Design and Analysis of a Controller from System Design Idea to AUTOSAR Architecture with Basic Software Modules

Emad Farshizadeh, Hermann Briese, Steffen Beringer, Dominik Holler, Hamza Raja, Lars Stockmann

Abstract

The development of electronic control units (ECUs) in the automotive industry is becoming increasingly complex. The integration of new controllers is a time-consuming process which is characterized by round trips between control engineers and system architects, especially in the domain of electric mobility. To reduce the number of iterations and therefore time-to-market, we propose that it is beneficial for control engineers to consider aspects of the later software early in the design process. We present a methodology that supports this approach by simulation on different levels of abstraction. It allows control engineers to address basic software such as diagnostics in the design phase. We apply this methodology to a recuperation-ready electrohydraulic brake controller and compare the simulation results.

1. Introduction

Due to dwindling fuel resources and the need to reduce emissions, today's automotive industry is on the move from traditionally combustion engine-powered cars towards alternative drive technologies.

Nevertheless, the number of purely electrically powered cars is still low. The reasons for this are manifold. One reason is that new controllers have to be developed, e.g., for energy management, engine and brake control. The engine controller must be designed, adapted and optimized for each specific electric motor to make driving more energy-efficient. To optimize energy efficiency when braking, the brake controller has to allow recuperation and thus has dependencies on both the brake and the engine. These kinds of controllers have then to be integrated into existing environments, which usually takes a lot of time due to the high safety requirements in the automotive industry. To face this complexity, model based development and code generation are increasingly being preferred over handcoding. The other reason is that in addition to developing the control functionality itself, integrating it into standardized architectures and systems, e.g., complying with AUTOSAR[1], is also a challenging task. Basic software (e.g., the operating system, diagnostics, and drivers) as well as communication middleware have to be configured and generated, which is traditionally not done in the domain of control engineering but by software developers. A round trip between these two domains is evolving, as the control engineers must adapt their models to the new requirements. Despite the distributed nature of

[1] www.autosar.org

the development process, integrity and traceability must be guaranteed from beginning to end.

This makes the development of new complex controllers a long and costly process. New concepts and methodologies must therefore be investigated and applied to accelerate the development of complex electronic control units (ECUs) and boost technological and economic progress in electric vehicle engineering. One concept is to bring both domains closer together by considering aspects from later phases of the development process, such as diagnostics, as early as possible as part of the basic software.

This is one idea behind the research project “Simulationsgestützter Entwurf für Elektrofahrzeuge” (Simulation-Based Development for Electric Vehicles). The project partners that represent the different domains participating in the distributed development of ECU software are DMecS GmbH & Co. KG, the LEA institute, the C-Lab (University of Paderborn) and dSPACE GmbH. In the project, new controller and environment models for electric vehicles are developed and tested. After the functional design the controllers are integrated into an AUTOSAR software architecture. This aspect is important, because AUTOSAR has become the de facto standard in the automotive sector, and the demand for AUTOSAR-compliant software is continuously growing. One main benefit of the AUTOSAR-methodology is that it supports the distributed development of ECU functionality by different suppliers. To easily integrate new software into existing software architectures, seamless tool support is essential. The tool chain used in the project has been further enhanced and adapted to the domain of electric vehicles to support the rapid development of the control software.

This paper describes a part of the project where controller software for a recuperation-ready electrohydraulic brake system is developed and integrated into an AUTOSAR architecture, including basic software modules. It is one in a series of papers. While the previous publication [1] mainly focuses on the functionality of the electrohydraulic brake system, the overall development process, and the integration of an ECU State Manager module, we now go a step further and address the integration of production-level basic software. We therefore extended the tool chain by adding EB tresos® tools from Elektrobit[2] for the configuration and generation of basic software modules. We exemplify our approach by integrating a diagnostic event manager (DEM) to trace erroneous brake system values. We show how control engineers can provide communication with the DEM as early as possible so that it can easily be integrated into the AUTOSAR system later.

2. Electrohydraulic Brake System

This chapter introduces the electrohydraulic brake system (EHBS), which provides braking assistance by a controlled motor. Figure 1 shows the main setup. The brake pedal force applied by the driver is transmitted through the pedal gear and an elastic push rod to the prima-

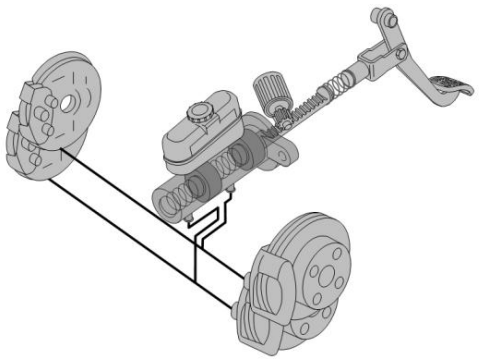


Figure 1: EHBS overview

[2] http://automotive.elektrobit.com/

ry pistons of the master brake cylinder. This force is assisted by the electric power brake booster (EHBS actuator), which consists of a current-controlled motor and a transmission that converts the motor torque to a force on the piston. The resultant displacement of the pistons in the master brake cylinder generates a pressure build-up which provides the braking torques at the wheel brakes. By using the pressure in the master brake cylinder as a controlled variable for superimposed control of the EHBS actuator, a highly dynamic pressure build-up is achieved. In the next section the plant model for the pressure controller design is introduced. After that, a symmetrical optimum design of a PID controller with active damping for EHBS is presented.

2.1 EHBS Plant Model

The physical substitute model of the plant, which is used for the model-based design of the EHBS controller, is presented in figure 2. This model contains the relevant parameters and the degrees of freedom s_{ped} and s_{pist} as well as the external forces F_{dr} and $F_{fr,red}$ and torque T_{epb} that contribute to the system. Here the parameters m_{red}, b_{red} and $F_{fr,red}$ represent the inertia, damping and friction of the motor and piston. The hydraulic capacity C_{red} represents the capacities of the master brake cylinder chambers, the lines to the wheel brake cylinders and the wheel brake cylinder chambers. The corresponding mathematical model of the EHBS plant is given by the equations

