Validation of a Battery Management System based on AUTOSAR via FMI on a HiL platform

Leonard Janczyk[1] Klemens Esterle[1] Stephan Diehl[1]
Michael Seibt[1] Arthur Gauthier[2] Viry Guillaume[3]

[1]Dassault Systèmes Deutschland GmbH, Munich, Germany
[2]Dassault Systèmes SE, Plouzané, France
[3]Dassault Systemes KK, Tokyo, Japan

leonard.janczyk@3ds.com, klemens.esterle@3ds.com, stephan.diehl@3ds.com,
michael.seibt@3ds.com, arthur.gauthier@3ds.com, guillaume.viry@3ds.com

Abstract

In systems which are sourcing their electric energy from a battery system, such as electric or hybrid electric vehicles, it is of crucial importance to monitor the battery's condition in order to ensure its usability and longevity. The battery management system (BMS) is a control unit which supervises the physical variables in order to assess the condition of the battery.

For the development and testing of control units in the automotive industry, such as the BMS, the AUTOSAR standard was introduced, which separates application code from platform-specific software. By using AUTOSAR tools and the model exchange via the Functional Mock-up Interface (FMI), this paper shows how BMS algorithms can be validated and tested in several abstraction layers. A sub-function of the algorithm is tested first in the Modelica-based system simulation tool Dymola on a personal computer and then on Hardware-in-the-Loop (HiL) platform which emulates the hardware of an automotive ECU.

In order to provide realistic inputs of the physical variables, a battery model in Modelica is built using the Dymola add-on Battery Library by Dassault Systèmes. In order to run on the HiL platform the battery model is implemented such that it is real-time compliant.

For both, the BMS algorithm and the battery model, it is described along the process which adjustments need to be made when switching from the simulation framework to the HiL platform.

Keywords: battery model, battery management system, AUTOSAR, FMI, ASim, MiL, SiL, HiL, XiL, Co-Simulation

1 Introduction

Today's system- and software development teams work quite isolated from one another. Information exchange is usually limited on written specifications. With the example of the battery management system (BMS) we will show a method in which information can effectively be exchanged through a model based on the Functional Mock-up Interface (FMI) as executable specification. This allows both parties closed-loop simulation at different stages of the V-Cycle. This way, software developers can more thoroughly test their software in a virtual environment. At the same time the system simulation teams can simulate their whole system without the need to manually re-implement the software algorithms of the ECU code.

This paper illustrates how based on FMUs (Functional Mock-up Units) source code from an AUTOSAR Battery Management Algorithm can be simulated on different abstraction levels in order to verify the algorithm for a failure mode.

At first, in section 2, the physical battery model will be introduced along with example battery module which it represents. In a second step, the function and tasks of a battery management system will be explained. The focus will shift on the specific algorithm, the charging status estimation, which is chosen as an example in order to demonstrate the process for the overall BMS. In section 3, the AUTOSAR standard and the used tool chain will be described.

2 Battery Simulation and Battery Management

Proper battery modelling plays an important role in this context. On the one hand, the model needs to provide a proper representation of the inner workings of the battery so the battery management system receives a realistic and complete set of signals.

On the other hand, the battery model needs to be performant enough in order to be compliant with real-time requirements. In the following two sections, the battery model will be introduced.

2.1 Battery Simulation Model

The battery pack which is modelled is a 48 V module. It could be deployed in micro-hybrid systems for the on-board electric power supply or as part of a traction

battery system. The module features two parallel-connected rows of each 13 battery cells in serial electric connection. Their combined capacity of the 26 battery cells amounts to 140 Ampere hours.

Table 1. Battery Cell Parameters.

Parameter	*Unit*	*Value*
Nominal Cell Capacity	Ah	2.7
Nominal Cell Voltage	V	3.6
Maximum Voltage	V	4.2
Minimum Voltage	V	2.5
Maximum Internal Resistance	mΩ	30
Shape	-	round

The battery module is modelled in Modelica using the Battery Library from Dassault Systèmes (Gerl, et al. 2014). The physical battery cell models is made up of the physical domains relevant for the batteries behavior: electric and thermal.

