Combining Real and Virtual Electronic Control Units in Hardware in the Loop Applications for Passenger Cars

Jan Röper, Arpita Bhattacharjee, Franz Kramer, Purushottam Kuntumalla, Andreas Junghanns

Combining Real and Virtual Electronic Control Units in Hardware in the Loop Applications for Passenger Cars

Jan Röper[1] Arpita Bhattacharjee[2] Franz Kramer[3] Purushottam Kuntumalla[2]
Andreas Junghanns[3]

[1]Mercedes-Benz AG jan.roeper@daimler.com
[2]Mercedes-Benz Research and Development India Pvt. Ltd.
arpita.bhattacharjee3@gmail.com purushottam.kuntumalla@daimler.com
[3]Synopsys GmbH {franz.kramer, andreas.junghanns}@synopsys.com

Abstract

For the testing of modern electronic control unit (ECU) software, different test-platforms like Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) but also in vehicle testing are used. To ensure a realistic behavior of the software in SIL and HIL testing, models of the environment as well as residual bus simulation are required. A detailed representation of other controllers in the vehicle network that feeds into the residual bus simulation is needed to test complex functions of the software and to close distributed control loops. We present a novel approach, where a virtual ECU designed for SIL testing is reused to generate the input to the residual bus simulation. For export from the SIL tool and import into the HIL modelling environment, a C-Code functional mockup unit is used. To show the benefit of this approach, the setup is compared to an existing setup, which uses both a simplified model of an ECU or the real ECU. In addition, drawbacks of the presented approach are presented and the potential of other methods is discussed.

1. Introduction

For the testing of modern electronic control units (ECU), multiple test environments exist. In the early stage, these include software in the loop (SIL) and hardware in the loop (HIL) testing. In SIL environments, the ECU code is executed on a standard PC, which requires virtualization of the ECU [1]. In HIL testing, the ECU code is executed on the target hardware, which is connected to a simulator for stimulating input signals and measuring output signals. In both cases, a complex simulation of the environment of the device under test (DUT) is required, which consists of plant models and a simulation of the communication also known as residual bus simulation (RBS). The RBS stimulates the communication interfaces of the DUT using both static fixed values as well as dynamic signals. This is necessary to ensure that the test conditions match the conditions in the real application, which in this case, is the control of an automatic transmission in a passenger car by a transmission control unit (TCU). Additional features of a RBS are the manipulation of signals, messages, message counters and various other mechanisms that control the data flow in an ECU network.

A major task in the development of a HIL test environment is to provide a mechanism to generate the dynamic signals for the RBS. One approach is to provide the dynamic signals through the test automation that executes test scripts. However, for complex controllers with multiple parallel control loops, this approach is unfeasible. An alternative can be simulated ECUs, where the basic functionality of an ECU is modeled in the same fashion as a plant. For an aggregate component HIL environment, multiple simulated ECUs provide the dynamic signals to the RBS to satisfy the DUT with adequate stimulus. While simulated ECUs allow for a more detailed representation of the DUT environment, their implementation effort is high: the ECU to be modeled has to be analyzed, the targeted functionality reengineered and the result tested on the HIL with the DUT. Additional challenges are the need for parameter management based on the specific variant of the simulated ECU and the incessant demand by testers to add further functionalities to the simulated ECU. In this paper, we present a novel approach where virtual and real ECUs are combined in a HIL simulator for component testing.

2. Previous Work

Because of the shortcomings of the simulated ECU concept, multiple approaches have been made to replace them with a more desirable design. Common goal of all these approaches is to reduce the engineering effort by reusing code or precursors of the code of the targeted real ECU. One approach is to import fragments of the models, which are the base for the ECU code into the HIL model development tools. While this approach is straightforward, it leaves the developer with the problem that the interfaces of these fragments do not directly correspond to the dynamic signals of the RBS. For proper integration, wrapper models are necessary. However, this approach faces the same challenges as the simulated ECU approach. In turn, the reuse of the code for smaller ECUs as discussed in [2] is carried out in the same fashion by building a wrapper around the ECU code. This process resembles the process of virtualizing ECUs for SIL applications mentioned above but is specific to the specific HIL environment. Another approach, presented by [3], employs a SIL that runs on a

standard PC in parallel to a HIL. The SIL communicates with the HIL and the DUT via bus systems. This represents a distributed RBS, where the part represented by the PC is not a hard real time system but runs in a standard Windows setup with additional tweaks that ensure acceptable stability. In this scenario, full reuse of an existing SIL reduces the effort for providing dynamic signals to the RBS. At the same time, this approach leads to more complexity, as the test automation has to control two environments and special mechanisms for manipulating signals in both RBSs are necessary. [4] discuss a similar scenario, where a virtual ECU is executed on a separate system, which communicates with a HIL and other test systems. A complex control system is presented that handles the resulting hybrid test system. This work does not state whether the virtual ECU running on a separate system is operating in real-time. Table 1 summarizes the above-mentioned approaches to generate dynamic signals for the RBS and their most relevant properties.