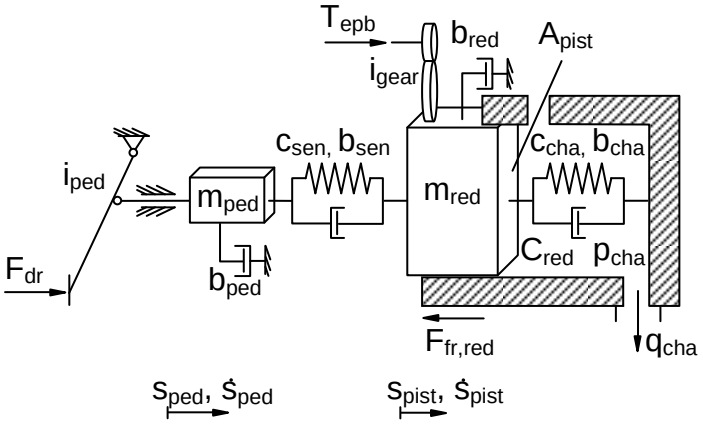


Figure 2: Physical substitute model of the plant

$$m_{ped}\ddot{s}_{ped} = F_{dr}i_{ped} - b_{ped}\dot{s}_{ped} - c_{sen}(s_{ped} - s_{pist}) - b_{sen}(\dot{s}_{ped} - \dot{s}_{pist}), \tag{1}$$

$$m_{red}\ddot{s}_{pist} = T_{epb}i_{gear} + c_{sen}(s_{ped} - s_{pist}) + b_{sen}(\dot{s}_{ped} - \dot{s}_{pist}) - b_{red}\dot{s}_{pist} - c_{cha}s_{pist} - b_{cha}\dot{s}_{pist} - A_{pist}p_{cha}, \tag{2}$$

$$\dot{p}_{cha} = \frac{A_{pist}}{C_{red}}\dot{s}_{pist} - \frac{1}{C_{red}}q_{cha}, \tag{3}$$

where the controlled motor torque T_{epb} is generated from the reference input $T_{epb,req}$ with the equation

$$\dot{T}_{epb} = -\frac{1}{\tau_{epb}}T_{epb} + \frac{1}{\tau_{epb}}T_{epb,req} \tag{4}$$

2.2 EHBS Controller

The plant model shown in the previous subsection forms the basis for designing the pressure controller. The aims of the controller are to ensure steady-state accuracy as well as a fast and well-damped transient response to the reference pressure, and to reject unknown disturbances. A PID controller with the transfer function

$$G_{PID}(s) = K_p \frac{(T_n s + 1)^2}{T_n s (T_r s + 1)} \quad (5)$$

is used for this and its parameters are determined by applying the symmetrical optimum method. Due to the fast resonant mode caused by the compliant coupling of the brake pedal and piston, the plant model of EHBS deviates from ideal plant for symmetrical optimum design. Hence, the resonant mode is damped by the feedback of the derivative of the pressure. The rough structure of the control loop with active damping is given in figure 3, where the derivative of the pressure is calculated by a real differentiator.

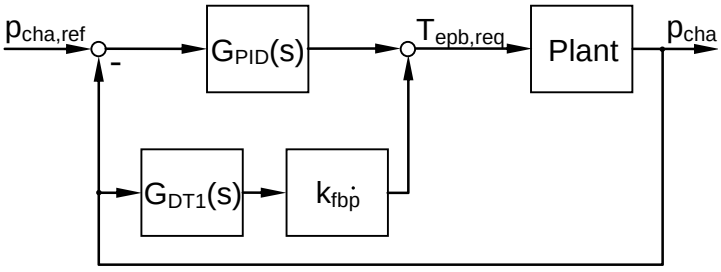


Figure 3: Structure of control loop

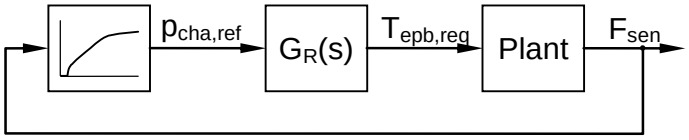


Figure 4: Structure of control loop with reference variable

The driver's desired braking value is detected by measuring the brake pedal force F_{sen} at the elastic push rod. The reference signal $p_{cha,ref}$ for the EHBS controller is generated as a function of the brake pedal force and mimics a conventional brake system with pneumatic brake booster. The structure of the control loop with reference variable is given in figure 4.

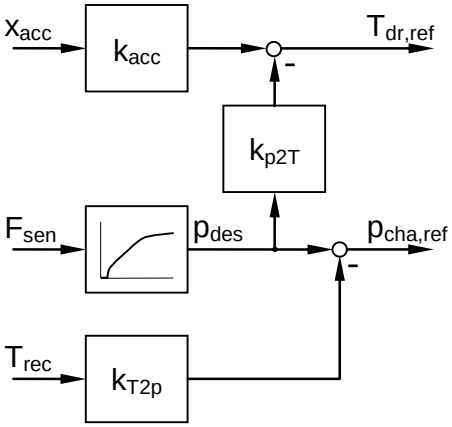


Figure 5: Structure of recuperation controller

2.3 Recuperation Controller

To recover the kinetic energy of the electric vehicle, the drive motor needs to operate as a generator and the EHBS has to prevent pressure being generated at the wheel brakes, which would provide braking torque in addition to the motor torque. Thus, a recuperation controller is required that generates the reference signals for the drive motor $T_{dr,ref}$ and for the EHBS $p_{cha,ref}$ according to acceleration pedal displacement x_{acc} and brake pedal force F_{sen}. Figure 5 shows the structure of the recuperation controller.

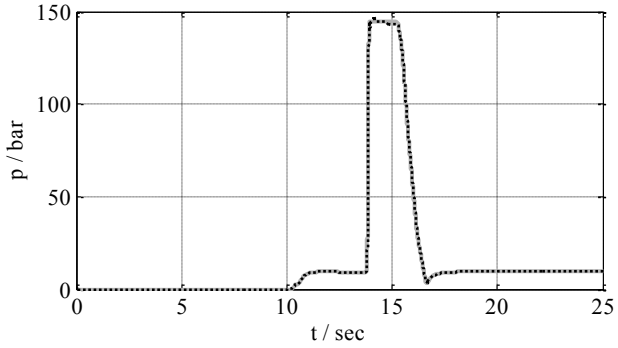


Figure 6: Simulated reference (gray, —) and controlled (black, ··) pressure

The look-up table shown in the previous subsection is now imple-

mented in the recuperation controller and provides the desired braking pressure p_{des}. If the acceleration and brake pedals are applied simultaneously, the reference signal $T_{dr,ref}$ for the drive motor is computed from the difference between the desired drive and brake torques. The reference signal $p_{cha,ref}$ for the EHBS is computed from the difference between the desired braking pressure and a pressure which is proportional to the recuperation torque T_{rec}.