The main requirement for cell models used in system simulation is to provide accurate information on the macroscopic characteristics (e.g. voltage, current and state of charge) combined with reasonable computation time. This way, the impedance characteristics of the real cell are replicated. In many applications models using an electrical equivalent circuit fulfill these requirements

The voltage of a battery U can be described as the difference between the open circuit voltage U_{OCV} and a number of over potentials η_i caused by different electrochemical effects:

$$U = U_{OCV} + \sum \eta_i \qquad (1)$$

These over potential can be modelled with equivalent electric circuit networks. In Figure 2 the voltage characteristic for the step current discharge of a NiMH cell is shown. The effect is similar for Lithium-Ion based cells, such as the ones used for this example.

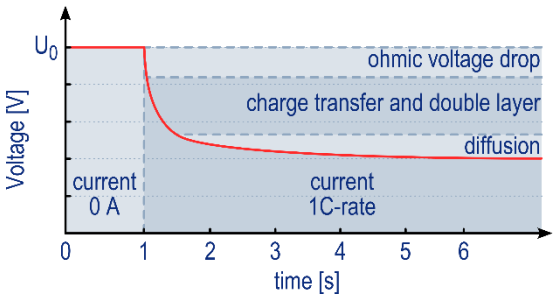

Figure 1 Voltage characteristic of an electrochemical cell (NiMH) **(Jossen und Weydanz 2006)**

The over potential is divided into an ohmic over potential η_{ohm}, over potential caused by charge transfer and the electrical double layer η_{trans} and over potential due to diffusion η_{diff}. An electrical equivalent circuit capable of reproducing the voltage characteristic from Figure 1 is shown in Figure 2, whereas the dynamic behavior of the over potentials are modelled using RC-circuits.

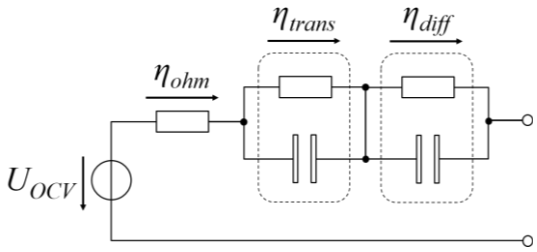

Figure 2 Voltages in the equivalent circuit model

In order to determine the influence of varying temperatures on electrical and aging behavior a thermal model of the cell and its surrounding environment is required. The heat inside the cell is generated mainly due to Joule effects, while the chemical reactions are exothermic or even endothermic to a minor degree. Thus the generated heat corresponds to the calculated power losses of the resistors of the equivalent electric circuit which are therefore connected to the thermal model.

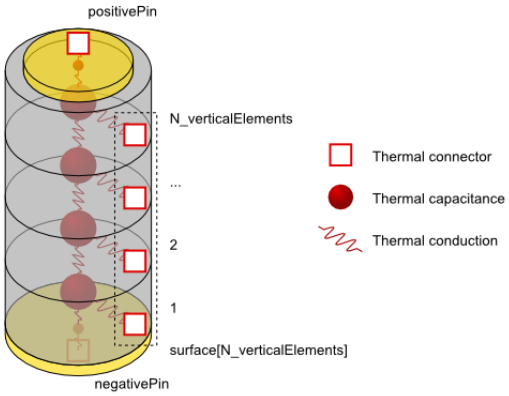

Figure 3 Representation of the thermal cell model

At the module level, where several cells form an electric, geometric and thermal entity, the major advantage in this context is that the battery pack can be adjusted according to the performance needs. In practical terms, this results in the question whether the battery cells are each represented as a Modelica object. A simplified approach would be modelling just one cell and scaling up the results to module size by multiplying the inputs and outputs by the cell numbers in accordance with their electrical wiring. Also, the thermal representation of the cells can be adjusted in the number of discretized elements. In the case of a round cell these elements are vertically slices which help to calculate the cell internal flow, as sketched in Figure 3.

