

HARDWARE-IN-THE-LOOP VERIFICATION OF A SAFETY-CRITICAL RECONFIGURABLE BATTERY MANAGEMENT SYSTEM FOR AN ELECTRIC AIRCRAFT

Purav Panchal[1], Wolfgang Bliemetsrieder[2], Nina Sorokina[2] & Stephan Myschik[1]

[1]Institute for Aeronautical Engineering, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, Neubiberg, 85579, Bavaria, Germany
[2]Institute for Energy Technology, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, Neubiberg, 85579, Bavaria, Germany

Abstract

This paper presents a hardware-in-the-loop (HIL) simulation and validation framework for a distributed reconfigurable battery management system designed for an electrically powered glider. The battery system follows a modular architecture consisting of a battery main controller (BMC) and nine battery cell controllers (BCCs), enabling scalable energy storage for electric aviation applications. The presented system comprises nine battery modules, each integrating twelve battery cells and a dedicated BCC. To enable realistic system-level verification, a modular HIL testbench is developed using a Speedgoat Performance real-time target equipped with battery cell emulation and fault insertion hardware, with temperature sensor emulation provisioned for future integration. The testbench is extended with a TechSAT ADS2 real-time computer executing a high-fidelity flight dynamics model, allowing closed-loop interaction between aircraft dynamics and the battery system. A model-based software development approach is implemented to develop and verify the battery controller software in accordance with the RTCA DO-178C and DO-331 standards. The paper focuses on the integration of distributed embedded controllers into the real-time HIL environment and demonstrates the effectiveness of the testbench for functional verification, fault response assessment, and system-level validation of a safety-critical battery management system for electric aircraft.

Keywords: electric aviation, reconfigurable battery system, safety-critical software, hardware-in-the-loop testing

1. Introduction

The electrification of aviation has gained significant momentum in recent years as a means to reduce emissions, noise, and operational costs, particularly for light aircraft and gliders [1]. Central to this transition is the reliable and safe operation of high-energy battery systems, which must function correctly under a wide range of operational and environmental conditions. The electric glider considered in this project has a maximum take-off weight of approximately 820 kg, an 80 kW propulsion system based on an EMRAX 268 motor with a clipped four-blade propeller, a wingspan of 20 m, and a maximum glide ratio of 43. In electric aircraft, battery management systems (BMS) play a safety-critical role by ensuring energy availability, protecting battery cells from hazardous conditions, and interacting with aircraft systems during both nominal and fault scenarios [2]. While such architectures improve scalability and fault tolerance, they also increase system complexity and pose significant challenges for verification and validation. Simulation-based verification, performed through model-in-the-loop and software-in-the-loop testing, increases the initial confidence in the software, enables rigorous and reproducible testing, and reduces development cost and time. It is therefore well suited to the early stages of software development and to rapid prototyping. However, simulation does not reflect

the actual behavior of the software on the embedded target, nor its interaction with the surrounding physical systems, and assumptions made during simulation remain unverified.
For a distributed battery management system, this gap between simulation and target execution is particularly relevant. The controller behavior depends not only on the functional algorithms, but also on communication timing, CAN message scheduling, update flags, signal scaling, quantization, and the interaction between multiple embedded controllers. These effects are difficult to represent completely in model-in-the-loop testing, but they can influence the observed system response and fault handling behavior. HIL testing therefore provides an intermediate verification step between pure simulation and flight testing, allowing the generated software and embedded hardware to be exercised under deterministic, repeatable, and safety-controlled conditions.
Real-time verification addresses these limitations by confirming that the software meets its real-time requirements and by validating the assumptions made during simulation. Verifying the software directly in its final embedded hardware together with the real physical environment is, however, often costly and potentially unsafe, while the deterministic and time-constrained nature of real-time systems makes such testing complex and expensive.
HIL simulation provides an effective means to address these challenges by enabling real-time interaction between embedded controllers and high-fidelity plant models. In the context of safety-critical battery systems, HIL testing is especially valuable because cell voltages, temperatures, communication losses, and fault conditions can be injected repeatedly without exposing the aircraft or hardware under test to unsafe operating conditions. By integrating actual target hardware with real-time battery and flight dynamics emulation, HIL testing allows the assessment of system-level behavior under realistic operational and fault conditions prior to flight testing. This paper presents a HIL verification framework for a safety-critical reconfigurable battery management system designed for an electrically powered glider. The proposed setup integrates a distributed battery controller architecture with real-time battery cell, temperature, and fault emulation, allowing the integrated battery system response to be evaluated under nominal and faulted operating conditions.
The main contributions are: (i) the migration of a representative distributed battery controller configuration, comprising the BMC and one of the nine BCCs, onto embedded target hardware; (ii) the integration of battery cell and fault emulation into the real-time testbench; and (iii) an automated workflow that derives the real-time test setup from the verified development models, enabling repeatable and traceable HIL verification and closed-loop fault injection during a simulated flight.
The remainder of this paper is structured as follows: Section 2. discusses the reconfigurable battery management system, while Section 3. explains the model-based approach to software development. Section 4. introduces the HIL testbench and Section 5. describes the automated workflow of generating real-time executables for the HIL tests. Section 6. presents the results of the HIL tests performed on the battery controller software, at component level and in closed loop with the aircraft model. Finally, the paper concludes with future research directions in Section 7..

2. Reconfigurable Battery System

Reconfigurable battery systems are being investigated for automotive traction batteries, stationary energy storage systems, and, more recently, electric aviation applications [3, 4, 5, 6]. Reconfigurable systems are characterized by distributed intelligence and individual battery cell monitoring. Such battery packs consist of a single battery main controller (BMC) and multiple battery cell controllers (BCCs), as shown in Fig. 1 [7, 8]. A key feature of reconfigurable systems is the ability to dynamically reconfigure the overall battery pack configuration depending on the power demand and the current state of the battery cells.
The BMC communicates with the engine controller, cockpit systems, and other aircraft subsystems, and primarily with the BCCs. The responsibilities of the battery controllers include balancing the battery cells and monitoring their voltage, current, and temperature. This reconfiguration is enabled by MOSFETs integrated into each battery controller submodule. Depending on the applied switching and interconnection strategy, parallel switching of neighboring cells can be realized, as demonstrated by the Battery Modular Multilevel Management (BM3) system described in [9]. Among the proposed approaches, the cascaded half-bridge topology has been selected for the project. This approach

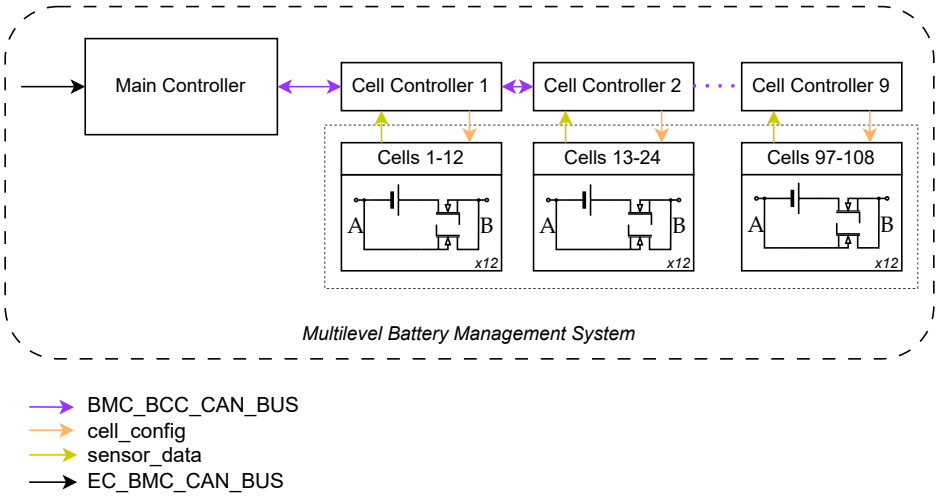


Figure 1 – Architecture of the reconfigurable battery management system including BMC and BCCs.