2.4 Simulation Results

To analyze the performance of the resulting control systems, the EHBS with recuperation controller has been implemented in MATLAB®/SIMULINK® using the dSPACE Automotive Simulation Models (ASMs). The ASM library contains ready-to-use models like a multi-body system, a virtual driver and a road. We refer to the combination of the EHBS plant model and the ASM as the "Environment model", which can be used in conjunction with the controllers to conduct realistic testing. The time histories presented in the following figures were obtained by simulating a simple driving maneuver with the augmented ASM, where the driver controls the vehicle velocity by actuating the acceleration and brake pedal. Figure 6 shows the reference and the controlled variable of the EHBS. The control yields a very good reference and disturbance behavior, as indicated by fast and well- damped settling and the steady-state accuracy of the pressure control error.

Furthermore, the vehicle velocity and the battery state of charge as well as the braking torques as

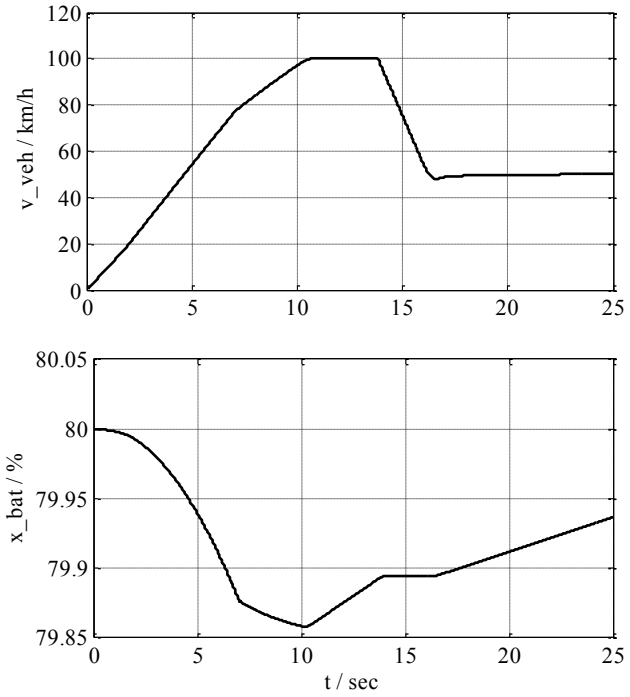


Figure 7: Simulated vehicle velocity and battery state of charge

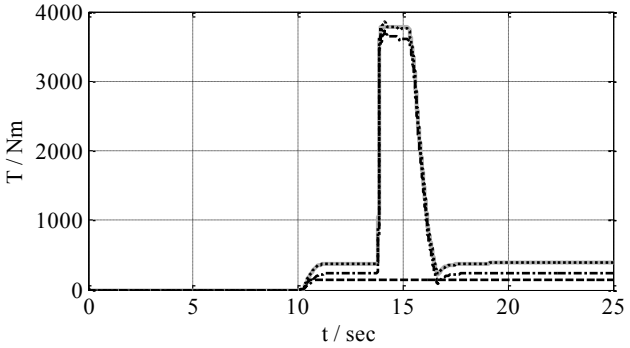


Figure 8: Simulated desired (gray, —), recuperated (black, --), friction brake based (black, -·) and total braking (black, ··) torques

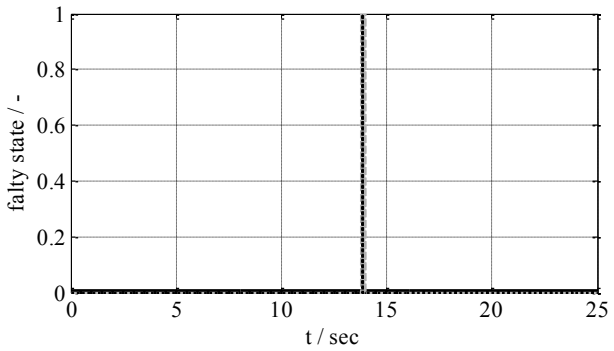


Figure 9: Simulated critical system variables of controller variable (black, —), control error (black, ··) and current gradient (gray, --)

shown in figure 7 and figure 8 confirm that recuperation takes place. The driver accelerates the vehicle for about 10 sec on a straight road, during which the battery state of charge decreases. After the vehicle velocity reaches 100 km/h, the driver starts braking to keep the velocity constant while driving downhill. This increases the battery state of charge. Before turning into a curve, the driver brakes harder to reduce the velocity to 50 km/h. Due to the hard braking the wheels lock for about 3 sec and no energy recovery takes place. On reaching the desired velocity, the driver releases the brake pedal almost entirely, the wheels unlock and the battery state of charge increases again. At the beginning of the braking maneuver, the desired braking torque is generated only via the recuperating drive motor. If the maximum recuperation torque is reached, the friction brakes generate an additional torque, so that the total braking torque, which is the sum of the recuperated torque and the friction brake-based torque, is equal to the desired braking torque.

2.5 Monitoring EHBS Behavior

One of the important tasks in the development of safety-critical vehicle components is a safety environment and the associated diagnostic functionalities for tracing critical system variables that might lead to erroneous system behavior if they deviate.

Following the principle described here, a simple diagnostic functionality was developed for the EHBS and implemented in ASM. The diagnostic monitor has the task to continuously detect the controller variable and the control error to ensure the proper functioning of the EHBS and the current gradient of the EHBS actuator for a stable vehicle onboard power system. Figure 9 shows the time histories of the diagnostics manager for the driving maneuver described in section 2.4. Due to the hard braking at about 14 sec, the monitoring of control error and the current gradient becomes momentarily active because the relevant thresholds are exceeded.

The processing and handling of the information collected from the diagnostic monitors now depends on the overall system implementation from the perspective of an ECU.

3. Developing the AUTOSAR ECU Software including DEM

The previous section shows how the control algorithms are developed in the control engineering domain with the domain-specific tool MATLAB/Simulink. These algorithms impose a high level of abstraction compared to the final ECU software. A simulation on this level cannot reveal errors caused by conversion to production code, e.g., fixed-point arithmetic-related defects like quantization errors or overflows. Furthermore, the correct behavior of the controller together with basic software parts like a DEM cannot be verified. The DEM receives erroneous controller states and therefore plays an important role in a robust ECU software design. Traditionally, control engineers did not need to worry about this functionality, which is regarded as the responsibility of the system architect, who integrates the controller software. However, to prevent errors and thus iterations in a later development step, control engineers and system architects alike should consider these aspects of the basic software early in the process.

The following subsections first present the dedicated methodology and tool chain that is used to get to the AUTOSAR ECU software, which also includes the necessary interfaces for the DEM. After that the model-based transition from the EHBS and recuperation controller in Simulink to a virtual ECU (V-ECU) is described. A V-ECU is the software that emulates a real ECU in a simulation scenario. It comprises components from the application and the basic software, and provides functionalities comparable to those of a real ECU.