Of practical importance is the fact that the Hardware-in-the-loop platform usually does not feature a data system. Modelica models in industrial environments might be parameterized by external parameter files. When exporting the model for the HIL environment, the data needs to be placed within the model without any external dependencies.

DOI 10.3384/ecp1612487

2.2 Battery Management System

The battery management system has to process sensor data and on-board model simulation results in order to obtain information about the state of the battery. Tasks of the battery management system (BMS) include the determination whether the monitored variables are still within the acceptable limits. Apart from state-of-charge (SoC) and the state-of-health (SoH) usually the cell temperature, the cell voltage and the system voltage are monitored. In case one of these variables appears to be out of the operational limits, the battery management

$$SoC = f(U) = f(U_{OCV}) \text{ for } \eta_i \to 0 \qquad (3)$$

system sends a signal to the overall power management control unit which restricts the power usage of the consuming components.

Especially the SoC and the SoH are variables, which need to be monitored to ensure overall system availability at any given moment (He, Wei and Brian 2010). When the SoC reaches a critically low level in general in the range of 5-10%, the electrodes of the battery take severe damage and the battery voltage might drop below a level at which the battery system cannot provide the required power anymore. On the other hand, when the SoC exceeds 100% by too much, the battery cell stores excessive amounts of energy beyond a level which it can safely handle. In severe cases, this might even a cause a "thermal runaway", a strong exothermal reaction after which the battery system is completely dysfunctional.

Therefore in any case the battery management system should encompass a SoC estimation algorithm. In the following, the realization of such an algorithm will be discussed.

2.3 Estimation of Battery State-of-Charge

In a battery simulation model the change of the SoC can be calculated by balancing the electric charge and discharge current such as in equation (2). The SoC is by definition part of the overall amount of electric charge available for discharge with C_n being the nominal battery capacity in Ampere seconds (He, Wei and Brian 2010).

$$\Delta SoC = \frac{\int_{t_0}^{t_{end}} I \, dt}{C_n} \qquad (2)$$

However the SoC determination is more complex when being implemented on a battery ECU.

First, integration is a mathematical operation which requires more resources in terms of on-board memory and computational time compared to other mathematical operations.

Secondly, current sensors do not necessarily deliver a constantly precise measurement output. Calibration errors result in constant drifts of the recorded battery current. This drift might not significantly influence the quality of the estimation during a short period such as a short inner city ride. However during longer trips such as an inter-city highway tour the drift in the charging status estimation might accumulate to a point where the battery is depleted while the battery management system assumes the charging status as being sufficient.

In order to ameliorate the quality of the SoC estimation, corrective back-up algorithms need to be included. A viable alternative is measuring the voltage of the battery when the battery is in electrochemical equilibrium, meaning that no electric load or charging is applied and excitation of previous electric load has faded. In this state the over potentials are negligible leaving the open circuit voltage as the dominant factor determining the cell voltage:

As a matter of fact, the open circuit voltage is usually measured during the initial rating of new cell type and also typically used for the parameterization of equivalent circuit cell models as shown in Figure 2.

Implementing the relationships presented in equations (2) and (3) in Modelica code could be drafted as followed:

```
der(SoC_count) * C_n = current + error;
SoC_est = SoC_count;
when zeroCurrentTimer > fadingTime then
  reinit(SoC_est, SoC_ocv);
end when;
```

In the first code line the charge counter (`SoC_count`) is implemented after the fashion of equation (2) with the `error` signal applied on the `current` signal. The output `SoC_est` is directly loaded with the result of the integration over the current and standardization with the nominal capacity `C_n`.

The *when*-clause representing equation (3) becomes active at the time point at which the current has been close to zero for a time period, implementation shown in Figure 4, with the influence of charge transfer over potential has most likely faded, in this case more than the time constant `fadingTime`. The calculated SoC will be replaced then with a charging status which has been extracted from a look-up-table describing SoC over OCV.