Table 1: Types of Sources for Dynamic RBS Signals

Signal Generation Type	Relevant Properties
script in test automation	limited to non-parallel control loops
simulated ECU	high effort for reengineering
reuse of ECU code fragments or precursors	wrapper for integration required
virtual ECU in a SIL on separate PC	distributed system with stability and automatization challenges
real ECU	replaces RBS; limited ability for manipulation; late availability

3. Proposed Approach

The approach presented in this paper combines the reuse of existing code in a SIL by [3], while maintaining the integrity of the RBS as described by [2]. A C-Code based virtual ECU of a powertrain control unit (PCU) that was originally designed for the usage on a Windows PC was adapted for platform independence. The effort and cost for adapting a virtual ECU for platform independence depend on the specific virtual ECU and can vary drastically. In addition, cost and effort are hard to estimate in advance and the complete virtual ECU has to be available as C-Code. The resulting platform independent virtual ECU represents a pretested black box for the HIL developer and can be parameterized for different variants before the export from the SIL environment.

The C-Code functional mockup unit (FMU), which was originally designed for plant model exchange, is chosen as a container for the exchange of the platform independent virtual PCU [5]. An important argument for using a C-Code FMU in this scenario is the support by many different simulator platforms. The ability of FMUs to be used as a container for ECU code by exporting a plant model to an actual ECU is shown in [6], while in [7] FMU is only used to import models into a HIL environment and a proprietary format is used to import controller code that is not further described. For future applications, the FMU for embedded systems (eFMU), as described by [8], can also be a potential container for the code of a virtual ECU. Like FMU, eFMU are designed for plant models as well but with the goal of integration in an embedded real time target. Specifically the features to add information that is relevant for the build process, e.g. target compiler options, can be useful when it comes to integrating a virtual ECU in a HIL environment.

After importing the virtual PCU in its FMU container into the build environment of the HIL application, all rest bus relevant signals are routed to the respective RBS model interfaces. The real PCU that is replaced by the virtual PCU uses CAN, Flexray and LIN as bus systems. An overview of the challenges faced when integrating the virtual PCU with the existing RBS model is presented below. Some of these challenges result from the characteristics of the RBS models, while others result from the characteristics of the different bus systems.

4. Generation of Virtual ECU FMU

The virtualized PCU is a level two virtual ECU according to [9]. It contains the full application layer and a simulation basic software providing necessary functionality to increase test coverage of the software under test. The tool for creating and simulating the virtual ECU is Synopsys Silver®, a virtual ECU tool available for Windows and Linux PCs. Notable components of the simulated basic software layer are signal and PDU based COM for multiple bus systems including network routing, NVM for nonvolatile memory and a replacement OS. This level of abstraction of this ECU was chosen as it provides high complexity with multiple bus systems and large target code size in application layer, without involving multiple stakeholders in the code to be virtualized. Integrating target code deeper in the ECU stack should pose little additional challenge provided it is microcontroller independent.

The move from a PC simulation to a platform independent virtual model can be seen in Figure 1. The challenge during FMU generation is removing dependencies on the compiler and linker toolchain as well as automatically generating platform independent C-Code for all required features and components provided by Synopsys Silver®. Developing the methodology initially required in depth compiler knowledge as well as a profound understanding on modern operating systems and CPU architectures. Once the tooling was developed subsequent updates of the target software or the simulation basic software require little effort.