enables configurable interconnection of battery cells, allowing the system to adapt the output voltage to varying load demands, improve overall efficiency, extend battery lifetime, and enhance fault tolerance. Each module can operate either in a series-connected mode or in a bypass mode, in which faulty or degraded cells are excluded from the active configuration. This supports both safety and longevity objectives and is achieved through dynamic reconfiguration of the battery modules during operation. The reconfigurable BMS considered in this work consists of a central BMC and nine BCCs, each managing twelve battery cells. The labels in Fig. 1 indicate the main communication and signal interfaces. The `BMC_BCC_CAN_BUS` connects the BMC and BCC modules; the BMC sends system commands, voltage commands, and error responses, while the BCCs return status, system-response, voltage-response, and error-status messages. The `cell_config` signal defines the MOSFET switching configuration used to connect or bypass individual cells. The `sensor_data` interface contains the measured cell voltages and temperatures. The `EC_BMC_CAN_BUS` CAN bus connects the engine controller and the BMC, carrying the requested battery voltage from the engine controller and the configured voltage response from the BMC. Based on this information, the BMC determines the required interconnection state of the battery modules and distributes a configuration vector defining the operating mode of each module. Further details on the development of the BCC software are presented in [10].

3. Model-Based Software Development

The software development process follows the industry standard V-Model, which improves software safety by allowing verification and validation activities throughout all development phases. There are certain software standards that must be adhered to in order to receive the certification. In the case of the aerospace industry, the software is regulated by the guidelines developed by the Radio Technical Commission for Aeronautics (RTCA), such as DO-178C, Software Considerations in Airborne Systems and Equipment Certification [11] along with its supplement, DO-331 (Model-Based Development and Verification Supplement to DO-178C and DO-278A), which provides additional objectives when the model-based design approach is employed [12]. DO-178C defines a structured software life cycle covering planning, development, requirements definition, design, coding, integration, verification, configuration management, and quality assurance. Although the standard defines the required objectives, it does not prescribe a specific development method or toolchain; developers may select their own approach provided that the applicable DO-178C objectives are satisfied. Model-based design supports the development of safety-critical software by representing system functionality through mathematical and graphical models. This approach improves readability, supports automated code generation, enables efficient functional testing, strengthens traceability, and simplifies review and analysis activities. One of its key advantages is the automatic generation of

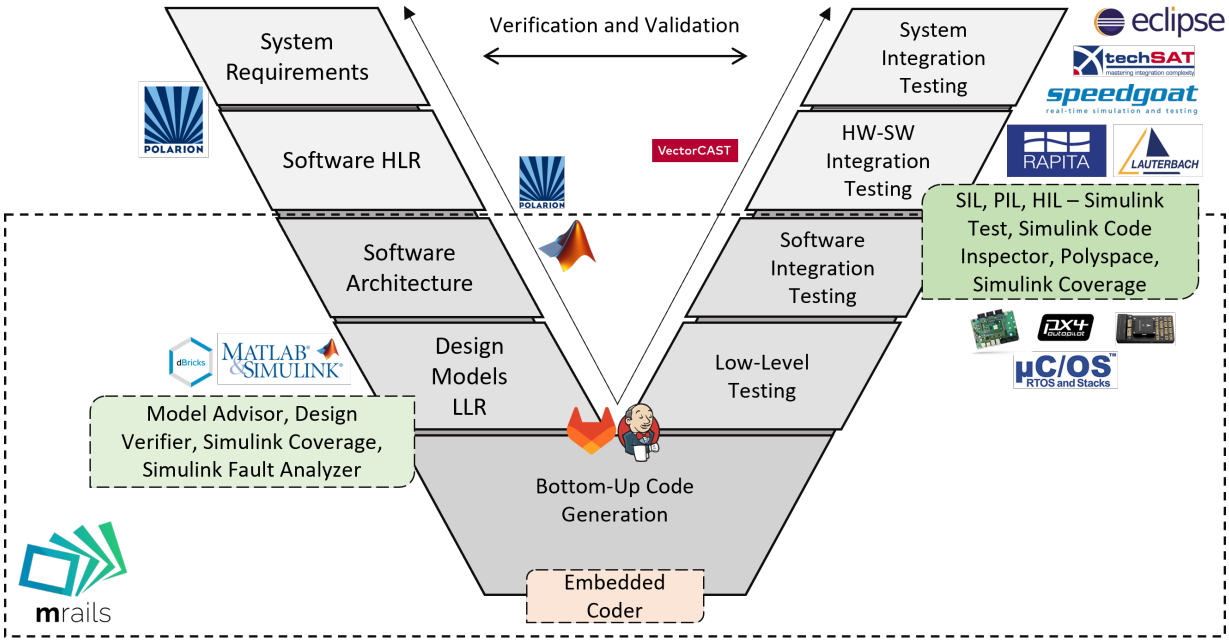


Figure 2 – Model-based software development V-model along with the necessary tools required during different stages of development and testing.

source code from verified design models, which supports back-to-back testing between model and code and helps establish bidirectional traceability, both of which are important in a DO-178C context. In addition, model-based development can reduce the impact of late requirement changes by avoiding a “big-freeze” situation, where modifications introduced late in the development cycle become costly and time-consuming. When combined with a model-based development approach, as shown in Fig. 2, the V-Model allows design models to serve as executable requirements, supports automated code generation and verification, and improves collaboration in distributed development environments [13].

In the present approach, the software is divided into two components: functional software, which implements the control logic, and the application framework, which contains interfaces and low-level driver code. Functional software is developed using MathWorks MATLAB/Simulink, where low-level requirements are realized as design models and software architecture is defined using MathWorks System Composer. A process-oriented build tool, developed at the Institute for Aeronautical Engineering of the University of the Bundeswehr Munich and the Institute of Flight System Dynamics of the Technical University of Munich, supports modular model-based development, automated code generation, and verification in accordance with DO-331 modeling guidelines [14, 15]. System and software requirements, test cases, and development artifacts are managed using Siemens Polarion PLM [16]. Application software requirements are specified textually, and the corresponding code is manually implemented. Verification activities include static analysis using Polyspace [17], simulation-based testing with Simulink Test, and unit testing using VectorCAST [18]. Timing analysis is performed using Rapita Verification Suite [19] to ensure that the timing requirements are met. The model-based approach enables systematic model-in-the-loop (MIL), software-in-the-loop (SIL), processor-in-the-loop (PIL), and hardware-in-the-loop (HIL) testing. After integration of the functional software into the application framework, real-time verification is performed on an HIL testbench. Interfaces between software and hardware components are managed using the dBricks interface management tool [20]. A transport-layer code generator is used to produce the source code responsible for handling communication messages and flags. For example, it processes the XML export from dBricks, which contains the message structure, and generates code to decode input frames, encode output messages, and manage associated flags [21]. Traceability across all development artifacts is ensured using SimPol [22], Polarion-Simulink connector, version control with Git, and an automated continuous integration framework based on Jenkins [23], enabling round-trip requirements verification.

