Virtual Test Method for Complex and Variant-Rich Automotive Systems

Andreas Lauber, Houssem Guissouma, and Eric Sax
Karlsruher Institute of Technology (KIT)
Institute for Information Processing Technologies (ITIV)
76131 Karlsruhe, Germany,
{lauber, houssem.guissouma, sax}@kit.edu

***Abstract*— The fast development of embedded automotive systems in form of connected Electronic Control Units (ECUs) has led to complex development processes. Especially for safety-critical functions, the testing activities are essential to check if the designed system complies with the requirements.**

Nowadays, the continuous development of mobile electronic devices through software updates is performed almost on a daily basis. This trend is now starting to be observed in cyber-physical systems with higher safety priorities. In the automotive field, the rising software portion in the vehicles and the shortening technology life-cycles are accentuating the need for Software Over The Air (SOTA) updates. Despite the opportunities offered by SOTA updates, the current test processes and methods must be adapted to manage the resulting complexity throughout the life-cycle of the vehicles. Especially the typical variants abundance in automotive product lines is considered as an important challenge, which cannot be solved only by "classical" testing methods such as Hardware-In-the-Loop.

In this paper, we present a testing method for variant-rich systems, which can be applied for automotive software updates. It uses virtual platforms for automated delta testing to handle the abundance of system configurations. Virtual testing is introduced as a powerful tool to reduce the amount of real tests and allow efficient variants verification. As a proof of concept, an Adaptive Cruise Control (ACC) composed of two ECUs has been implemented both in real hardware and using a virtual platform. With this approach, virtual delta tests, i. e. specific test-benches targeting the differences to a basic variant, can be rapidly executed for various system configurations. To prove the feasibility of the presented test method in more complex systems, a scalability study has been conducted.

I. Introduction

Nowadays, modern premium cars have more than one hundred micro-controllers on-board [1] while the portion of embedded hardware and software has been continuously increasing in the last few decades [2]. In addition to that, complex communication schemes are commonplace [3]. The evolution of integrated electronic and software in automotive systems since the introduction of the first ECUs in the seventies is presented in [4] and shown in Figure 1.

Due to this complexity, early investigation of the system can improve the detection of design errors before implementing and integrating into the real hardware. It can also facilitate different design activities such as distributed function definition and integration, or hardware-software Co-Design of Electronic Control Units (ECU) [5].

However, when software is developed for embedded systems, no hardware prototypes are available in the early stages of development. Therefore, the control devices are built as

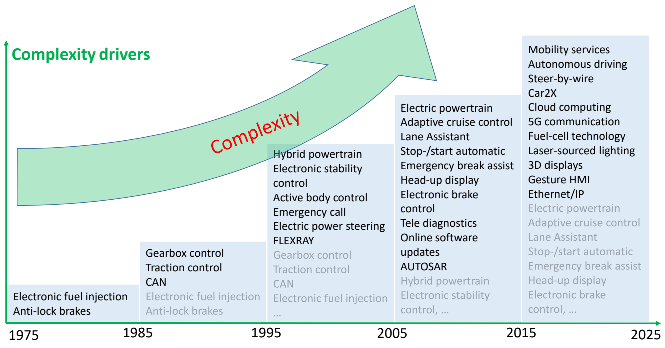

Fig. 1. Evolution of IT complexity drivers in vehicles [4]

mathematical models and the behavior of the software is simulated [6]. This means that development and testing of the software can begin at an early stage, which helps discovering problems at the design phase of the development process.

One way to test applications at an early stage is to simulate them on the computer. For this purpose, graphical models of the algorithms, e. g. in the form of state charts, are generated or described on a text basis. These models can be simulated on a PC as Model-in-the-Loop (MiL) or Software-in-the-Loop (SiL). The aim of the simulation on the computer is to investigate the robustness and applicability of the algorithms used in the system [6]. Rapid control prototyping (RCP) is a process to quickly test and iterate control strategies. It consists in importing models of the software on a powerful real-time machine with I/O interfaces to connect to real-world systems. In contrast to RCP, virtual prototyping focuses on the system simulation on a computer, including the simulation of the hardware (virtual platform). For the virtual platform simulation, the models of the hardware and the application run on a host computer. However, unlike simulation on the computer, the same compiled code is executed on the virtual platform and the real world control unit [7]. Through integration of these virtual platforms into heterogeneous multi-domain simulation systems, powerful co-simulation environments can be developed building some kind of virtual Hardware-in-the-Loop (HiL) [8].