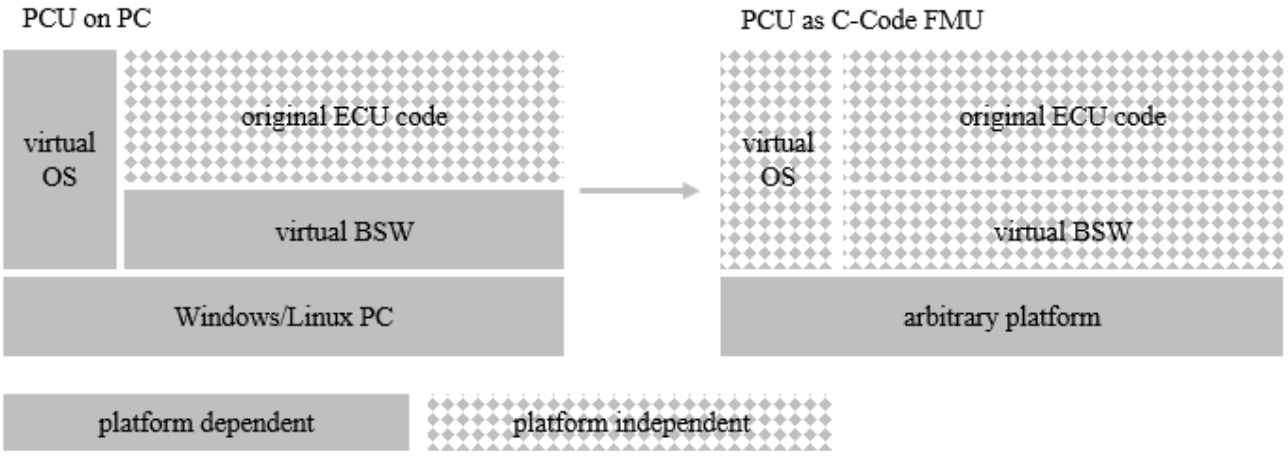

Figure 1: Porting a PC based simulation to be platform independent.

While the approach is conceptionally straight forward and the project was successfully integrated, the team gained key insights which influenced the initial favorability assumptions. The use of open C-Code can be problematic when integrating intellectual property from multiple parties. This issue was limited by keeping the number of involved parties small but must be kept in mind when applying this method to future projects.

There are mandatory virtual ECU features which cannot simply be provided as platform independent code. One example is the parametrization of ECU CHARACTERISTICs using production parameter and A2L files. Exporting such a feature as C-Code requires that certain assumptions like endianness or type sizes are known about the platform the FMU is later executed on. This is natural as this information is also required by industry measurement tools and is encoded in the ECU A2L. Removing such a feature to achieve true platform independence would drastically reduce the useability of the resulting FMU, so these assumptions were verified on the HIL platform and introduced into the export process. Doing so voids the platform independence of the FMU, as it is now compatible to the two specific PC and HIL platforms.

The use of pure source code to become compiler/linker independent shifts the build process to the HIL environment. To minimize the frequency of feedback loops related to the build step, the compiler toolchain of the HIL and the PC were aligned, as it is available for both platforms. Even then it is only possible to functionally verify the FMU built for PC without consuming HIL resources. A high-quality acceptance test on PC kept the number of functional errors and the time spent validating the FMU on the HIL system to a minimum. However, even validation using source level debugging on the PC system only assures the quality of the PC-built binary and due to the mandatory rebuild on the HIL it is possible that the different HIL platform introduces side effects.

In general, there are not many hard limitations. One is the computing power of the HIL system. In this case it was a multi core HIL x86 CPU and the FMU runs on a single HIL core without encountering task overruns. The second limitation is some required compatibility between the HIL and PC architecture, as true platform independence would have only been achieved with an unreasonable amount of engineering. When designing the complexity of such a virtual ECU export to the HIL it is advisable to evaluate level of detail versus the potential gain to avoid lengthy and costly iterations in the HIL environment.

5. Integration

In order to integrate the virtual PCU in the HIL model, the FMU is imported into the build environment. In our setup the build environment consists of two main components: 1. The tool Configuration Desk® that is used to configure the dSpace® HIL hardware, the task configuration and the signal routing between plant models or FMUs and the IO of the HIL and 2. The modeling tool Simulink® that is used to implement plant models as well as the RBS. Figure 2 shows the structure of the overall setup after integrating the virtual PCU. As the PCU only interacts through bus systems, only the parts relevant to the integration of bus systems is visualized in Figure 2. All other parts that a HIL system normally requires like power supplies and multi IO are omitted.