4. Hardware-in-the-Loop Testbench

HIL simulation is a real-time testing approach in which the embedded controller hardware is connected to simulated plant models before deployment on the aircraft. This allows the controller software to be tested with realistic inputs and communication interfaces, while keeping the test environment repeatable and safe. The HIL test rig architecture diagram is shown in Fig. 3 and Fig. 4 shows its physical counterpart. The setup is split across two real-time computers. The flight dynamics model of the electric aircraft, together with a supervisory flight controller for automated mission execution, runs on *RTPC1* [24]. This computer is a TechSAT ADS2 system and is used for the aircraft-level simulation and cockpit/visualization interface [25]. The battery system, power electronics, motor model, and controller interfaces run on *RTPC2*, a Speedgoat Performance real-time target [26].
The separation between *RTPC1* and *RTPC2* allows the aircraft model and the electrical powertrain model to be executed on hardware suited to their respective timing requirements. The TechSAT ADS2 system is suitable for flight-dynamics simulation with frame rates up to 2 kHz, while the Speedgoat target is designed for fast electrical system simulation and provides FPGA-based I/O for high-frequency signals. The Speedgoat computer used in this work includes an I/O-334-325k Simulink-programmable FPGA I/O module with analog input and output capability. This enables the powertrain and battery models to exchange signals with the embedded controllers in real time.

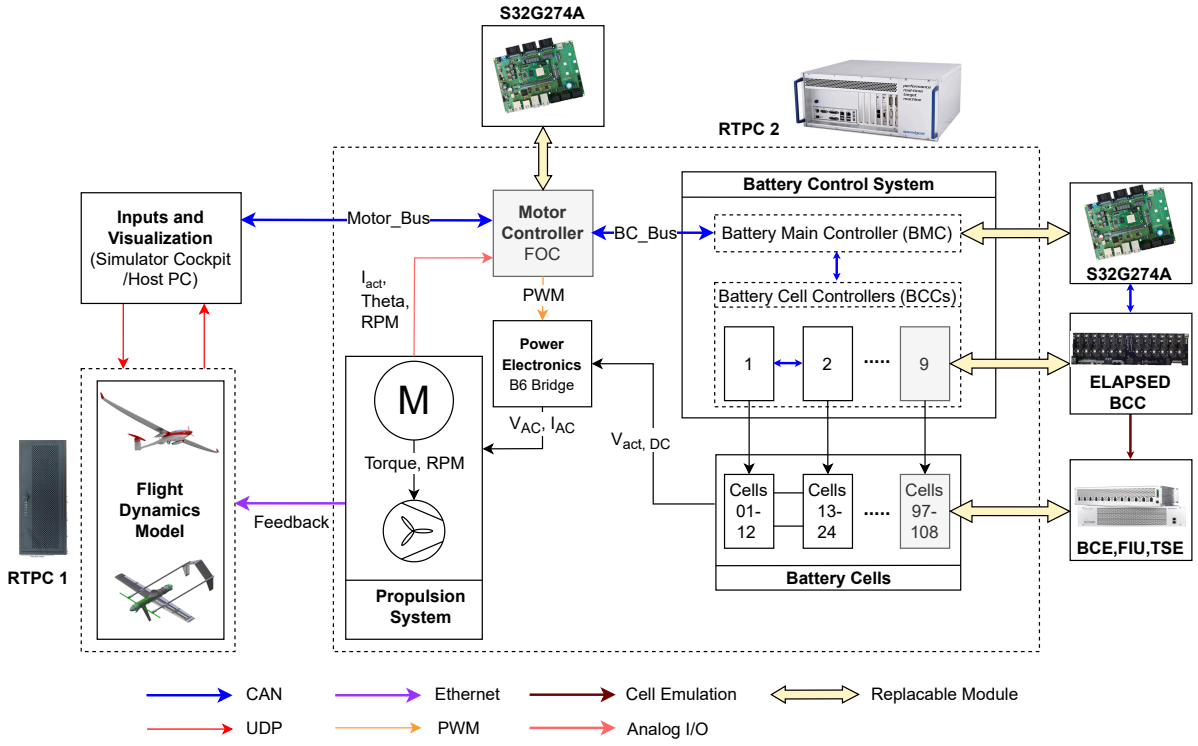


Figure 3 – Hardware-in-the-loop test architecture with the two real-time computers, the embedded battery controllers, and the battery, temperature, and fault emulation hardware.

On the Speedgoat side, the electrical powertrain is represented by models of the power electronics, motor, and battery system. The battery cells are emulated using Speedgoat Battery Cell Emulation (BCE) hardware. Temperature sensor inputs are generated using Temperature Sensor Emulation (TSE), and cell-level faults are introduced using a Fault Insertion Unit (FIU) [27]. The BCE and FIU support twelve cell channels, while the TSE supports up to four temperature sensor channels. This makes it possible to test nominal operation as well as fault cases such as open-cell connections or invalid sensor readings without using real battery cells. Temperature sensor emulation is not used in the current setup because the BCC PCB does not provide the required measurement interface for the resistance values generated by the TSE. Two parts of the simulated battery control system are replaced by embedded hardware. First, the Battery Main Controller (BMC) is executed on a