In this work, we show the use of a testing method based on virtual platforms to verify a new software for a high number of system variants. This is seen as an important step towards validating a new software release, which fixes for example some serious bugs, for all the variants of the system. As it is very costly and time-consuming to configure and test each variant on real hardware, virtual platforms offer support and can reduce the validation effort significantly for closed-loop systems in an economical way [9].

This paper is structured as follows: after explaining the terms variant and version and presenting the state of the art of current test methods and virtual platform technology in section II, we analyze some of the challenges in the validation process of complex software/hardware systems in section III. Thereafter, the virtual test method is considered in section IV. The used proof of concept system for the suggested test method for variant-rich systems including its virtualization is introduced in section V. The scalability analysis of the used virtual platform simulator is topic of section VI. At the end, the results are evaluated considering different criteria in section VII before finishing with a conclusion and future work in section VIII.

II. State of the Art

A. Variants and Versions

System variants in software engineering, as defined in [10], describe intentional variations that exist in parallel; this is also called variation in space, while a revision is an ordered variation over time. Versions encompass both revisions and variants.

The growing software complexity and customization possibilities of automotive systems has led to the introduction of software product lines, where multiple products of a manufacturer share substantial similarities and are developed and maintained together [10]. A product line could be a family of car models including many configurations and equipment combinations. In order to deal with the immense variant space, different methods to model the variability like the feature-models, are used. Feature models allow the developer to capture commonality and variability within a software product line and define the relations and constraints between them [11]. In addition to managing variants, an efficient version control system is essential. Especially in the case of frequent software updates. The software changes of a module or system are usually documented and tracked inside a codeline diagram, where the temporal evolution of the system is documented, and parallel development for different purposes is possible through branching of the mainline [1].

In automotive product lines, variants are usually defined on the vehicle level. In this paper, we concentrate on ECU variants: i. e. the existing configurations for an ECU network realizing a specific distributed function, such as the case for advanced driver-assistance systems.

B. Current Test Methods

Testing requires a systematic approach with preconditions, stimuli and expected outcomes [12], which are defined in test cases. The definition of these test cases is very important for a successful test phase. Since an exhaustive testing covering all possible scenarios and use cases of a system is not possible, a test strategy is required to reduce this incompleteness as much as possible.

Current Test methods differentiate between three strategies: Blackbox, Greybox and Whitebox. In the so-called *Blackbox-Test*, the test engineering has only knowledge about the interfaces. The internal behavior of the program or the module is not considered. The test data is generated from the specifications of the system. Having information about the interfaces and the internal behavior leads to the so-called *Greybox-Test*. When using the internal program logic to define the test data, the strategy is called *Whitebox-Test*.

Due to the decoupling of the applications from the hardware within ECU software as defined by the AUTOSAR standard [13] making the same application usable for different hardware platforms, the development is simplified and the individual logical behavior of one device is defined by the software modules of the application layer. For this reason, function and integration tests for application modules are usually carried out long before the software is integrated in the corresponding hardware platforms [12]. Depending on the maturity level of the system and the form of its external logic, different kinds of test methods can be employed. These are described in Figure 2. The external logic in Figure 2 includes the environment of the vehicle, additional ECUs, sensors, and actuators.

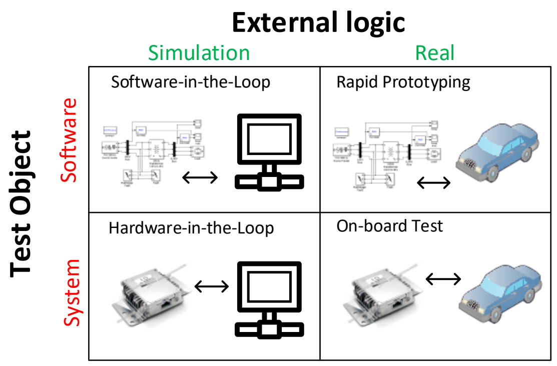

Fig. 2. Different test technologies used in the automotive field [12]

There are four different ways to test an embedded system: Software-in-the-Loop, Hardware-in-the-Loop, Rapid Prototyping and on-board Test.