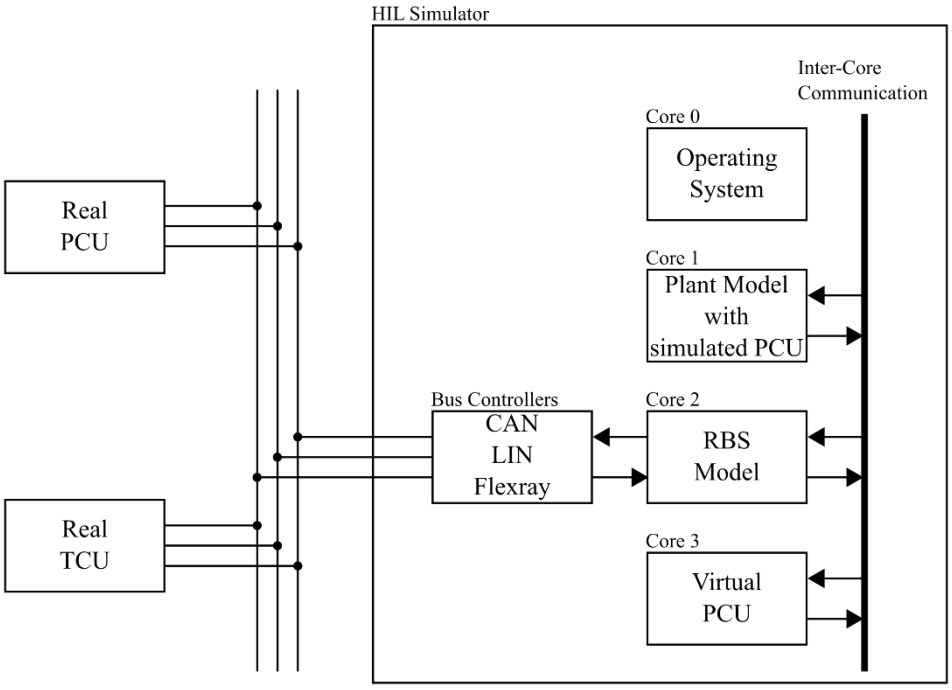

Figure 2: Integration of Real, Simulated and Virtual PCU in the HIL System with Focus on Bus Systems

Like the plant model as well as the RBS model and the operating system, the FMU is executed on a dedicated core of the HIL simulator's processor. The cores exchange signals through an inter-core communication mechanism. FMU do not allow calling portions of their code from different tasks. The virtual PCU integrated into the HIL setup handles task scheduling by using internal subtasks and is called by the HIL by an external 5 ms task. For applications with time critical tasks, a design that imports multiple FMU and calling them by separate tasks would be required. Note that the setup allows running either a simulated, a real or a virtual PCU with the same executable by configuring the model through the HIL automation. The TCU is always kept as a real component and is the DUT, as described above.

The signal routing for the three implementations real, virtual and simulated PCU integrated into the HIL model is fundamentally different and in the case of the simulated and the virtual PCU depends on the way the RBS is implemented. Figure 3 shows an example for the signal routing for all three PCUs in a minimal example. Only the different implementations of the PCU, the real TCU as well as an additional simulated ECU are displayed for the implementation of signal routing of CAN bus signals.

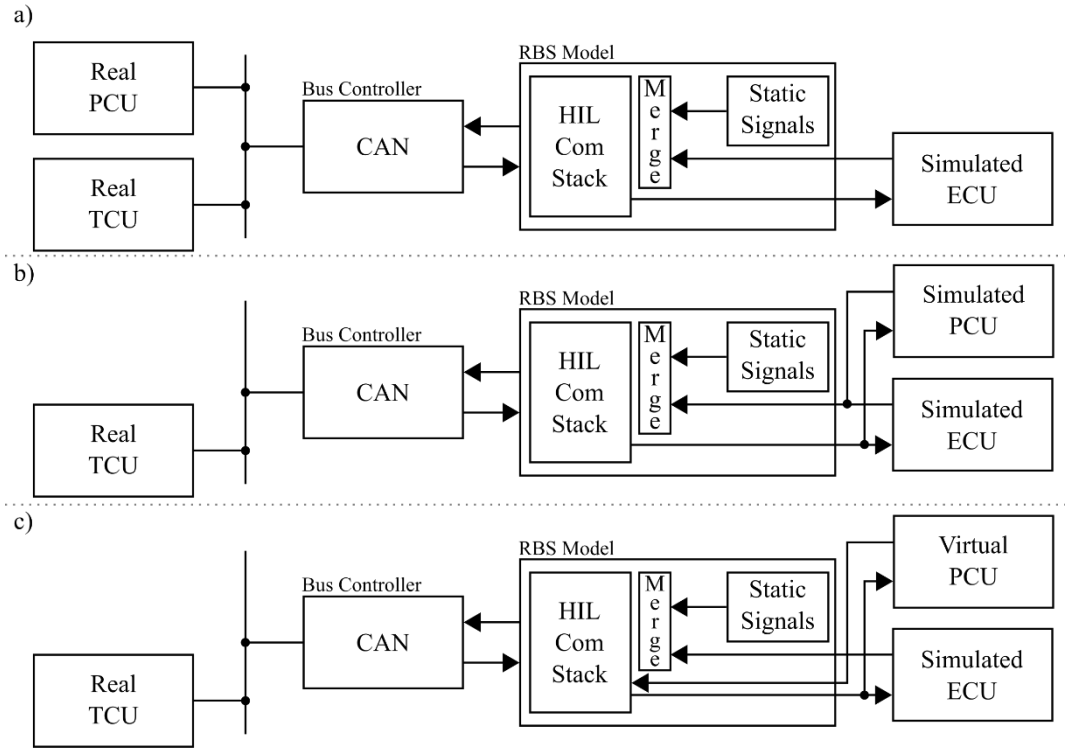

Figure 3: Signal Routing for a Setup with a) Real PCU, b) Simulated PCU and c) Virtual PCU