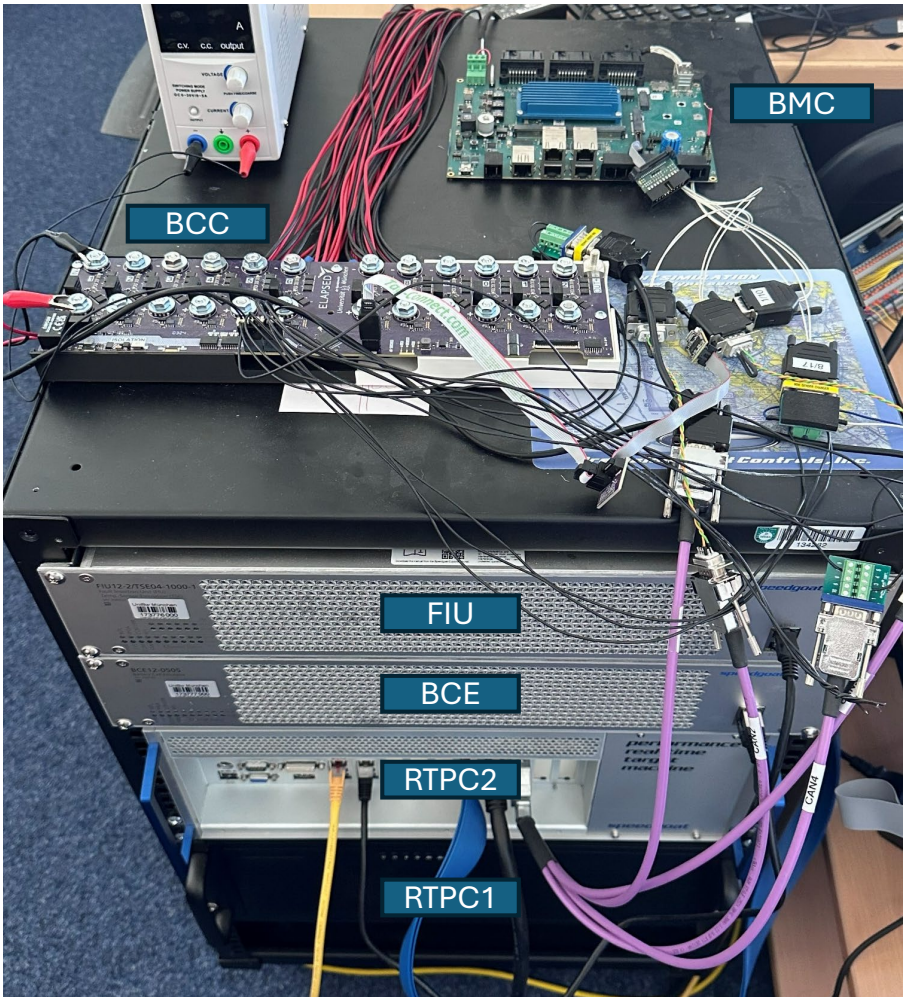
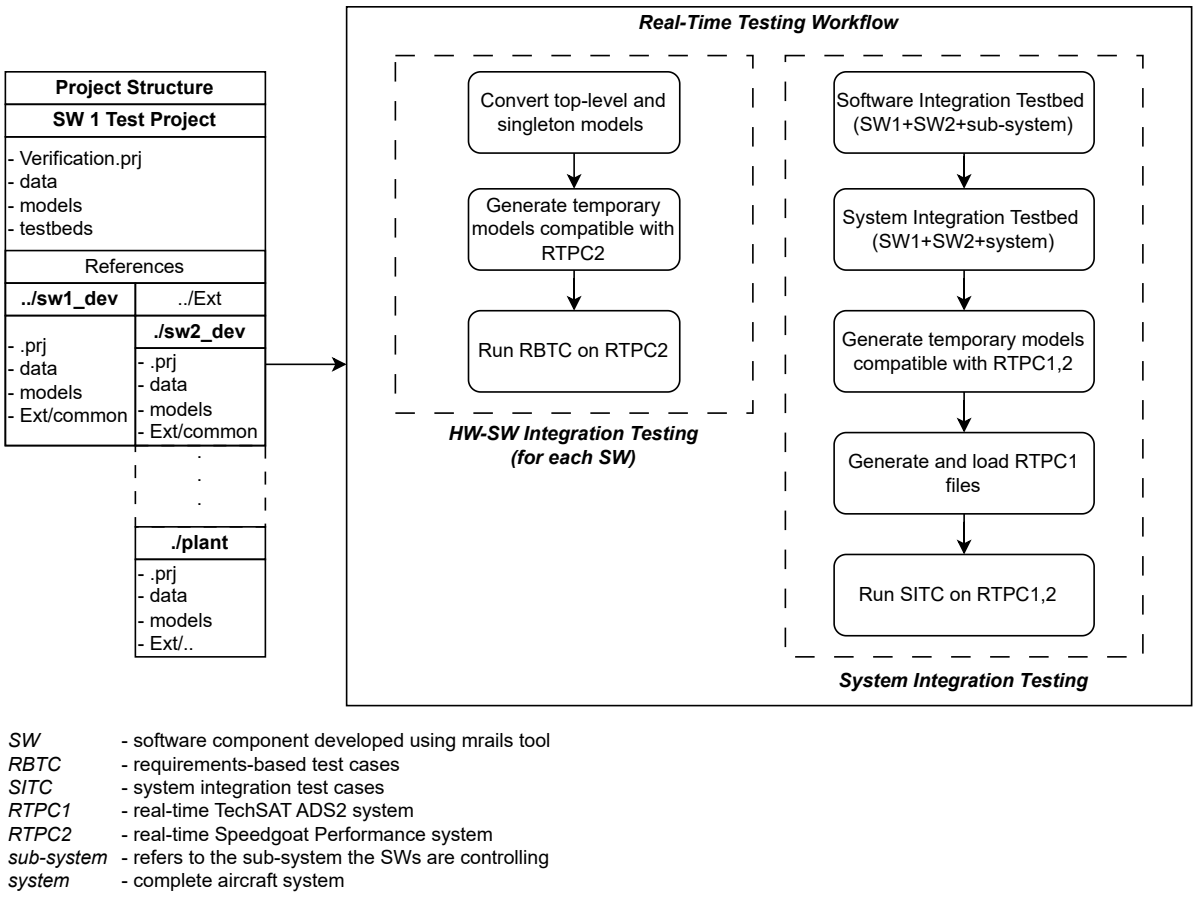


Figure 4 – Physical realization of the HIL testbench: the Battery Main Controller (BMC, MicroSys S32G274A [28]) and Battery Cell Controller (BCC, custom STM32G491 PCB) on embedded hardware, communicating over CAN with the Speedgoat Performance real-time machine (RTPC2) [26] hosting the Battery Cell Emulator (BCE) and Fault Insertion Unit (FIU) [27]; the TechSAT ADS2 system (RTPC1) [25] runs the flight dynamics model.

MicroSys development board based on the NXP S32G274A processor. The processor contains three Arm Cortex-A53 cores and four Cortex-M7 cores. The BMC application runs on the first A53 core under a custom certifiable real-time operating system based on μC/OS-II. The integration of an earlier version of the BMC software into this operating system is described in [10].
Second, one battery module is replaced by the physical Battery Cell Controller (BCC) hardware. The BCC software runs bare-metal on an STM32G491 microcontroller mounted on a custom in-house PCB. This BCC is connected to the Speedgoat BCE and FIU hardware, so that its cell-voltage and fault inputs are provided by the HIL testbench. The remaining battery modules are represented by simulation models on *RTPC2*. This mixed setup allows the interaction between the BMC, the physical BCC, the emulated cells, and the aircraft-level simulation to be tested in closed loop.
The complete testbench therefore supports several verification levels. At component level, the BMC and BCC can be tested against an emulated battery pack with controlled voltage commands and injected cell faults. At aircraft level, the same setup can be connected to the flight dynamics model, allowing the battery system response to be evaluated during an automated mission. This makes it possible to observe not only whether the controller detects and reacts to faults, but also how the response affects system-level quantities such as configured pack voltage, aircraft altitude, and battery state of charge.