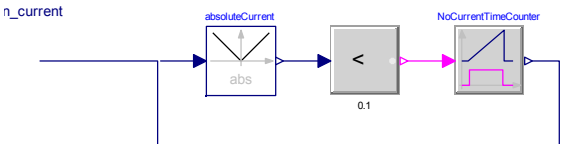

Figure 4 Counting time with current close to zero

During the verification phase of the system engineering process, the battery management system needs to be verified if it lives up to battery safety requirements, i.e. if the variables describing the battery state are recorded properly, the operational limits are correctly determined and their violations duly signaled.

In context of this paper, the ability of the charging status estimation algorithm to correct an erroneous

current measurement signal will be verified. For demonstration purposes the implementation is limited to charge counter and correction by voltage comparison as laid down in this chapter. A typical question answered during this process might be whether an estimation correction via cell voltage is sufficient to ensure that the battery management system and the driver are provided with the correct battery charging status.

3 Virtual Testing of AUTOSAR compliant controller software using FMI

3.1 What is AUTOSAR?

AUTOSAR (AUTomotive Open System ARchitecture) is a well-accepted standard for developing software for automotive electronic control units (ECU), as documented by (Bertsch, et al. 2015) for Bosch ECUs. It defines a layered architecture, separating hardware, application software and basic software through standardized interfaces.

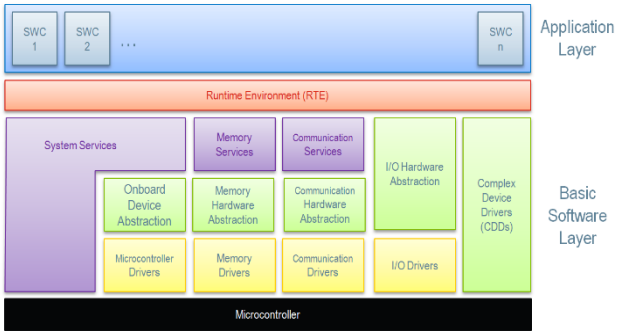

Figure 5 AUTOSAR Layered Architecture

As shown in Figure 5 the interface between application software components and the interfaces between application software and basic software (BSW) is handled through the runtime environment (RTE), which implements different types of communication mechanisms (AUTOSAR 2016).

3.2 AUTOSAR Unit Test

One essential part in the AUTOSAR software development is the testing of individual software components and the whole software architecture (top level composition). Ideally those tests should be executable without any hardware-dependencies to enable testing as soon as possible in the development cycle. AUTOSAR addresses this through the Virtual Function Bus (VFB) Abstraction Level. The AUTOSAR test environment ASim from Dassault Systèmes is also applying this concept taking a real AUTOSAR compliant operating system (OS) and RTE into account. This allows testing of software components on a very granular level, also considering effects through the OS, e.g. scheduling, or through the RTE, e.g. synchronize queued and non-queued communication. Even fixed-point arithmetic is taken into account using AUTOSAR datatypes.

3.3 FMI-based Export of virtual AUTOSAR ECUs

Unit tests are in general open-loop tests, which means the user has to define sufficient and reasonable test-vectors and test-constraints, which is often quiet challenging and time-consuming. Hence integrating the software "model" in a virtual environment which closes the loop through a plant-model would make the conditioning of many of the software component-ports obsolete, as they will be fed directly through the connected plant model. In addition to that timing effects and delays could also be taken into account by a plant model. ASim opens this possibility through FMU export the extraction of a virtual AUTOSAR ECU which can then be integrated into other simulation platforms supporting the FMI-Standard.

3.4 FMI-based XiL Tool Methodology

The Modelica-based simulation tool Dymola supports the import, export and simulation of FMUs (FMI for Model Exchange and Co-Simulation, (Blochwitz, et al. 2011)). Software- and plant-models can be simulated on different abstraction levels, which allows MiL- and SiL-testing. FMUs can also be exported via the source-code generation capabilities. These can then be compiled for different HiL platforms. Based on a Battery Management Unit it is illustrated, how XiL-tests can be performed using the FMI-standard.