Figure 3 a) shows the case where the real PCU communicates with the real TCU and one additional simulated ECU. In this setup, three physical bus controllers are connected on the bus lines. The controller of the real TCU, the controller of the real PCU and the controller of the HIL RBS. In the case of the real TCU and the real PCU all signals are calculated in their respective application and are then transferred to the controller and vice versa. In the case of the simulated ECU, only a subset of the signals that require dynamic changes during the HIL simulation are calculated in the model of the simulated ECU. These signals are then merged with static default signals in the RBS and the resulting values are then transferred to the communication stack of the HIL and the HIL bus controller. The same is true for incoming signals from the bus. Only signals that are required for the computations in the simulated ECU are selected in the HIL communication stack for transfer to its model.

Figure 3 b) shows the case where the simulated PCU communicates with the real TCU and one additional simulated ECU. What was described above for the generalized simulated ECU is also true for the simulated PCU. Dynamic signals from the simulated PCU model are merged with static default signals, are processed by HIL communication stack and then sent on the bus via the bus controller. The switching between the setup from Figure 3 a) and the one from Figure 3 b) requires deactivating the power supply as well as activating the nodes for the simulated PCU in the HIL RBS. When modifying the physical setup of the bus, proper CAN termination must be ensured as well.

Figure 3 c) in turn shows the case where the virtual PCU communicates with the real TCU and one additional simulated ECU. Compared to the setup in Figure 3 b), all signals are computed by the virtual PCU and merging with static default signals is only required for the simulated ECU. As the virtual PCU runs on a separate core, signal routing to the Simulink® model containing the RBS is required. The RBS model implementation accepts signals of the type double and converts these internally to the required data type. As the virtual PCU provides all signals in the equivalent data type real, no additional handling of datatypes is required. While this simplifies the implementation, drawbacks regarding performance have to be accepted due to usage of unnecessarily expensive data types for some signals. The large amount of signals in a vehicle network requires an automated process to generate the signal routing parts of the model. The automated process requires naming of the virtual ECU input and output signals based on the description files of the bus systems. Due to differences between the implementations of RBS models for the bus types CAN, LIN and Flexray, both the signal routing as well as the merging with static default signals requires variants of the approach presented above.

6. Experiment and Results

The resulting HIL setup can execute tests with the newly integrated virtual PCU, the simulated PCU that allows manipulation but limited features and the real PCU. The DUT in all three scenarios is the TCU as mentioned above. To show the validity of our approach and to compare the three different implementations of the PCU, a test scenario is executed on the HIL. A common test scenario is to use a vehicle speed profile such as the worldwide harmonized light vehicles test procedure (WLTP) and to use a speed controller to set accelerator and brake pedals accordingly. This approach has the disadvantage that the speed controller can mask the differences in the behavior of the PCU on the vehicle speed level. To ensure that the differences in the behavior stay unmasked, a simple test scenario with a fixed sequence of inputs to accelerator and brake pedal is executed on the HIL. Figure 4 shows the stimulus as well as an example of the resulting rescaled dimensionless vehicle and engine speed.