3.1 Methodology and Tool Chain

This section gives an overview of the methodology and tool chain for testing controller software, including the diagnostics-related basic software that is relevant for function development.

The tool chain and basic methodology are shown in figure 10. In step 1, the behavior of the controller and the environment are modeled in Simulink (depicted at top left). Note that this environment model can be used throughout the whole development process, including later hardware-in-the-loop (HIL) simulation scenarios, without any modification. In this step the control engineer uses model-in-the-loop (MIL) simulation, which was already described for the EHBS system in section 2.4.

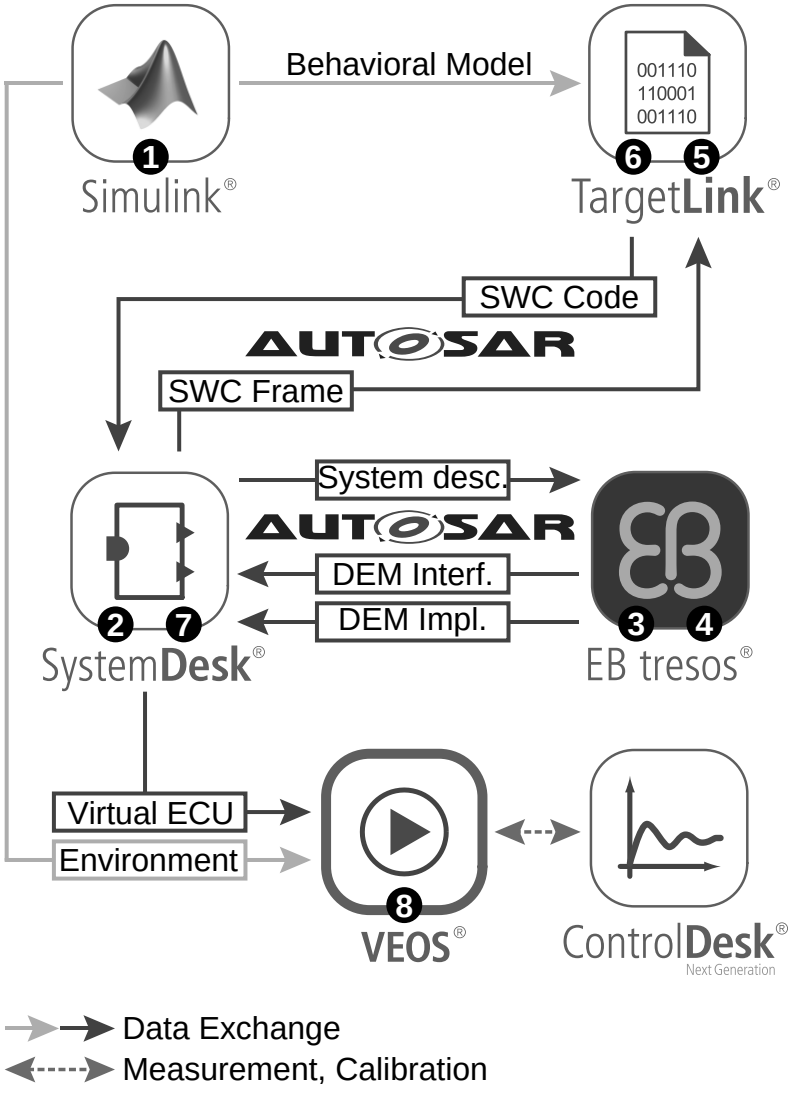


Figure 10: Methodology and tool chain

In parallel to the behavioral modeling, an AUTOSAR system [3] is designed using dSPACE SystemDesk® (step 2). Here, the developer takes the role of the system architect and creates the architecture that includes the software components with ports, interfaces, and the configuration of the runnable entities. Each runnable contains the implementation of a particular functionality. Later, the runnables will contain or point to the generated code files from TargetLink®. However, at this point, there are no code files attached yet. In addition to the runnables that link to the actual controller functionality, the monitoring functionality has to be considered. Each monitoring function requires a dedicated monitor runnable and a set of communication ports,

which are later connected to the DEM service component. The necessary interface descriptions for these ports are supplied as a template by EB tresos AutoCore (step 3 in figure 10). After adding the runnables and the ports, the developer creates a system and maps the application software components onto an ECU instance.

In step 4, the system description containing the software architecture must be exported from SystemDesk into EB tresos Studio, which is Elektrobit's configuration tool for their AUTOSAR basic software. Here, the DEM is configured to match the desired number of monitor runnables. EB tresos Studio is used to generate the DEM service component, which allows client-server communication with the application software components via the run-time environment (RTE).

The system is then exported from EB tresos Studio back into SystemDesk. The actual implementation for the DEM is generated by the EB tresos Studio code generator. Note that a full-fledged configuration of the DEM is not needed at this point. An implementation that resolves the symbols that are needed for linking the code together is sufficient. The complete implementation, which interacts with other basic software like the Diagnostic Log and Trace (DLT), is not required in this early phase of development.

The next task is to obtain the production code and link it to the runnable entities. This can be achieved through a round trip between SystemDesk and TargetLink (depicted in the upper part of figure 10). First, the software architecture as an AUTOSAR description is exported and imported into TargetLink. TargetLink in turn generates a subsystem hierarchy containing a subsystem for each runnable entity (step 5). These subsystems form the containers for the implementation model. Their ports resemble the external interface according to the AUTOSAR communication parts (ports, interfaces, data elements), including those for the monitoring functions. This ensures that the runnable structure and the interface of the implementation model strictly adhere to those of the software architecture. Inside these subsystems the control engineer integrates the implementation model (step 6). In the project we use an automatism to transform the Simulink model from step 1 into the implementation (TargetLink) model. For details of this automatism, see [1]. From this model TargetLink generates the production code and automatically associates it with the corresponding runnable entities. The control engineer can now test the generated code for each component using software-in-the-loop (SIL) simulation in Simulink together with the environment model by comparing the results with those from the earlier MIL simulation.

The implementation for the functionality and monitoring, plus the AUTOSAR description, which now references this implementation, are exported and then imported back into SystemDesk. In SystemDesk the developer creates an ECU configuration (step 7 in figure 10). This defines the RTE connections between basic software and application software. An overview of the system is given in figure 11. It shows the relevant artifacts in the AUTOSAR layered architecture. Note that besides the DEM, other services, like the ECU state manager (EcuM), are in fact mandatory for any later simulation. However, they can be generated automatically by SystemDesk and do not require further attention here. Furthermore, a configuration of the operating system (OS) can also be generated automatically with SystemDesk, provided that the runnables have been configured with the necessary events in step 1. This includes the mapping of runnables to tasks as well as the configuration of timers and alarms, so that the correct scheduling of the tasks that execute the runnables is ensured.