4 Results and Discussion

4.1 Model-in-the-Loop

In general, system simulation starts at an earlier stage in product development than the software development. In this context both battery model and BMS algorithm are implemented as models to evaluate system behavior and the response algorithm together in a Model-in-the-Loop (MiL) simulation. One could argue that the BMS algorithms could be coded from the beginning in a software development platform for the ECU software instead of being implemented in the same simulation environment as the model itself. However when taking a closer look at the model equations and the required solvers, the advantage of this method will become obvious:

Using an acausal object-oriented Modeling language like Modelica for modeling physical systems often results in a higher-order differential algebraic equation system (DAE) with slow dynamics, looking at the thermal behavior of the battery case and fast dynamics caused by the electrical cell behavior. An implicit solver like DASSL is designed to deal with those type of systems. As only explicit fixed-step solvers can be used in a real-time environment, numerical stability for the

DOI 10.3384/ecp1612487

sampling rate of the ECU has to be ensured. Testing the battery model in the modeling environment with the step-size matching the sampling rate reduces the use of generally much more expensive real-time hardware.

In addition by using this approach the functionality of the battery model can be verified in parallel to the development of the battery management system by the functional development engineer resulting in an acceleration of the design phase. Moving forward in the product development, this procedure allows early simulation experiments.

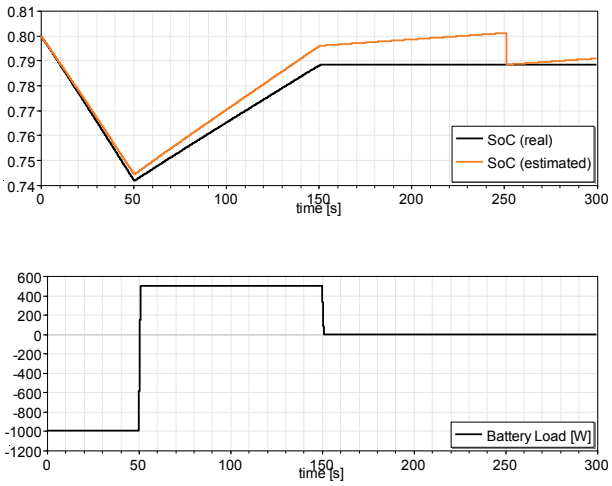

Figure 6 Model-in-the-Loop Simulation: SoC (*top*) and power load (*bottom*) over time in seconds. Negative power indicates discharging of the battery module.

When coupling the Modelica implementation of the SoC estimation algorithm with the Battery Library model and applying a load cycle and environmental conditions, the following results are obtained as shown in Figure 6. The load power cycle consists of a discharge phase of 1 kW, an immediate recharge of 500 W and a subsequent unstressed time period at room temperature. One can observe that due to the forced condition offset error of +0.5 A on the current sensor, the output of the charging status estimation drifts away from the actual SoC up to the point where the divergence amounts to over one percentage point. A short time period later the power load is stopped. The cell voltage is largely no longer influenced by the electrochemically induced overpotentials but only by the open circuit voltage. The corrective algorithm steps into action, looks up the SoC value which matches the measured voltage. At second 250 the *reinit* command is ignited and replaces the calculated SoC value with the one based on the measured cell voltage.

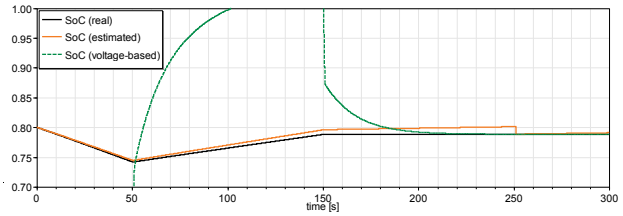

Figure 7 Model-in-the-Loop Simulation: Different Methods of SoC over the course of the battery load cycle

A comparison in Figure 7 between the SoC values also shows at this early software design stage why charging status based on the measured cell voltage cannot serve as a lone signal source, and why a certain time period needs to pass before the correction is applied.