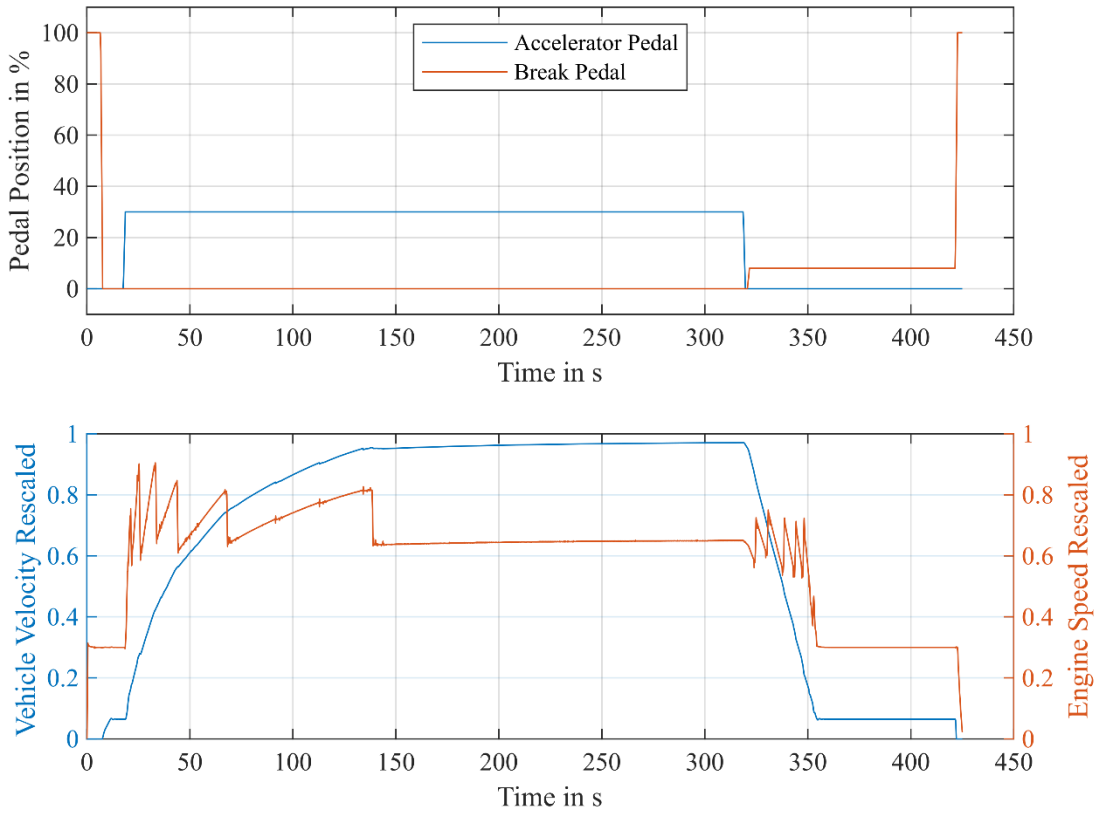

Figure 4: Stimulus and Reference Result for Comparison

At the beginning of the sequence, the combustion engine of the vehicle is started and the gear selector lever is set to drive. In the next step, the brake pedal is released and the vehicle starts to creep (t = 10 s). After reaching a steady creep velocity, the accelerator pedal is set to a constant value of 30 %, which leads to a rising vehicle velocity with moderate acceleration (t = 20 s). After a fixed wait time that is sufficient for the vehicle to reach a constant velocity, the accelerator pedal is released and the brake pedal is set to 5 % (t = 320 s), which causes the vehicle to reduce the velocity to creep velocity (t = 355 s). Finally, after another wait time, the vehicle comes to a stop by fully applying the brake pedal (t = 420 s). During the phases of accelerating and decelerating the vehicle, multiple up- and downshifts are visible in the engine speed profile. They stand out as steep decreases or increases of engine speed. The engine speed level at which a gearshift occurs is relevant for the overall powertrain strategy, as the shift levels affect the ability of the combustion engine to produce torque as well as its fuel consumption. The ECUs in the powertrain network negotiate the shift levels during runtime, which makes them ideal for a comparison of the performance of simulated and virtual PCU versus real PCU.

Figure 5 shows the rescaled dimensionless vehicle and engine speed for virtual, simulated and real PCU when executing the stimulus from above. The close matching of virtual and real PCU are visible, while the results of the run with the simulated PCU deviate. The reasons for these deviations are in the limited functionality of the simulated PCU as well as in a simplified variant dependent parameterization. The difference of the results with the simulated PCU become even more

apparent when comparing the shift points for upshifts while accelerating the vehicle. The reasons for the deviation in Figure 5 lie in the simplified representation of functions for torque coordination. This can be addressed by adding more detailed functions to the simulated PCU and parameterizing them as required for the specific variant. If this action is carried out until no deviations occur in any scenario, the effort for implementing the simulated PCU will match the effort for implementing the real PCU. The reuse of the code of the real PCU in a virtual PCU yields the same result with limited effort, which makes it much more attractive.