5. Automated Real-Time Test Workflow

Migrating a functional Simulink design model from development to the real-time testbench requires changes that must not modify the verified design models. Configuration sets have to be switched to the real-time targets, model references re-pointed, and dedicated interface models built for each real-time computer. Performed by hand, these steps are error-prone and break traceability. To avoid this, a set of automation scripts derives the real-time test setup from the unmodified design models. The functional software is developed as a set of modular Mrails projects, each exporting top-level, reusable, and singleton models that are configured for code generation and simulation rather than for real-time execution. The first automation step copies each module's top-level models into the test project and adapts them for integration. Models that depend on singleton models, which cannot be referenced more than once, have their model-reference blocks converted into atomic subsystems, and the development configuration set is replaced by a reusable configuration so that the model can be embedded as a reference within the testbed. The top-level configuration is similarly adapted in the same step. The original Simulink blocks remain untouched. The testbed is then assembled from a single top-level integration model whose plant components and controllers are interchangeable between simulation and hardware through variant subsystems. When a real-time run is requested, disposable copies of the integration model and all of its references are generated in a working directory, and the appropriate real-time configuration set, which differs between the Speedgoat target *RTPC2* and the TechSAT ADS2 target *RTPC1*, is activated. All references are re-pointed to these temporary copies, so the build operates entirely on throwaway artifacts and the verified models are never modified.



SW - software component developed using mrails tool
RBTC - requirements-based test cases
SITC - system integration test cases
RTPC1 - real-time TechSAT ADS2 system
RTPC2 - real-time Speedgoat Performance system
sub-system - refers to the sub-system the SWs are controlling
system - complete aircraft system

Figure 5 – Automated real-time test workflow. The verified development modules referenced by the test project are processed without modification to derive two real-time test configurations: a component-level HIL test of the controllers on the *RTPC2*, and a system-level closed-loop test that additionally couples the flight dynamics model running on the *RTPC1*.

For the flight dynamics model executing on *RTPC1*, the workflow additionally generates the ADS2 files required by the TechSAT toolchain and a corresponding Simulink interface model containing the UDP send and receive blocks, together with an input/output mapping that binds the Simulink bus signals to the UDP messages exchanged between the two real-time computers. This interface model is inserted as a model reference into the testbed running on *RTPC2*. In this step, the ADS2 files consisting of the source code, component file, I/O map and session file are deployed automatically on the *RTPC1* via SSH.
A single command orchestrates the process: it optionally regenerates the ADS2 interface, produces the Speedgoat temporary models, removes stale generated code, builds the real-time executable, and deploys and starts the application on the target, optionally launching Microsoft Flight Simulator for visualization. The complete closed-loop setup, with both controllers on embedded hardware, the emulated battery and powertrain on the Speedgoat, and the flight dynamics on the TechSAT machine, is thereby derived automatically from the verified development models. This preserves traceability and makes the incremental migration of components from simulation to hardware repeatable. The Speedgoat communication model, which handles CAN, CPU, and FPGA interfaces, is built manually because it depends on the specific test objectives. The FPGA module requires a bitstream to be generated using the HDL Workflow Advisor. This workflow is discussed in [29]. Figure 5 summarizes the workflow.

6. Results

The HIL testbench is used to verify the battery controller software across three levels of integration: unit level (individual controller functions), component level (the battery pack), and aircraft level (closed-loop flight). A central benefit of the model-based, requirements-driven process is that the same requirements-based test cases used during model-in-the-loop (MIL) verification are reused without modification at the hardware level, providing direct evidence of behavioral equivalence between the design model and the embedded target. A prior study [29] established baseline aircraft-level behavior with the BMC and motor controller executed in simulation and the battery supplied by a constant power source; the present work extends this with the BMC and one representative BCC on embedded hardware, real-time cell emulation and in-flight fault injection.

6.1 Unit-Level Verification: Test-Case Reuse from MIL to HIL

At unit level, the requirements-based test cases authored for the model-in-the-loop verification of the BCC functional software are executed, unchanged, against the software running on the embedded target. This reuse provides numerical and behavioral equivalence evidence between the design model and the target implementation, supporting the evidence needed for model-to-target verification under a DO-178C/DO-331 process. Figure 6 illustrates this for the BCC system-command (SYS) response test: a system command drives the BCC through its operating states (Wake-Up, Ramp-up, Emergency), and the test asserts that the decoded SYS response matches the expected state for each commanded transition. The same assessment statements pass for both the MIL and the HIL execution, confirming that the embedded implementation reproduces the verified model behavior.
The only difference between the two executions is a fixed time offset of approximately $0.1\,\mathrm{s}$ in the HIL response. This latency is introduced by the transport-layer code on the target, which transmits the CAN response messages at a fixed refresh rate; it has no counterpart in the MIL configuration, where the signals are exchanged directly between models. The test tolerances accommodate this deterministic transport latency, so the requirements-based assessments pass at both levels.

6.2 Component-Level Verification: Distributed Control and Fault Response

At component level, the distributed control chain consisting of the BMC and one representative BCC, both running on embedded hardware, is verified against the emulated battery pack on the Speedgoat target, communicating over CAN. The engine controller commands a pack voltage profile, and the BMC reconfigures the battery modules to realize it by computing the required series/bypass configuration from the cell voltages reported by the BCC and distributing the corresponding commands. This command is then received by the BCC and provides the requested voltage by switching its cells.