As the focus is the evaluation of the concept per se, the simulation is performed on a high-performing workstation with characteristics described in Table 2.

Table 2 Technical Characteristics of the workstation **(Intel Corporation 2016)**

Parameter	*Value*
CPU Type	i7-4810MQ
Instruction Set	64 Bit
Number of CPU Cores	4
Base Frequency	2.8 GHz
L2 Cache size	1 MB
L3 Cache size	6 MB
RAM	16 GB

With summoning such computing power while using a language which is native for the Dymola solver shows a satisfying result for simulation time: The 17671 equations of the Modelica are integrated in 110 seconds while the CPU-time for one GRID interval is 0.365 milliseconds.

4.2 Software-in-the-Loop

Once the algorithms of the BMS have been drafted and evaluated during the MiL testing, they are implemented as software functions for the ECU. Using the ASim plugin of the Autosar Builder, the BMS algorithm can be exported as FMU and coupled with the physical battery simulation model in Dymola. As the algorithm is now in the same format as on the ECU, this stage is called Software-in-the-Loop (SiL) simulation.

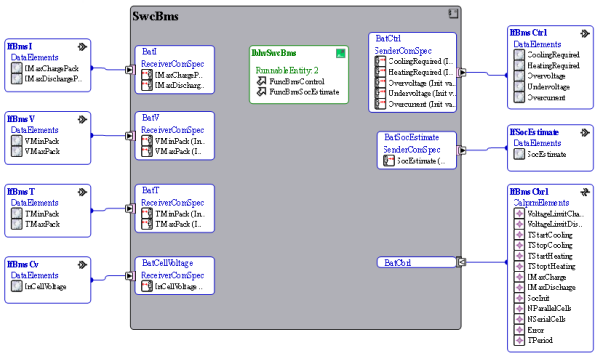

Figure 8 AUTOSAR Software Component of BMS. The interfaces for I/O and calibration are marked in blue, the internal behavior is marked green (presented in Figure 9).

In this stage of the process, the stability is first verified with the ECU code coupled with the battery model in Modelica in Dymola and in a second step as in FMU both again imported in Dymola. Taking into account the sampling. time of the Hardware-in-the-Loop platform, it is to be tested whether coupled entities are running stable when being operated with a fixed-step solver with a sampling time of one millisecond.

When implementing the charge status algorithm as sketched out in Modelica for the MiL simulation, some methods need to be altered in order to ensure a sufficient performance on an ECU for implementing the charging status estimation algorithm:

As mentioned before in 2.3, the integration operation used in equation (2) would use too much on-board memory and is not always available in the ECUs instruction set, so it needs to be replaced by a discrete sum operation which accumulates the measured current with each time step.

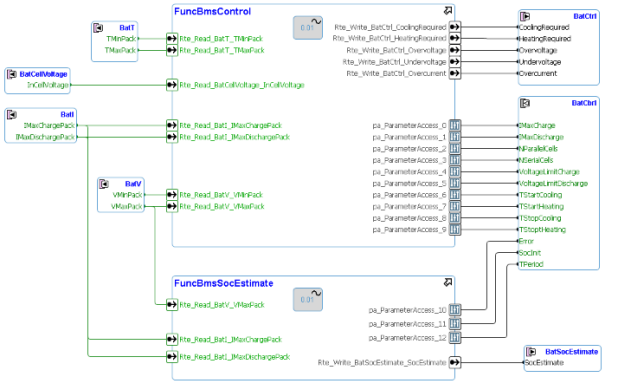

Figure 9 AUTOSAR Builder Screenshot of the internal behavior of BMS. The supervision algorithm for the operational limits is in the upper block *FuncBmsControl*, while the charging status estimation is in bottom block *FuncBmsSocEstimate*.