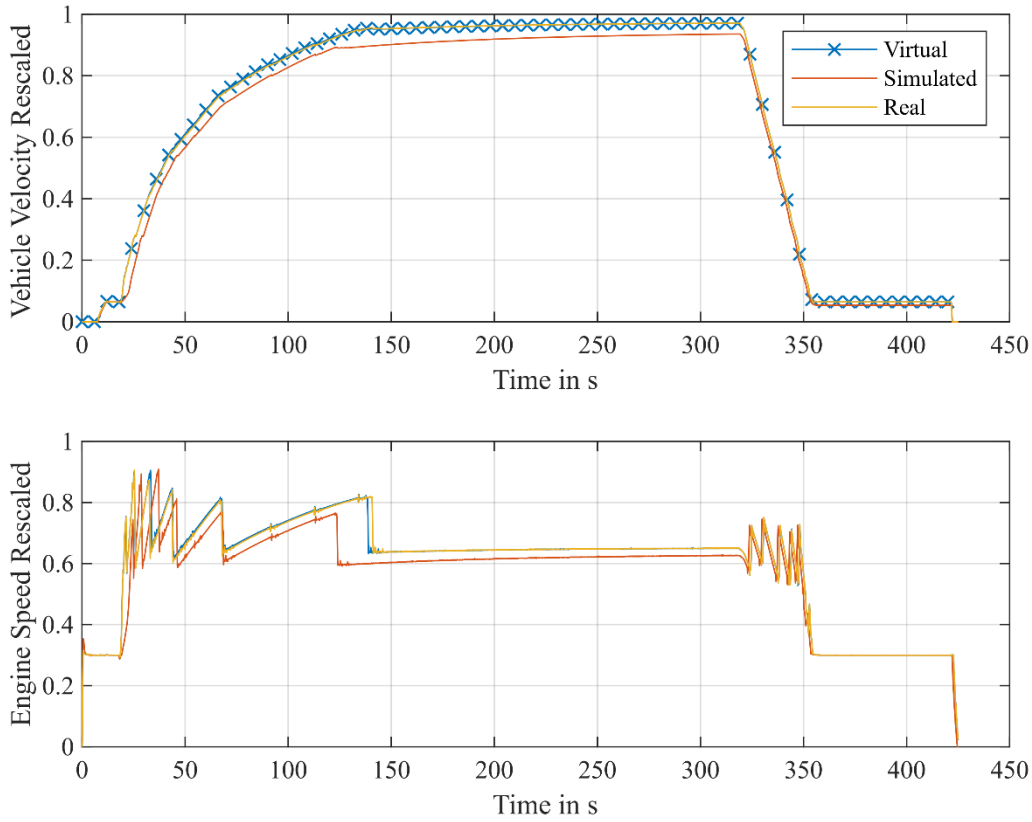

Figure 5: Results for Virtual, Simulated and Real ECU

For the further discussion, the results acquired with the simulated PCU are set aside and we focus on the deviations between the virtual and the real PCU. For both setups, three runs of the scenario shown in Figure 4 are carried out. Subsequently, the recorded vehicle speed is used to compare the fluctuations between runs for the same setup to assess the reproducibility, as well as the deviations between the setups to assess the accuracy of the representation of the real PCU by the virtual PCU. The scope of the scenario used covers the most important functionalities in a drive train and is useful for characterizing the accuracy of the representation. However, functionalities of the PCU software that are not active in this scenario cannot be assessed by this analysis.

Columns two and three of Table 2 show the root mean square errors (RMSE) acquired from the three runs with real and simulated PCU. The RMSE is calculated between first and second run, second and third run, as well as third and first run for each setup. The results show that the RMSE representing the variations of repeated runs with real PCU is higher than the RMSE for runs with virtual PCU. This is true both for all three calculated values of RMSEs as well as for the mean of the RMSEs shown in the last row of Table 2. On average, the RMSE of the vehicle speed calculated with real PCU with a value of 0.18 km/h is almost double the average RMSE of the one calculated with the virtual PCU with 0.1 km/h. This analysis itself only shows that the setup with real PCU varies more between repeated executions than the setup with virtual PCU. It does not give any indication of the accuracy of the representation of the real PCU by the virtual PCU.

Table 2: RMSE of Vehicle Speed for repeated Executions with Real and Virtual ECU

Run	real PCU RMS Vehicle Speed [km/h]	virtual PCU RMS Vehicle Speed [km/h]
1 vs. 2	0.22	0.09
2 vs. 3	0.18	0.09
3 vs. 1	0.13	0.11
mean	0.18	0.10

Reasons for the higher fluctuations between runs with real PCUs become apparent when comparing the nature of the two setups. In the setup with the real PCU, the code runs on a separate processor with its own clock. Drift and jitter between the clocks of the HIL and the PCU are one example for a source of deviation if they are not compensated for by special methods [10], which is not the case in our setup. In addition, the communication via bus systems like CAN is not deterministic and the dataflow between the real PCU and the TCU can vary between executions. The same is true for other I/O interfaces. For example, analog I/O with noise can introduce additional variance in the behavior. Most of the abovementioned effects are absent for the virtual ECU, as it runs on the HIL processor and does not possess any real I/O. The remaining sources of variance between the runs are the CAN communication with the TCU, the loopback through the communication controllers of the HIL, as described above, as well as the fluctuations introduced by the TCU. In summary, the setup with a virtual PCU shows a better reproducibility than the setup with a real PCU, which satisfies the minimum requirement that using a virtual PCU may not negatively affect the reproducibility.

Table 3 lists the RMSEs calculated between runs with real and virtual PCU and allows us to assess the accuracy of the representation of the real by the virtual PCU. The RMSEs in this assessment are in the same range as the RMSEs for repeated runs of the real PCU in Table 2. This shows that the error introduced by using a virtual PCU instead of a real PCU is below the level for repeated executions of the real PCU. In other words, the difference between one test execution with a real PCU and one test execution with a virtual PCU does not exceed the difference witnessed when executing a test twice with the real PCU. This means, that for this specific implementation and scenario, the virtual PCU suffices the reproducibility as well as accuracy expectations if the reproducibility with the real PCU is satisfactory.