Using the software together with a simulated external logic in an automated test loop is called Software-in-the-Loop (SiL). This technique has been subject to different research works, such as studied by [14]. It allows for verifying the behavior of automotive software early during the development process, saving costs and improving quality.

Subsequently to the SiL, the resulting ECU is tested within a simulated environment. This technique is called Hardware-in-the-Loop (HiL) and is an essential method on the way to validate the system, allowing to discover inconsistencies and different kinds of system-related errors [1].

When algorithms are used for the first time in prototypes with the real external logic, the technique is called Rapid

Prototyping. The algorithms are implemented and tested on powerful hardware inside a real world prototype [6]. The programmed high-performance hardware is either tested in HiL test-benches or directly in the real environment.

For the on-board test, the system is build into a vehicle and tested on dedicated test sites or public roads. Therefor the test object and the external logic including the environment of the vehicle are real.

A newer approach consists of emulating the target hardware platform and executing the cross-compiled software modules directly on it. Using this concept as a virtual Hardware-in-the-Loop (vHiL) can be found in [8]. This enables to emulate the behavior of the embedded system with more precision than traditional SiL simulation considering also hardware and communication artifacts. The placement of such tests in Figure 2 is between SiL and HiL.

C. *Virtual Platforms*

For testing the system with a vHiL, a virtual platform of the ECU or the hardware is needed. A virtual platform is a model that contains all components of the hardware. This includes the processor, memory, peripherals and the connecting bus structures (local busses).

For the simulation of embedded systems in a vHiL, diverse virtualization tools exist. The different platforms will be compared below. An overview of typical use cases for these can be found in [15].

GXemul [16] is a paravirtualization that supports ARM, MIPS and PowerPC. Paravirtualization is a virtualization technique that uses a software interface, which is similar but not identical to the hardware interface. Therefore, the software for the paravirtualization needs to be adapted compared to the one running on the embedded system.

Simics [17] supports a high number of processors and operating systems. The simulation environment gives the possibility to run the same code on the virtual platform as on the real platform. However, the interfaces to control the system architecture can't be build by the programmer. Therefore, no customized view inside the architecture is possible.

QEMU [18] is a fast simulation of the overall hardware platform (including interfaces). It supports several automotive ECUs and different operating systems and has the possibility to build own control software in the platform. However, the supported list of operating systems does not include automotive operating systems complaint with as AUTOSAR.

Synopsys Virtual Platform [19] is a virtual prototyping platform that supports different processor architectures and operating systems. The tool simulates the overall system with peripherals and memory. The simulation can be adapted with different files.

OVPSim [20] is a fast processor and platform modeling tool. It supports a high number of processors and operating systems (including custom build processors and operating systems). Further, it allows to build a platform with all components, memory and peripherals. Adapting the simulation files, the simulation gives a deep view in the processor.

The best adaption of the simulation is given in OVPSim, therefore this paper will focus on this tool for its results. Nevertheless other simulation tools (e.g. Synopsys Virtual Platform or QEMU) can be used as well.

The used simulator is instruction accurate. Although it does not simulate the exact time behavior, the execution time can be recalculated by the Instruction Set Architecture (ISA), taking this into account it is even possible to meet real time requirements.

III. Research Challenges for Testing Variant-Rich Systems

ECUs in automotive systems do not only vary in time with different versions of the software functions. They also vary in space (see section II-A) through the various hardware configurations realizing the distributed ECU function. The amount of variants for a simplified ACC function resulting from the combinations of two different micro-controllers, sensors and actuators is depicted in Figure 3. These variants need to be separately verified for compatibility for each new software update. Using more hardware components, adding communication, or changing the ECU will even increase the number of variants for a modern vehicle. And adding to these considerations the variation in time through different in parallel existing software applications, the variants problem becomes even more serious.