As shown in Figure 10, in both cases the system of physical model and algorithm works stable with reasonable results similar as obtained in the Model-in-the-Loop simulation in chapter 4.1.

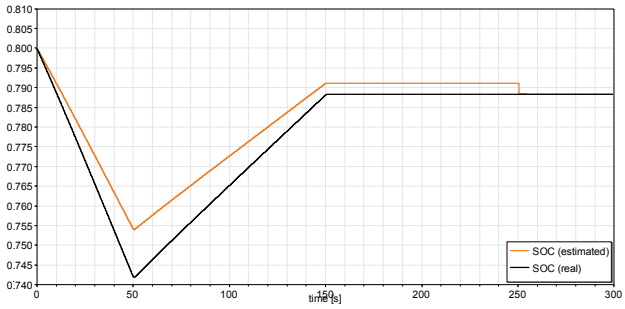

Figure 10 SiL Simulation: Results AUTOSAR-FMU and Modelica Model in Dymola.

In the SiL simulation, the Dymola solver is now slowed down by processing the FMU. The integration time amounts now to 187 seconds with a 0.618 milliseconds per grid intervall.

4.3 Hardware-in-the-Loop

In the final stage of the verification of the charging status algorithm, the battery model in Modelica and the BMS algorithms in AUTOSAR C-code are exported as FMUs and executed on a platform which emulates the hardware of an ECU of the target system. This stage is therefor called Hardware-in-the-loop. At this point, the stability of the software in a real-time environment can verified. Additionally, hardware specific effects, such as the influence of signal propagation delays, limited memory, cache and processing speed are playing out as well.

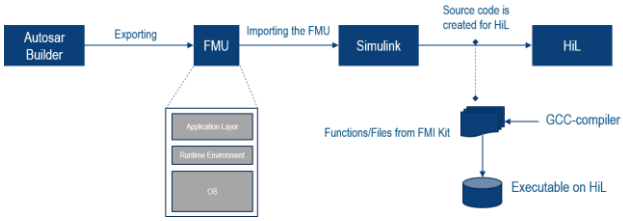

Figure 11 Toolchain and process for creating an FMU for a HiL platform.

The battery model FMU and the Autosar FMU are both set on the HiL platform. For this purpose, a dSpace DS1006 Processor Board is employed as specified in Table 3.

For being executed on the HiL platform, the AUTOSAR FMU had to be equipped with operating system functionalities such as scheduling. The toolchain is visualized in Figure 11 **Toolchain and process for creating an FMU for a HiL platform.**Figure 11.

DOI 10.3384/ecp1612487

Table 3 Technical Characteristic HiL platform's CPU **(dSpace GmbH 2016)**

Parameter	*Value*
CPU Type	Opteron
Instruction Set	64 Bit
Number of CPU Cores	4
Base Frequency	2.6 GHz
L2 Cache size	1 MB
L3 Cache size	-
RAM	128 MB

For the virtual validation of the charging estimation algorithm, the output signals are compared to the results in the MiL simulation in chapter 4.1, as shown in Figure 12.

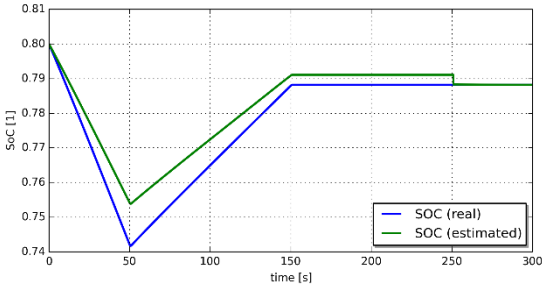

Figure 12 HiL Simulation: Results of the combined BMS FMU exported from AUTOSAR and battery model FMU exported from Dymola on the HiL platform.