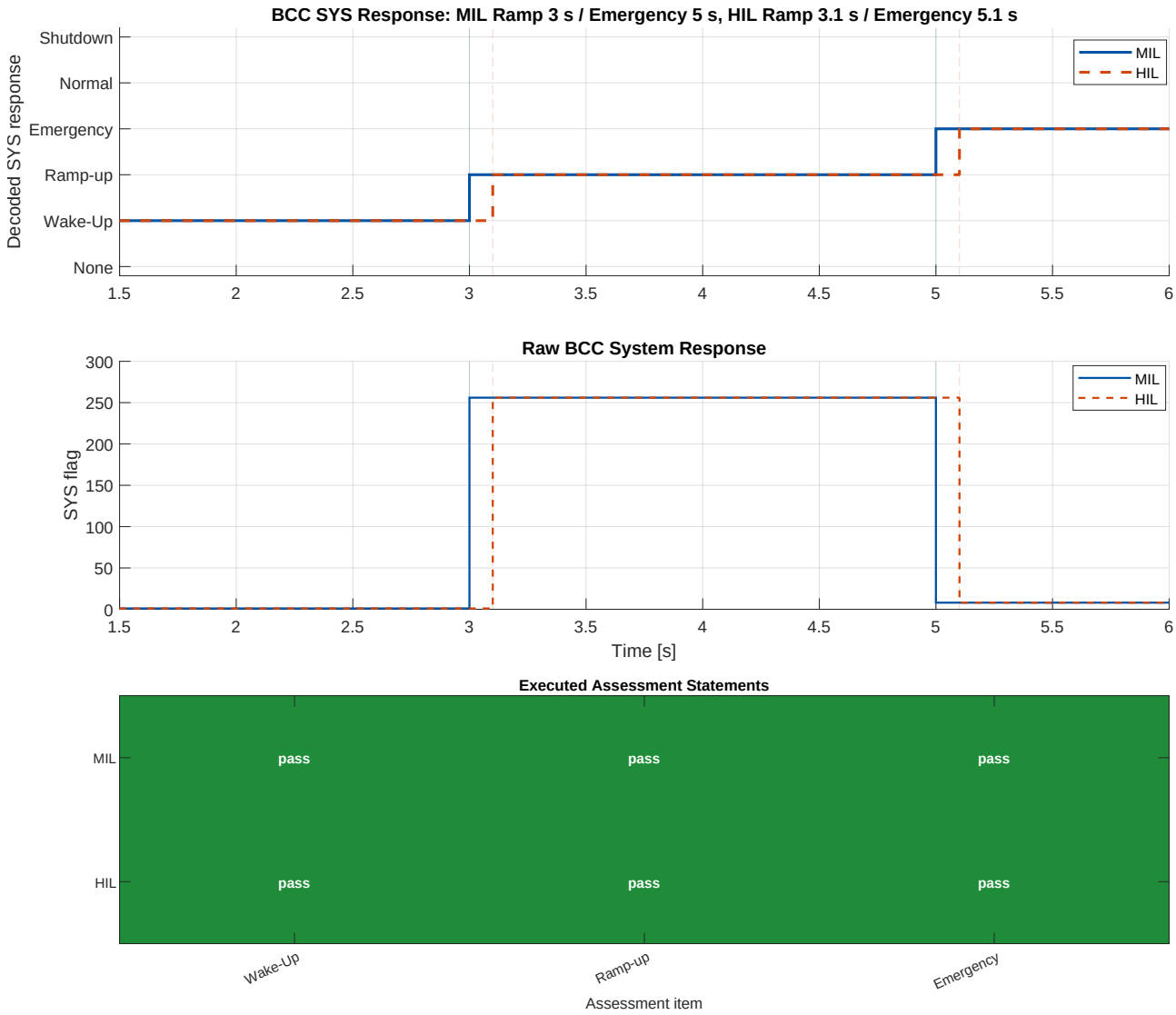


Figure 6 – BCC system-command (SYS) response verified with the same requirements-based test case in MIL and HIL. The decoded response follows the commanded state transitions and all assessment statements pass at both levels; the HIL trace is offset by the deterministic CAN transport-layer refresh latency.

Figure 7 compares two runs of this setup driven by the same voltage command, one nominal and one with a cell fault injected.

In the nominal run, the configured pack voltage follows the engine command throughout the profile, including the commanded step-downs, confirming correct cooperative operation of the two controllers on the target hardware; over the plotted interval the maximum tracking deviation is approximately $12\,\mathrm{V}$, which is consistent with the quantized cell-configuration resolution. In the fault-injection run, an open-circuit fault is injected on cell 1 of BCC 1, breaking the cell connection so that the BCC measures zero volts on that cell during three intervals (middle and bottom plots). In response to this open-circuit reading, the BMC enters its Limp mode and commands the maximum output voltage to the BCCs; the configured pack voltage therefore departs from the engine command and saturates at its maximum for the duration of each fault, returning to correct tracking as soon as the fault is cleared. Repeating the fault across three intervals confirms that the testbench can inject and clear cell-level faults deterministically while the controllers run on their target hardware. The detection of the open-circuit cell and the resulting transition to Limp mode correspond to the fault conditions defined in the subsystem design specification, demonstrating requirements-traceable fault handling on the integrated hardware.

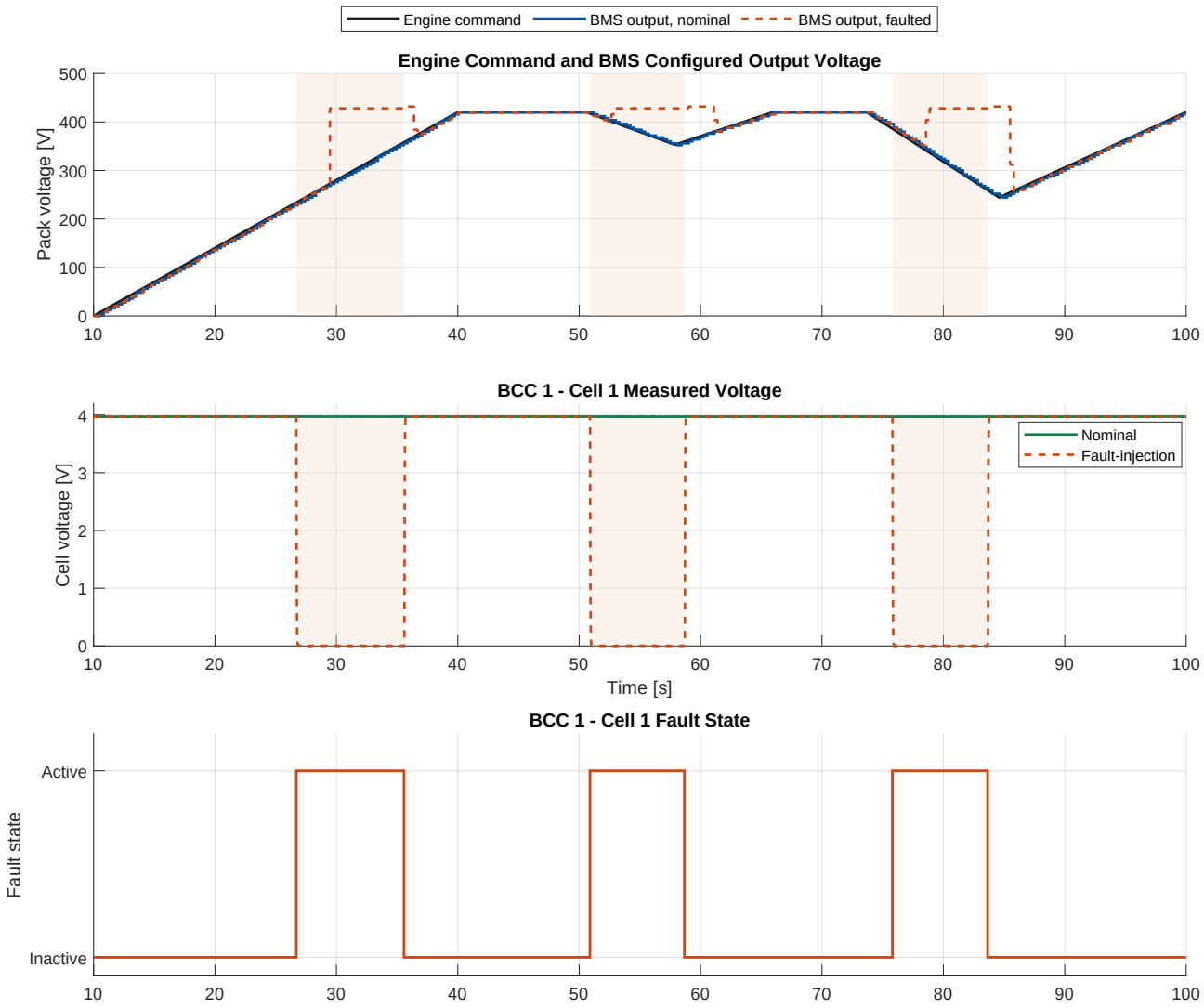


Figure 7 – Component-level verification with cell fault injection. In the nominal run the BMC and BCC on embedded hardware track the engine voltage command (top, blue). An open-circuit fault on cell 1 of BCC 1 (middle) forces its measured voltage to zero (bottom); the BMC responds by entering Limp mode and commanding the maximum output voltage, so the configured pack output (top, red dashed) saturates for the duration of each fault and recovers once the fault is cleared.