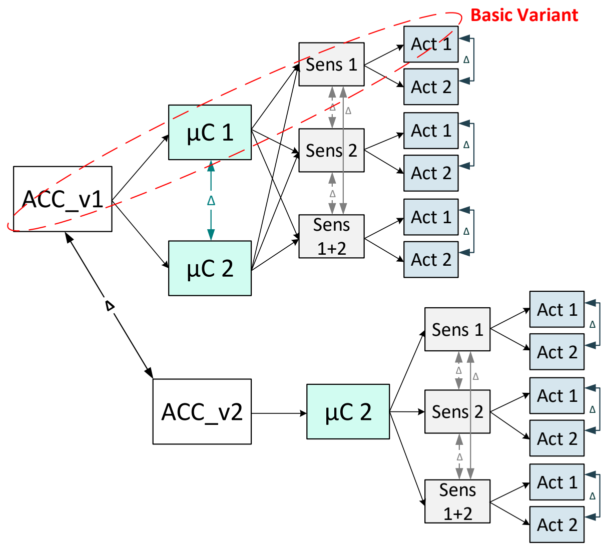

Fig. 3. Considered Variants of the System Under Test

In addition: If one takes the fast changing software for cybersecurity updates into account, that is updated multiple times per month on mobile devices, the number of variants with different software is increasing even more. Such a high software update rate can only be achieved by adding SOTA. However adding SOTA will increase the testing effort, since more software variants have to be tested in a shorter time frame.

Usually variant-rich systems are verified through testing all relevant variants by running automated test cases for each variant; these test cases have to support variability on the different abstraction levels [21]. This can be achieved by testing the basic variant with HiL and focusing other tests on the deltas to decrease the test effort. A model-based testing framework based on a delta-oriented Software Product Line (SPL) test model and regression-based test artifacts is, for example, introduced in [22]. Nevertheless, the delta tests have to be executed on a physical hardware inside the HiL. I. e. the physical hardware for all variants needs to be present for testing [21]. Putting this effort of delta testing into simulation using a vHiL approach (as discussed in section IV), the physical hardware can be replaced by virtual platforms.

IV. Virtual Testing Method

The main use-case of virtual platforms is design verification allowing to discover design and implementation errors early within classical development processes such as the V-model. We consider the use of this method for the case of continuous development, such as for regular SOTA updates, as powerful support to deal with the enormous variants numbers in the automotive field. Within the virtual system, an ECU including its internal architecture and communication schemes can be modeled. Adding an adequate co-simulation including the environment of the network under test, all variants of the system can be reproduced virtually. These variants can subsequently be tested in a complete virtual environment based on the delta, i. e. differences, to the basic system configuration, which should have been thoroughly verified and validated before. The considered system and its variants represent the distributed ECU network realizing a common vehicle function, and not the whole vehicle.

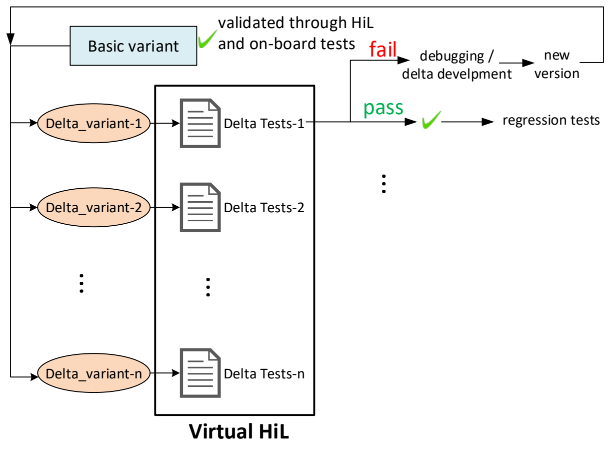

Fig. 4. Virtual delta tests methodology

If the delta-tests are passed successfully, final regression tests should be conducted to make sure the changes did not lead to unexpected errors within the system. As to the question whether only virtual tests are enough for the validation of the variants, the answer is mainly depending on the safety criticality of the considered system or sub-system.

Instruction set simulators like OVPSim [20] can be used to model a processor with the corresponding peripherals and run the cross-compiled application. Running the cross-compiled application inside an instruction set simulator gives the same behavior as on the target platform.

