for functional safety, resulting in SW dedicated to the HW in TO3.
- Ensuring / Providing reliable and secure connections between BMS, CMUs (incl. sensors) and the Cloud, verified and validated through penetration testing of the (cloud) infrastructure and connectivity service
- Information exchange with VCU on ambient temperature, trip info including topography (expected climbing/descending), weather data, powertrain operation, EV Energy management and HVAC.
- Suitable for over-the-cloud software updates and firmware replacements.

5 Conclusion

This deliverable presented the software architecture and integration framework developed for the InnoBMS Edge ECU, providing the foundation for deploying the advanced Battery Management System (BMS) functionalities developed by the project partners.

A common software architecture, standardized interfaces, and a harmonized documentation approach were established to ensure the efficient integration, interoperability, and maintainability of independently developed software components. Furthermore, the deliverable documented the development process of the Edge-executable software models, including their parametrization, Functional Mock-up Unit (FMU) implementation, compilation, and validation methodology. The validation activities confirmed the correct functional behaviour of the individual software modules prior to system-level integration, thereby reducing integration risks and improving the reliability of the overall software framework.

Overall, the developed Edge ECU software framework provides a scalable and modular platform capable of accommodating advanced battery monitoring, diagnostics, estimation, and control functions contributed by multiple partners. This work establishes the basis for the subsequent integration, real-time validation, and demonstration activities within the InnoBMS project, contributing to the realization of a flexible and intelligent next-generation Battery Management System. Moreover, the detailed integration of the Edge ECU SW will be discussed in D3.3.

In parallel, FMF and Potenza Technology has developed the software package based on the requirements defined in Deliverable D1.1. As part of the current development, the V2 board supports the correct transmission of Cell Monitoring Board (CMB) data over the CAN bus to the Wireless Network Manager (WNM/BCU), the correct reception of CMB data by the Smart BMS (SBMS), the correct interpretation of the received data, and the successful forwarding of the processed information to the Edge ECU.

The software development timeline presented in this deliverable is aligned with the planned availability of the hardware components. It also incorporates sufficient time for early integration, debugging, and validation activities, ensuring that any issues identified during system integration can be addressed during the testing phase planned for the latter half of the project.

6 Risks and interconnections

6.1 Risks/problems encountered

Risk No.	What is the risk	Probability of risk occurrence[1]	Effect of risk[1]	Solutions to overcome the risk
D4.2	Delay of WNM selection	1	2	Support and collaboration from other consortium partners who have technical experience on the topics, to push the task forward
D2.2	Delay to delivery of Models from 50% of the consortium partners	2	2	Support and collaboration from other consortium partners who have technical experience on the topics, to push the task forward

[1)] Probability risk will occur: 1 = high, 2 = medium, 3 = Low

6.2 Interconnections with other deliverables

D1.1 Requirements and interface definition
D2.2 Advanced functionalities edge
D4.2 Rapid hardware prototype of the wireless battery module system
D4.3 Advanced pack-level assembling for ultimate battery pack performance

7 Deviations from Annex 1

No major deviations from Annex 1 are reported in terms of planned functions or activities. However, the development of a harmonized and unified software architecture, together with the integration of eight advanced models contributed by different partners into a common edge-executable framework, required more effort and coordination than initially anticipated. To support this process, VUB, with active support from BOSCH, organized 61 Edge Software Integration Sessions, each lasting approximately 3–4 hours, to coordinate software development, interface harmonization, and system integration across the consortium. This additional effort was essential to ensure FMI compliance, interoperability, standardized software interfaces, and the seamless deployment of advanced battery management functionalities on the InnoBMS Edge platform.

Submission of D3.1 was originally scheduled for M28 of the project. However, due to the required additional effort and coordination as described above, the deliverable is submitted with a delay of four months in M32.

8 References

Vasilyev, A., & Altun, E. (2024). D1.1 - Requirements and interface definition. InnoBMS GA No. 101137975. Retrieved from https://cordis.europa.eu/project/id/101137975/results

S. Ozturk, D. Lyall (2025) - D4.1 Concept of scalable cell monitoring, control unit and sensorics layout. InnoBMS GA No. 101137975.

Faltermeier, Markus (2025) - D4.2 Rapid hardware prototype of the wireless battery module system. Located: Situationally aware innovative battery management system for next generation vehicles | InnoBMS | Project | Results | HORIZON | CORDIS | European Commission

D Lyall, E Altun, A Bancroft, A Dilhan, S Ozturk (2026) – D4.3 Advanced pack-level assembling for ultimate battery pack performance. InnoBMS GA No. 101137975. Located: InnoBMS_D4.3_Advanced pack-level assembling for ultimate battery pack performance - SENSITIVE.pdf

9 Acknowledgement

9.1 The consortium

The author(s) would like to thank the partners in the project for their valuable comments on previous drafts and for performing the review.

Project partners:

#	Partner short name	Partner Full Name
1	VUB	Vrije Universiteit Brussel
2	TOFAS	TOFAS Turk Otomobil Fabrikasi Anonim Sirketi
3	BOSCH	Robert Bosch SRL
4	AVL	AVL List GmbH
5	AVL-SFR	AVL Software and Functions Gmbh
6	IDIADA	Idiada Automotive Technology SA
7	CID	Fundacion Cidetec
8	UL	Univerza v Ljubljani
9	THIL	Tajfun Hil Društvo sa Ograničenom Odgovornošću za Istraživanje, Proizvodnju, Rgovinu i Usluge Novi Sad
10	UNR	Uniresearch BV
11	FMF	FPT Motorenforschung AG
12	PTE	Potenza Technology Limited

9.2 Disclaimer/ Acknowledgment

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101137975. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.