Table 3: RMSE of Vehicle Speed for Comparison between Executions with Real and Virtual ECU

Run	real vs. virtual PCU RMS Vehicle Speed [km/h]
1	0.22
2	0.10
3	0.19
mean	0.17

The data used for the discussions in the last paragraphs is both limited in the number of datasets and in its variation. For a more detailed analysis, more executions with more complex scenarios are required. However, the limited analysis already shows that a high level of reproducibility, as well as accuracy can be achieved when replacing real ECUs with virtual ECUs.

7. Summary and Outlook

In this work, we were able to show, that virtual ECU can be reused to enhance dynamic inputs to a HIL residual bus simulation and to thus enable a more realistic representation of the DUT environment. The results obtained with this novel approach are not distinguishable from results obtained when using a real ECU instead. Regardless of the good results during runtime, some drawbacks remain during the implementation phase of the HIL executable. These include the large

number of signals that have to be routed to the RBS, the workarounds necessary in order to combine the virtual ECU signals with signals of other simulated or virtual ECUs, as well as the complex process to make the C-Code FMU buildable in the modelling environment of the HIL simulator.
While the first two items can only be addressed by completely automated signal routing and by redesigning the model structure, the latter requires a change in the process of the generation of the virtual ECU in a FMU container. In the most recent version 3.0 of FMU that will be released in 2022, additional features for specifying the process for compiling and linking of the sources will be available [11]. Furthermore, other new features like events for the triggering of sub functions can be useful for extending the approach to virtual ECUs that are more complex. With the current tool chain, only FMUs of the Version 2.0 are supported and the new features should be evaluated once full support for FMUs of the version 3.0 is available.
A different approach to address the build issues discussed above can be to work with precompiled object files, which need to be compatible to the x86-based platform and compiler of the HIL modeling environment. In addition to moving the compile process out of the complex HIL modeling environment, this change in the process also ensures protection of intellectual property. Particularly larger ECU software projects, where supplier and OEM software modules are combined to create the ECU software, can benefit from this.

Acknowledgments

This project was only possible with the continuous support of a large group of engineers. Many thanks go to Muralidhar Rajaram, who supported in commissioning the modified project on the HIL simulator and to Gyaneshwar Singh for his support with acquiring the data for offline analysis. For the modification of the network model and their creative approaches to the problems arising, we thank Ashutosh Singh and Felix Maier.

References

[1] A. Junghanns, R. Serway, T. Liebezeit and M. Bonin, "Building Virtual ECUs Quickly and Economically," *ATZelektronik worldwide*, pp. 48-51, 01 06 2012.

[2] C.-F. Nicolas, I. Ayestaran, T. Poggi, G. Sagardui and J.-M. Martin, "A CAN Restbus HiL Elevator Simulator Based on Code Reuse and Device Para-Virtualization," in *IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, Toronto, Canada, 2017.

[3] I. Matheis, T. Dörsam and W. Hoffmann, "Integrating a SiL into a HiL Test Platform," in *Simulation and Testing for Vehicle Technology, 7th Conference*, Berlin, 2016.

[4] A. Himmler, L. Stockmann, S. Walter and S. Laux, "Developments Targeting Hybrid Test Systems for HIL Testing," in *AIAA SciTech Forum*, Sandiego, California, USA, 2019.

[5] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz and S. Wolf, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models," in *Proceedings of the 8th International Modelica Conference*, Dresden, 2011.

[6] C. Bertsch, J. Neudorfer, E. Ahle, S. Arumugham, K. Ramachandran and A. Thuy, "FMI for Physical Models on Automotive Embedded Targets," in *11th International Modelica Conference*, Versailles, 2015.

[7] A. Himmler, A. Pillekeit, B. Loyer and V. D. Stéphane Mouvand, "Using FMI- and FPGA-Based Models for the Real-Time Simulation of Aircraft Systems," in *AIAA Modeling and Simulation Technologies Conference*, Kissimmee, Florida, USA, 2018.

[8] "Functional Mock-Up Interface for embedded systems (eFMI)," [Online]. Available: https://emphysis.github.io/pages/downloads/efmi_specification_1.0.0-alpha.4.html. [Accessed 02 09 2021].

[9] "Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs)," prostep ivip association, Darmstadt, 2020.

[10] M. Akpınar, E. G. Schmidt and K. W. Schmidt, "Drift Correction for the Software-based Clock Synchronization on Controller Area Network," in *IEEE Symposium on Computers and Communications (ISCC)*, 2020.

[11] "Functional Mock-up Interface Specification, Version 848f13a," [Online]. Available: https://fmi-standard.org/docs/3.0-dev/. [Accessed 21 12 2021].