Intercepting the application as defined in [7] can cause the simulation to stop the application and run test and verification code in the simulation environment. This includes among others the interception of the virtual platform before each instruction is morphed, specific instructions are executed and a specific address range is read or written. This method helps to check the output of the stimuli against the specified and expected output and detects undefined behavior due to change in variants.

In this way, clearly defined delta-tests from variant management methods using the deltas to the basic variant can be defined in test-benches and run for each-variant. If the tests are successful, the same software update can be released for the considered variant. Otherwise, delta development based on the debugging of the considered test cases must be conducted.

V. Automotive Test System

To overcome the issues having physical hardware for the delta tests, the behavior of the hardware is modeled with virtual platforms and simulated with OVPSim. As a system under test (SUT), we consider an adaptive cruise control (ACC), which communicates with a motor controller (MC), as described in Figure 5. The connected ECUs represent the distributed character of an automotive embedded systems. The considered use-case is a simple longitudinal control maintaining a safe distance d_s to a leading vehicle. Besides the two ECUs (ACC and MC), the system includes a distance sensor to measure the range to a leading car, a speed sensor, a motor to control the system dynamics and an LED display to indicate if the car is accelerating or decelerating. The communication between the two ECUs is realized through a CAN bus.

In order to apply the testing method for variant-rich systems, we defined features for the SUT which result in different variants. These features are the number of ECUs (one or two), the used sensors (Sens 1, Sens 2, Sens 1+2) and the required actuator (Act 1 or Act 2), as depicted in Figure 3. The software variants change according to the used hardware configuration and deltas are generated by the features. For simplification, we do not consider different software versions as feature, however this could be done as well. The SUT is defined as the basic variant, and all other variants are described by the deltas.

The SUT is build in two different ways:

1) Using physical hardware
2) Simulation with virtual platform

A. Physical Hardware Device

The SUT was build using two Tiva C Series TM4C123G micro-controllers [23] using a ARM Cortex-M4F according to the schematic shown in Figure 5. The micro-controllers are

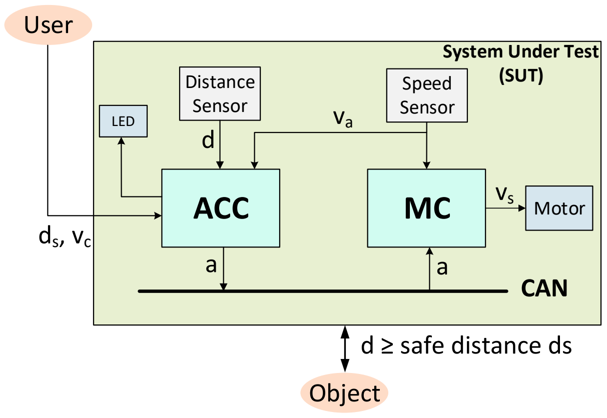

Fig. 5. Concept of the system under test

connected via CAN each other, distance sensors, and motors. The test data are fed into the distance sensor to check the function of the ACC.

B. Virtual Test Platform

In addition to the physical device the system was build as a virtual platform with two processors, memory and a common bus as shown in Figure 6. Using a virtual platform requires a sufficiently accurate model of the hardware. For the tests, models provided by Imperas were used. The CAN bus has been simplified to a common bus with access to a shared memory. The sensor models have been replaced by the object list of the lead car. This object lists are used as a testbench data and no co-simulation is used.

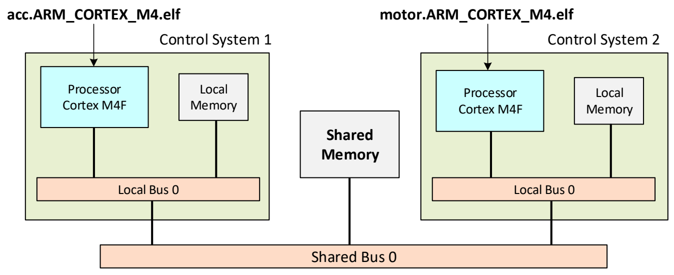

Fig. 6. Simplified virtual platform for the simulation of the ACC system

The platform above can be extended with peripheral models, bridges to connect the different busses, co-simulation for models of sensors, actuators, bus communication and environment. Also the simulation can contain more ECUs leading to the virtual test system depicted in Figure 7.