The important characteristic for ensuring real-time requirements is the turnaround rate in percentage points. It states which fraction or multiple of the fixed-step sample time interval is consumed for the execution of the software code.

The turnaround time of the combined BMS FMU exported from AUTOSAR and battery model FMU exported from Dymola on the HiL platform is below 0.4 milliseconds. With the HiL platform processing according to a step time of 1.0 millisecond, the turnaround indicating a performance fast enough in order to be real-time capable.

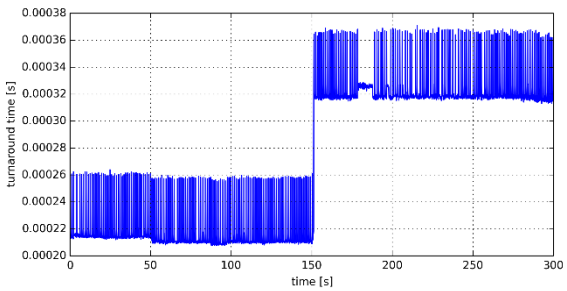

Figure 13 HiL Simulation: Turnaround time when executing the combined FMUs on the HiL platform.

5 Conclusion and Outlook

In this paper, it could be shown that the detailed battery model based on Dassault Systèmes Battery Library can be used for real-time applications as the derived system could be solved using a fixed-step integration method with a step-size of one millisecond. The BMS functionalities developed in AUTOSAR could be validated using the battery model as a FMU on the HiL platform.

With the example of the charging status estimation algorithm, it has been shown how BMS functions could be developed from draft to real-time verification using FMI across all stages of the process from MiL over SiL up to HiL.

As outlook from the perspective of the tool chain it should be noted that currently the FMU generated from the ASim in the AUTOSAR Builder uses the VFB level, which doesn't take the Basic Software or Complex-Device Drivers (CCDs) into account. In a next step, parts of the AUTOSAR Basic Software or CDDs could be also modelled and exported with the FMU. This would then also allow the consideration of propagation delays induced by the Basic Software Layer.

Acknowledgements

We would like to thank Dan Henriksson from Dassault Systèmes AB in Lund, Sweden, for his support on the HiL platform.

References

AUTOSAR. 2016. *Technical Overview.* February 06. http://www.autosar.org/about/technical-overview/.

Bertsch, Christian, Jonathan Neudorfer, Elmar Ahle, Siva Sankar Arumugham, Karthikeyan Ramachandran, and Andreas Thuy. 2015. "FMI for physical models on automotive embedded targets." *11th International Modelica Conference.* Versailles, France. 43-50. doi:10.3384/ecp1511843.

Blochwitz, Otter, Bausch, Clauß, Elmqvist, Junghans, Mauss, et al. 2011. "The Functional Mockup Interface for Tool independent Exchange of Simulation Models." *8th International Modelica Conference.* Dresden, Germany.

dSpace GmbH. 2016. "DS1006 Processor Board." *Technical Details.* February 06. http://www.dspace.com/en/pub/home/pr

oducts/hw/modular_hardware_introduction/processor_boards/ds1006.cfm.
Gerl, Johannes, Leonard Janczyk, Imke Dr Krüger, and Nils Modrow. 2014. "A Modelica Based Lithium Ion Battery Model." *10th International Modelica Conference.* Lund, Sweden: Linköping University Electronic Press. 335-341.
He, Yongsheng, Liu Wei, and Koch J. Brian. 2010. "Battery algorithm verfication and development using hardware-in-the-loop testing." *Journal of Power Sources* (195): 2969-2974. doi:10.1016/j.powersour.2009.11.036.
Intel Corporation. 2016. "Intel® Core™ i7-4810MQ Processor." *Specifications.* February 06. http://ark.intel.com/products/78937/Intel-Core-i7-4810MQ-Processor-6M-Cache-up-to-3_80-GHz.
Jossen, Andreas, and Wolfgang Weydanz. 2006. *Moderne Akkumulatoren richtig einsetzen.* Reichhardt Verlag.