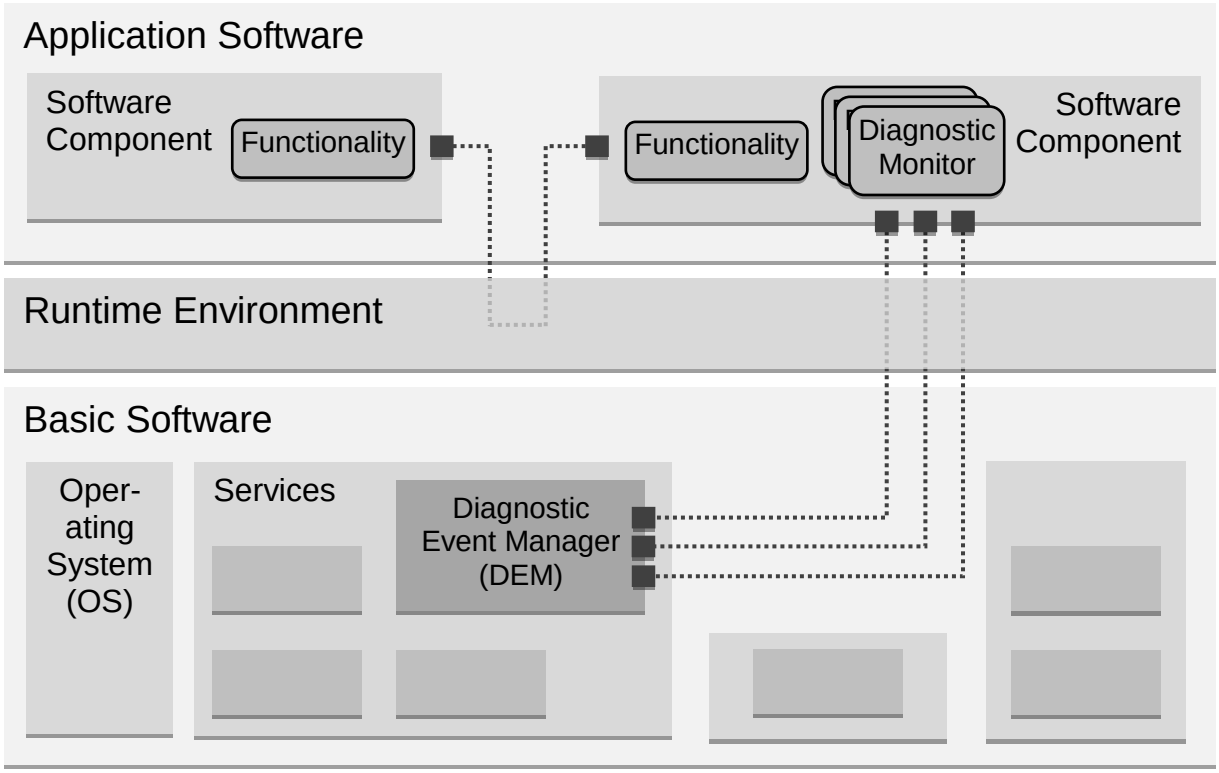


Figure 11: The interaction between DEM and application software

The developer also connects the as yet unconnected ports of the application software to the virtual I/O, which represents the interface to the environment model.

Next the RTE and the implementation for the automatically generated services mentioned earlier can be generated by SystemDesk. Together with the production code from TargetLink and the DEM implementation from EB tresos AutoCore, this forms the implementation for the V-ECU. This V-ECU can now be simulated using the VEOS® simulation platform (step 8 in figure 10) with the environment model used in steps 1 and 5 in figure 10.

The advantages of simulating a V-ECU compared to MIL simulation in Simulink (step 1) or SIL simulation in TargetLink (step 5) are as follows:

- Simulation of an RTE, including the respective events
- Interaction with basic software via client-server interfaces – here, especially the communication between monitor runnables and the DEM can be tested before the basic software is completely implemented (frontloading of tests).
- Requirements regarding the scheduling of the control functions can be verified in the context of an actual OSEK operating system.
- Experiments can be conducted using typical HIL experimentation and testing tools (e.g., dSPACE ControlDesk®) as the V-ECU includes measurement and calibration variable descriptions according to the ASAM MCD-2 standard and allows communication according to ASAM MCD-1 (XCP).
- In terms of communication delays between the controller software and the environment model, the simulation behavior is closer to that of a HIL simulation. Thus, the control engineer gets a better picture of the control algorithm's stability in a later HIL context.

In the following section the methodology presented above is applied to the Electro-hydraulic Brake System presented in section 2. Note that the focus is on the integrating of diagnostics. A more general description of the application can be found in [1].

3.2 Transforming Controllers in Simulink into a Virtual ECU

Following the methodology described above, the first step towards a V-ECU is to model the controller behavior in Simulink and the software architecture in SystemDesk (step 1 and step 2 from the methodology part). For diagnostics, the part of interest is the creation of the monitor runnables, which later contain the monitoring functionality. This is done in SystemDesk. In the case of the brake pressure controller, the monitor runnables are named *Monitor_MotorTorqueLimit*, *Monitor_ControlErr* and *Monitor_MotorCurrentGradient.* The interface to the DEM is obtained from EB tresos AutoCore (step 3 in the methodology). The resulting architecture is shown in figure 12. The interface to the DEM can be seen at bottom right ("Monitor_CE", "Monitor_MCD" and "Monitor_MTL").

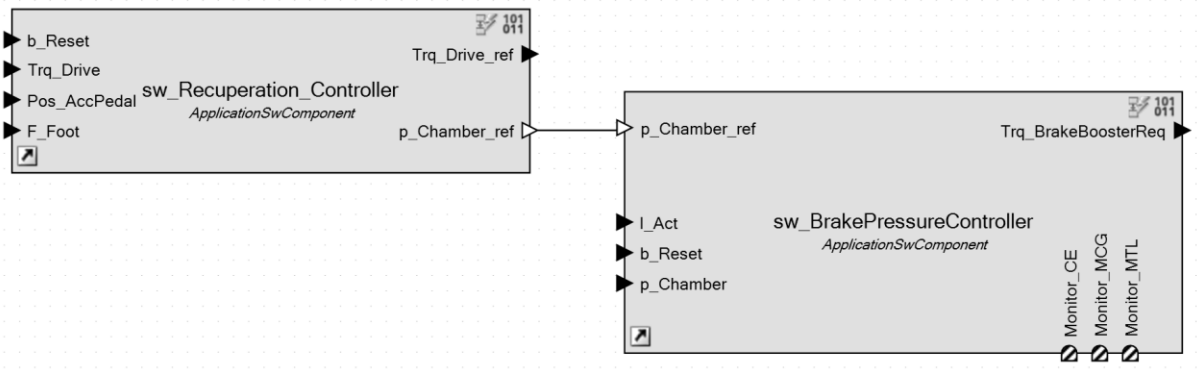


Figure 12: The EHBS controller architecture in SystemDesk

3.2.1 Configuring the DEM

To obtain the required interface from EB tresos Studio, the DEM basic software module must be configured. The DEM service component is generated from this configuration in a process described below.