6.3 Closed-Loop Aircraft-Level Verification

At aircraft level, the battery system is verified in closed loop with the nonlinear flight dynamics model executing on the TechSAT ADS2 machine, with both controllers running on embedded hardware and the cells emulated in real time. Figure 8 shows a representative mission flown twice over the same profile: once nominally and once with intermittent cell faults injected in flight. The engine commands the maximum pack voltage during the high-power take-off and the subsequent high-demand segments, and a lower voltage during the climb and cruise, while the aircraft altitude increases from approximately 580 m to a cruise value of about 1570 m. Throughout the mission, open-circuit faults are injected and cleared repeatedly on cell 1 of BCC 1 (bottom plot); during each fault the BMS enters Limp mode and commands the maximum output voltage, exactly as observed at component level (top plot, dashed). Despite the repeated in-flight faults, the altitude traces remain closely aligned for this mission profile between the nominal and fault-injection runs, confirming that the BMS tolerates the faults without preventing completion of the simulated mission. The cost of the fault response instead appears in the battery state of charge, which falls from 100% to approximately 93% in the nominal run but to about 87% with the intermittent faults, quantifying the additional energy drawn by the repeated Limp-mode excursions. This demonstrates closed-loop, in-flight fault injection and its consequences.

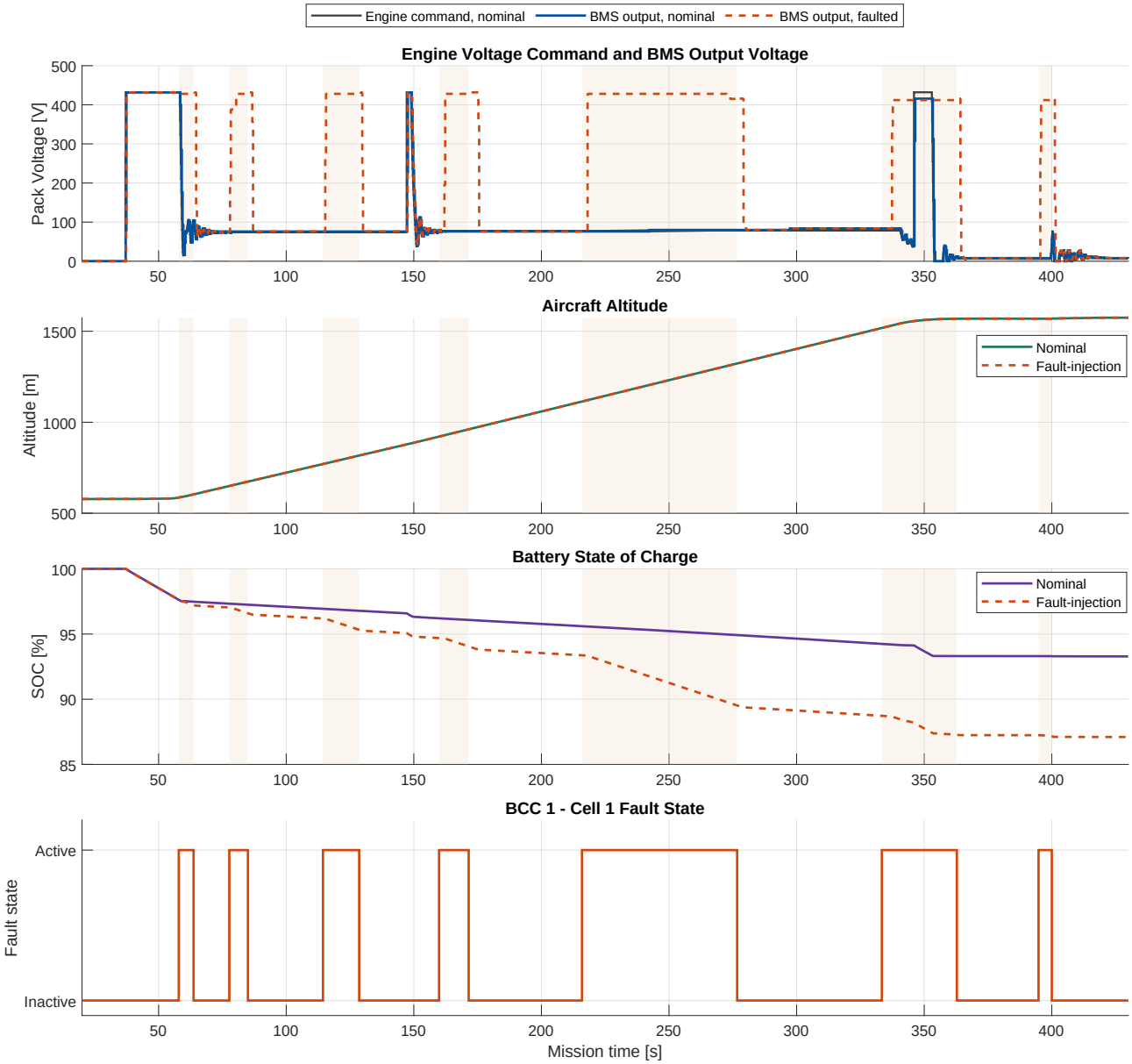


Figure 8 – Closed-loop aircraft-level verification over a representative mission, flown nominally and with intermittent in-flight cell faults. The distributed BMS runs on embedded hardware against the emulated pack and the real-time flight dynamics model on ADS2. Each open-circuit fault on cell 1 of BCC 1 (bottom) drives the BMS into Limp mode and maximum commanded voltage (top, dashed); the aircraft altitude is unaffected (second), while the battery state of charge depletes faster under the repeated faults (third).

6.4 Real-Time Performance

The integrated controller and plant models execute in real time within the $1\,\mathrm{ms}$ base sample time of the application. Across a full mission, only five CPU overloads occur, all confined to the initial start-up phase within the first ten seconds; thereafter the application runs without any overloads, and the same five start-up overloads appear in both the nominal and the fault-injection runs. The measured task execution times (TETs) are essentially identical for the two runs, confirming that fault injection adds no measurable computational load: the base-rate task averages approximately $0.19\,\mathrm{ms}$ in both cases, and its worst-case execution time of about $5.6\,\mathrm{ms}$ occurs only during start-up, which accounts for the five transient overloads. In steady state the TETs remain well within the base sample time, leaving substantial timing margin. Notably, this real-time performance is achieved with the battery plant models running on the target CPU, without offloading them to the FPGA. This contrasts with the earlier per-cell, serial-communication architecture, for which CPU overloads necessitated partial

migration of the plant models to the FPGA [29]; the present CAN-based distributed architecture and reduced model count make such offloading unnecessary at the current fidelity.

7. Conclusion

This paper presents a HIL testbench for the verification of a reconfigurable battery management system developed for an electrically powered glider. The battery controller software is developed using a model-based approach following the objectives of RTCA DO-178C and DO-331, enabling systematic verification across multiple development stages. The proposed HIL framework supports system-level evaluation of the controller software with real CAN communication, embedded targets, and emulated battery cells. The results demonstrate the reuse of requirements-based tests from MIL to HIL, correct component-level fault response on embedded hardware, and closed-loop aircraft-level fault injection with measurable system-level effects on battery state of charge. The modular structure of the testbench makes it applicable to a wide range of electrified aircraft architectures and provides a scalable foundation for future extensions.