First, a list of events must be specified in the DEM event configuration list. Each event in the event list defines a communication interface between the DEM basic software module and the monitor runnable at the applications software layer. Three monitors thus require three events. This is shown by figure 13.

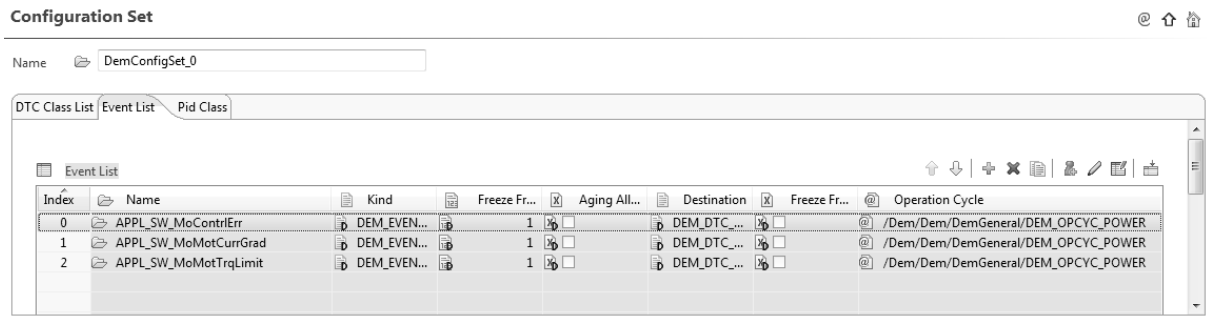


Figure 13: Creating DEM events for the brake pressure controller (EB tresos Studio)

When the DEM events, the DEM software components have been configured, the code files for the DEM are generated.

Later, in the context of RTE configuration (step 7 in the methodology) in SystemDesk, the brake pressure controller can be connected to the DEM service, which is shown in figure 14.

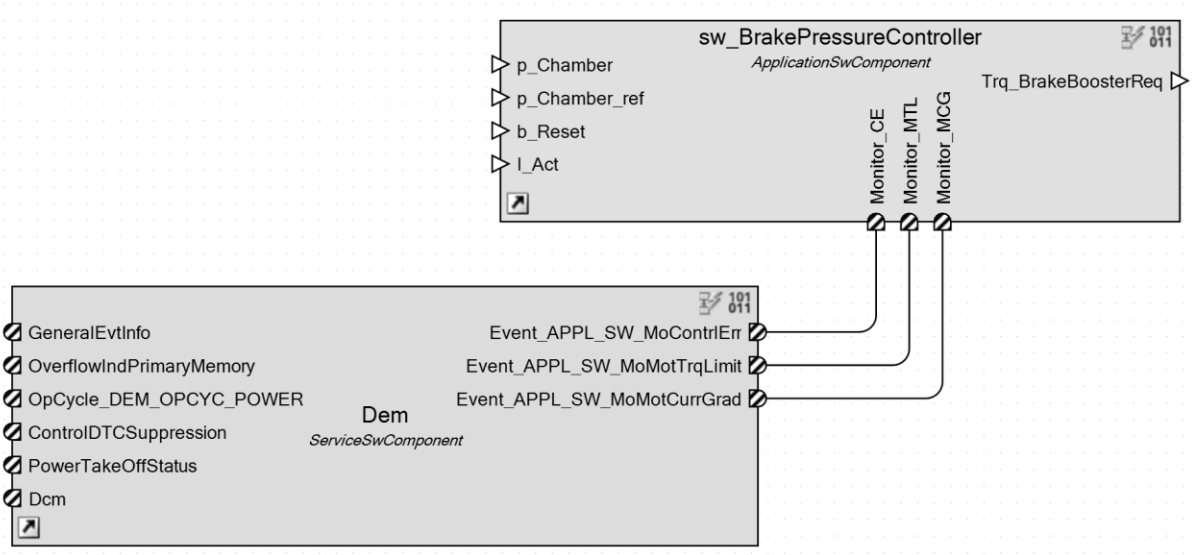


Figure 14: The connected brake pressure controller and DEM in SystemDesk

The software architecture is now ready to be exported to TargetLink, where the behavior of the monitor runnables is specified.

3.2.2 From Behavioral Model to Software Architecture and Back Again

So far the functional behavior on application level has been implemented in Simulink, a basic AUTOSAR system has been set up and the DEM has been generated. What is still missing is the implementation of the runnables including the diagnostic monitors.

The software architecture, which includes the definitions of the monitor runnables, is therefore exported from SystemDesk and imported into TargetLink to integrate the implementation and generate the production code (step 5 and 6 in the methodology). In our project we already had an implementation model of the recuperation controller without monitoring capabilities. By using the update mechanism supplied by TargetLink, the subsystems representing the monitor runnables can be generated into an existing implementation model. So we just had to implement the error detection functionality, which is described in chapter 2.5. The resulting controller hierarchy, including the environment model, is depicted in figure 15 with the three subsystems for the monitor runnables highlighted in the bottom right corner. Afterwards, the code for the monitor runnables is generated. Both control algorithm implementations are tested using SIL simulation in TargetLink. The results (see figure 16) show that the generated code's behavior corresponds to that of the Simulink model.

The resulting architecture is exported and imported back into SystemDesk. Now all the artifacts required for generating a V-ECU with DEM for a recuperating electrohydraulic brake controller are available.

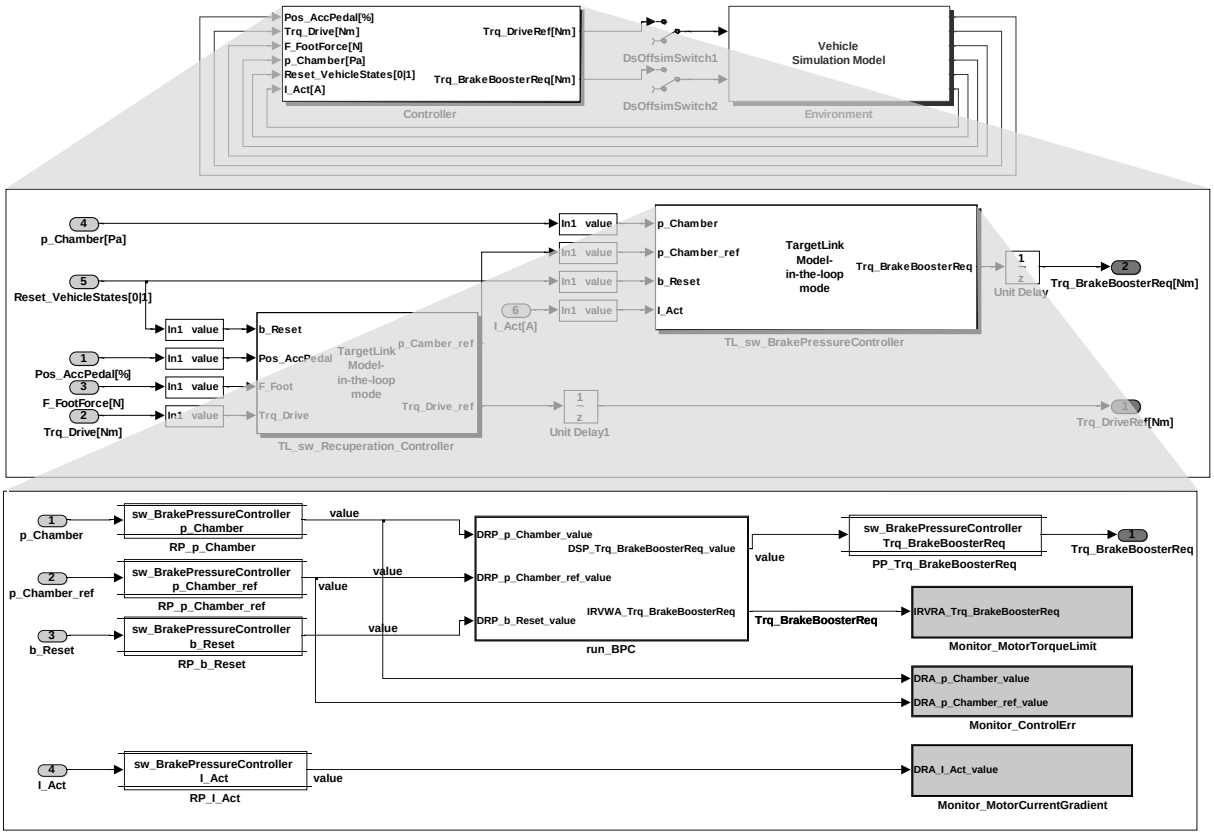


Figure 15: Controller hierarchy including the environment model

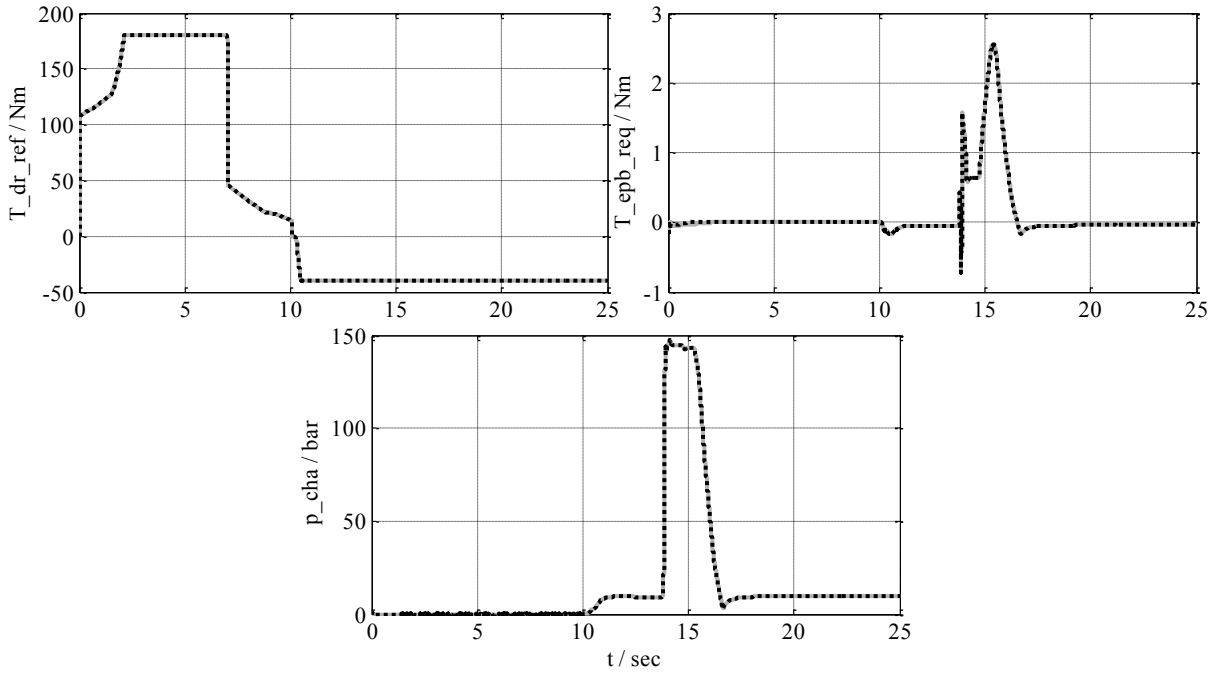
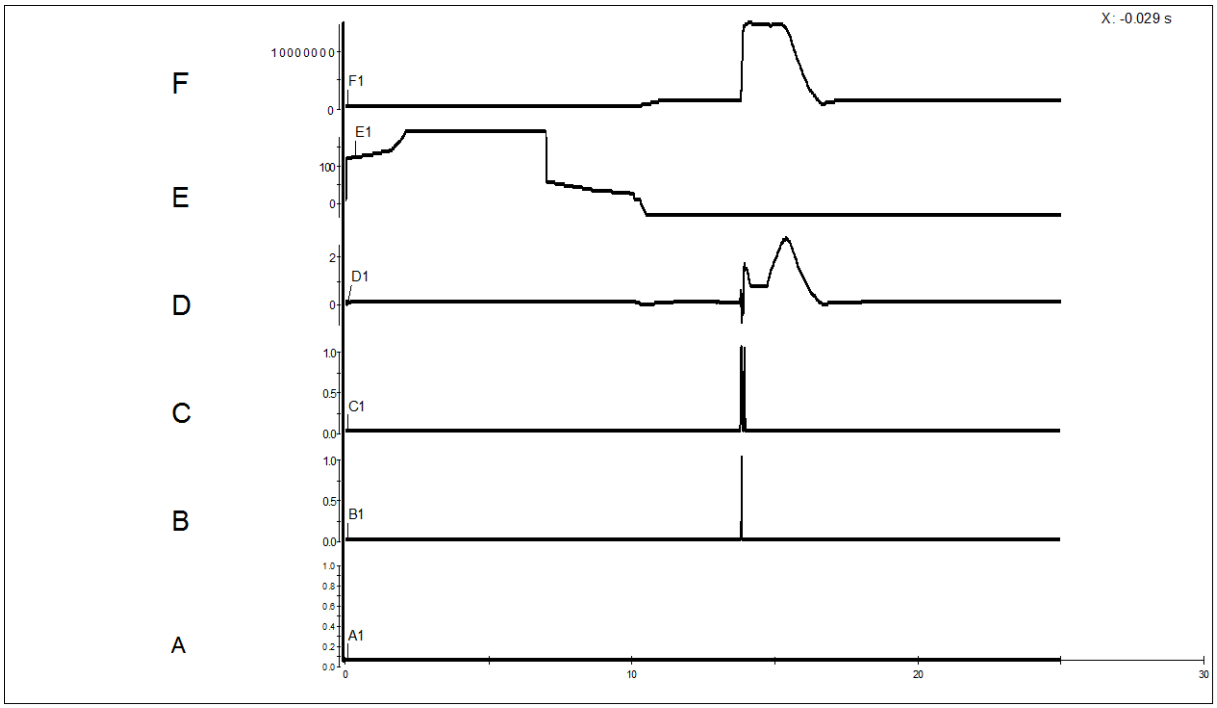