Future work includes the automated generation of CAN communication interfaces from CAN database files and Simulink bus definitions. In addition, a graphical user interface can be developed to support communication between the joystick, Microsoft Flight Simulator, and the real-time systems. Once the temperature measurement PCB becomes available, the TSE module is integrated to emulate temperature sensor behavior and extend the testbench to combined voltage, temperature, and fault-injection scenarios.

Overall, the presented work shows that the same controller software can be exercised in MIL, component-level HIL, and closed-loop aircraft-level HIL without modifying the verified development models, providing a practical bridge between model-based software verification and aircraft-level validation. This is particularly important for distributed battery systems, where controller logic, communication timing, embedded hardware, and fault behavior must be evaluated together before progressing toward flight testing.

Acknowledgements

This research and the ELAPSED project are funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr. dtec.bw is funded by the European Union – NextGenerationEU [30].

Contact Author Email Address

Purav Panchal (Primary author), purav.panchal@unibw.de, +4915143666767

Copyright Statement

The authors confirm that they, and/or their company or organization, hold copyright on all of the original material included in this paper. The authors also confirm that they have obtained permission, from the copyright holder of any third party material included in this paper, to publish it as part of their paper. The authors confirm that they give permission, or have obtained permission from the copyright holder of this paper, for the publication and distribution of this paper as part of the ICAS proceedings or as individual off-prints from the proceedings.

References

[1] Melih YILDIZ. Electric energy use in aviation, perspective and applications. *Politeknik Dergisi*, 24(4):1605–1610, December 2021.

[2] Burak Tarhan, Ozge Yetik, and Tahir Hikmet Karakoc. Hybrid battery management system design for electric aircraft. *Energy*, 234:121227, 2021.

[3] Yawen Liang, Gautham Ram Chandra Mouli, and Pavol Bauer. Charging technology for electric aircraft: State of the art, trends, and challenges. *IEEE Transactions on Transportation Electrification*, 10(3):6761–6788, 2024.

[4] Amir Farakhor, Di Wu, Yebin Wang, and Huazhen Fang. A novel modular, reconfigurable battery energy storage system: Design, control, and experimentation. *IEEE Transactions on Transportation Electrification*, 2023.

[5] Song Ci, Ni Lin, and Dalei Wu. Reconfigurable battery techniques and systems: A survey. *IEEE Access*, 4:1175–1189, 2016.

[6] Hahnsang Kim and Kang G. Shin. On dynamic reconfiguration of a large-scale battery system. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 87–96, 2009.

[7] Vidhi Patel, Concettina Buccella, and Carlo Cecati. Analysis and implementation of multilevel inverter for full electric aircraft drives. *Energies*, 13(22), 2020.

[8] Lidiya Komsiyska, Tobias Buchberger, Simon Diehl, Moritz Ehrensberger, Christian Hanzl, Christoph Hartmann, Markus Hölzle, Jan Kleiner, Meinert Lewerenz, Bernhard Liebhart, Michael Schmid, Dominik Schneider, Sascha Speer, Julia Stöttner, Christoph Terbrack, Michael Hinterberger, and Christian Endisch. Critical review of intelligent battery systems: Challenges, implementation, and potential for electric vehicles. *Energies*, 14(18), 2021.

[9] Manuel Kuder, Julian Schneider, Anton Kersten, Torbjoern Thiringer, Richard Eckerle, and Thomas Weyh. Battery Modular Multilevel Management (BM3) Converter applied at Battery Cell Level for Electric Vehicles and Energy Storages. In *International Exhibition and Conference for Power Electronics*, pages 1–8, 2020.

[10] *Software Development of a Multilevel Battery System for an Electric Aircraft*. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 2024.

[11] DO-178C. Software Considerations in Airborne Systems and Equipment Certification. Standard, RTCA, 2011.

[12] DO-331. Model-Based Development and Verification Supplement to DO-178C and DO-278A. Standard, RTCA, 2011.

[13] W. Eric Wong, Andrea Demel, Vidroha Debroy, and Michael F. Siok. Safe Software: Does It Cost More to Develop? In *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement*, page 198–207. IEEE, June 2011.

[14] Purav Panchal, Konstantin Dmitriev, Stephan Myschik, and Florian Holzapfel. Comprehensive Overview of a Process-Oriented Build Tool for Airborne Safety-Critical Software Development. In *2023 10th International Conference on Recent Advances in Air and Space Technologies (RAST)*, pages 01–06, 2023.

[15] Markus Tobias Hochstrasser. *Modular model-based development of safety-critical flight control software*. Dissertation, Technische Universität München, München, 2020.

[16] Siemens. Application Lifecycle Management (ALM), Requirements Management, QA Management | Polarion - Software — polarion.plm.automation.siemens.com. `https://polarion.plm.automation.siemens.com/`, 2004.

[17] PolyspaceMetrics. MathWorks Account Sign In — mathworks.com. `https://www.mathworks.com/help/bugfinder/ug/review-code-metrics.html`, 2022.

[18] Vector. VectorCAST - Software Test Automation for High Quality Software. `https://www.vector.com/int/en/products/products-a-z/software/vectorcast/`, 2023.

[19] Rapita. Rapita Systems | DO-178 Software Verification Tools & Services. `https://www.rapitasystems.com/`, 2025.

[20] Viktor Sinitsyn, Nils Schlautmann, Florian Schwaiger, and Florian Holzapfel. Streamlined Airborne Software Development for Large UAVs: From Unified Data Collection to Automated Code Generation, 2025.

[21] Florian Thomas Schwaiger. *Efficient Software Tools for Scalable Model-Based Validation and Verification*. PhD thesis, Technische Universität München, 2026.

[22] TUM SimPol. SimPol | TUM; Institute of Flight System Dynamics; Software — fsd.lrg.tum.de. `https://www.fsd.lrg.tum.de/software/simpol/`, 2018.

[23] Jenkins Community. Jenkins - Continuous Integration Platform. Website, 2024.

[24] Luca Hein, Purav Panchal, Denis Surmann, and Stephan Myschik. Performance analysis of an electrically powered general aviation aircraft using parallelized automated mission simulations. In *AIAA AVIATION 2023 Forum*. American Institute of Aeronautics and Astronautics, June 2023.

[25] TechSAT. TS-ADS2-ProductFamily | TechSAT — techsat.com. `https://www.techsat.com/testsystems/ads2-productfamily`, 2023.

[26] Speedgoat. Performance real-time target machine | Speedgoat — speedgoat.com. `https://www.speedgoat.com/products-services/real-time-target-machines/performance-real-time-target-machine`, 2023.

[27] Speedgoat. Testing Battery Management Systems | Speedgoat — speedgoat.com. `https://www.speedgoat.com/products-services/i-o-connectivity/bms-and-battery-testing`, 2026.

[28] MicroSys. SBC S32G274A ARM Architecture | MicroSys — microsys.de. `https://www.microsys.`

de/en/products/sbc/arm-architecture/miriac-sbc-s32g274a/, 2023.

[29] Purav Panchal, Luca Hein, Wolfgang Bliemetsrieder, Nina Sorokina, and Stephan Myschik. Integration, Verification and Validation of a Model-Based Avionic Software in a Real-Time Flight Dynamics Environment. In *AIAA SCITECH 2024 Forum*, 2024.

[30] DTEC. Electric Aircraft Propulsion – die Zukunft der Flugzeugantriebe, 2021.