Figure 16: Simulated reference signal of drive motor, control and controlled variable with floating (gray, —) and fixed- point arithmetic (black, ··)

3.2.3 Simulating the V-ECU

To analyze the behavior of the controller software in combination with the basic software, a V-ECU is built as described in section 3.1. The ECU is connected to the environment model previously used for MIL simulation (section 2.4) and SIL simulation (section 3.2.2). The unconnected ports in the software architecture (the black triangles in figure 12) form the external interface of the V-ECU. These ports are connected to the environment model via virtual I/O (IoHwAb module in SystemDesk).

Figure 17 shows that the resulting signals of the V-ECU simulation match those of the SIL simulation in TargetLink (figure 16). It shows that the communication with the DEM is configured correctly and behaves in the same way as in the MIL simulation (compare figure 9 with plots A, B and C). It also verifies the correct configuration of the operating system, the AUTOSAR specification of the controllers' interfaces and communication via the RTE.



ID	Color	Name	Platform / Device	X-Axis Offset
F1		In_Read_sw_BrakePressureController_p_Chamber_value	XCP	0
E1		Out_Write_sw_Recuperation_Controller_Trq_Drive_ref_value	XCP	0
D1		Out_Write_sw_BrakePressureController_Trq_BrakeBoosterReq_value	XCP	0
A1		Monitor_MotorTorqueLimit	XCP	0
B1		Monitor_MotorCurrentGradient	XCP	0
C1		Monitor_ControlErr	XCP	0

Figure 17: Results of the V-ECU simulation in ControlDesk

4. Results and Future Work

This paper presents an approach to develop and simulate a controller for an electro-hydraulic brake system up to production code level on a virtual AUTOSAR ECU including basic software modules. The designed controllers show the expected behavior. The EHBS controller is accurate in the steady state, and the transient pressure errors are rapidly compensated. A monitor component is also integrated to measure faulty controller behavior, which can be evaluated and reacted to later on in the Diagnostic Event Manager.

The transition to the V-ECU is supported by a tool chain that allows control engineers to design and test a complex controller on different levels of abstraction using simulation. Extending the tool chain with EB tresos facilitated the integration of the DEM.

By using the presented methodology the number of possible errors during later integration into an existing system is reduced, as the later AUTOSAR system is already considered in the early functional design phase of the controller. Thus, the number of iterations required between the system architect and the control engineer will be lower as well. This effectively saves development time and costs.

This work completes the tool chain that has been developed in the context of the E-Mobil project. The paper also concludes the results of a series of publications: In a first step, we defined a methodology for a seamless development and testing process of controllers from the functional behavior to the final AUTOSAR software architecture using offline simulation [4]. In a second step we applied our methodology to a first example system, a simple recuperation-ready brake controller [2]. Then we applied our methodology to a recuperative electrohydraulic brake system developed in the E-Mobil project, which can be considered a real-world example. Here, we made a first attempt to integrate an EcuM basic software module to show the general approach of integrating basic software in our AUTOSAR-architecture [1]. Finally, in this publication, we integrated an externally generated basic software module with EB tresos Studio from EB tresos AutoCore. We showed the functionality of the Diagnostic Event Manager (DEM) as well as its configuration and generation, thereby demonstrating how EB tresos integrates into the tool chain.

Literature

[1] E. Farshizadeh, H. Briese, S. Beringer, D. Holler und L. Stockmann, “Design and Analysis of a Controller for Electrohydraulic Brake Systems on AUTOSAR Architecture Level,” *Stuttgarter Symposium,* 2013.

[2] K. Krügel, L. Stockmann, D. Holler und K. Lamberg, “Simulation-based Development and Testing Environment for Electric Vehicles,” *IAV,* 2012.

[3] AUTOSAR, “http://www.autosar.org/download/R4.1/AUTOSAR_TPS_SystemTemplate.pdf,” [Online].

[4] L. Stockmann, D. Holler und D. Spenneberg, “Early Simulation and Testing of Virtual ECUs for Electric Vehicles,” *Electric Vehicle Symposium 26,* 2012.

Acknowledgements

EUROPÄISCHE UNION
Investition in unsere Zukunft
Europäischer Fonds
für regionale Entwicklung

The project “Simulationsgestützter Entwurf für Elektrofahrzeuge” is funded by the European Union and the state of Northrhine-Westphalia (grant No. 64.65.69-EM-1008A).

Autoren / The Authors:

Dipl-Ing. Emad Farshizadeh, DMecS GmbH & Co. KG, 51105 Köln

Dipl-Ing. Hermann Briese, DMecS GmbH & Co. KG, 51105 Köln

Steffen Beringer, M. Sc., dSPACE GmbH, 33098 Paderborn

Dipl-Inf. Dominik Holler, dSPACE GmbH, 33098 Paderborn

Dipl-Ing. Lars Stockmann, dSPACE GmbH, 33098 Paderborn

Hamza Qadir Raja, M. Sc., dSPACE GmbH, 33098 Paderborn