C. Co-Simulation for External Logic

Co-simulation can combine the advantages of different simulators to replicate the behavior of the system in a more accurate way. The required co-simulation can also include precise models of the internal bus communication using e. g. CANoe and real-world test scenarios including the entire surrounding environment using e. g. CarMaker. With a database including all variants of the models involved in the co-simulation, each system variant can be virtually reproduced and tested without the need of configuring a multitude of instances in real hardware.

In the case of using virtual platforms, co-simulation with other tools like MatLab Simulink or even real hardware as demonstrated in [24] can be implemented. Poppen et al. [25] presented a proof-of-concept implementation of a co-simulation interface between a c-based system and MatLab Simulink. OVPsim uses an interception mechanism to stop the simulation and run a Simulink model representing the environment of the embedded system.

VI. Scalability of Virtual Testing

To use the virtual platforms for the simulation of multiple control units the scalability is an important factor. Therefore we run a simple application for 750 seconds on the platforms shown in Figure 7, where the control systems communicate with each other. For the scalability tests only testbenches without co-simulation were used.

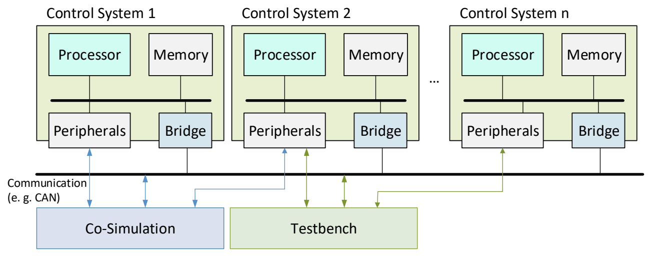

Fig. 7. System for the Virtual Testing and Scalability Measurement

Compiling the application for different kind of processors will lead to different binary outputs, since the instruction set depends on the processor architecture (see Figure 8). Using a processor with reduced instruction set (OR1K) will lead to the highest number of instructions and therefore to longer execution times for the same application. Taking the aforementioned ARM-Cortex M4F with an complex instruction set a small number of instructions is achieved.

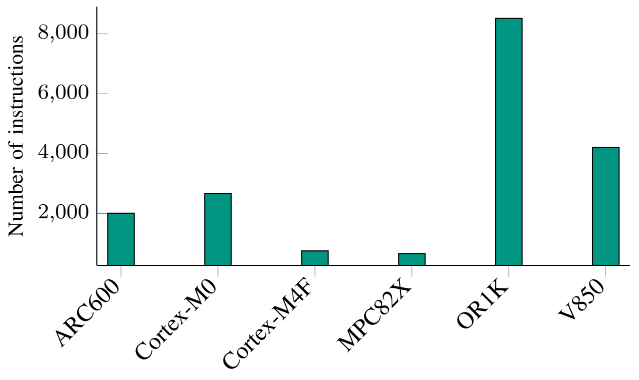

Fig. 8. Comparison of the instruction count of different processor types

For the measurement of the scalability the ARM Cortex-M4F is used, because the hardware tests are done with the same model. Therefore the following section only revers to this processor type.

Using a Linux PC (Quad-Core with 1.2 Ghz, 8 GB RAM) the results of Figure 9 can be measured. The graph shows the real time ratio (see logarithmic axis on right in Figure 1), taking $user\_time$ (left axis in Figure 9) as the time needed for the simulation at the computer and $simulated\_time$ as the real time on the hardware (which is 750 seconds for ARM Crotex-M4F).

$$real\_time\_ratio = \frac{simulated\_time}{user\_time} \tag{1}$$

The real time ratio stays above one using up to 60 processors simulating with the Linux PC mentioned above. I. e. simulating less processors leads to a speed up in tests compared to physical devices. Simulating only one controller gives a speed-up of 50 times faster than real time. Increasing the computation power will proportionally increase the real time ratio. Increasing the computation power gives the possibility to simulate whole electric-electronic (E/E) architectures of a vehicle and co-simulate the corresponding external logic.

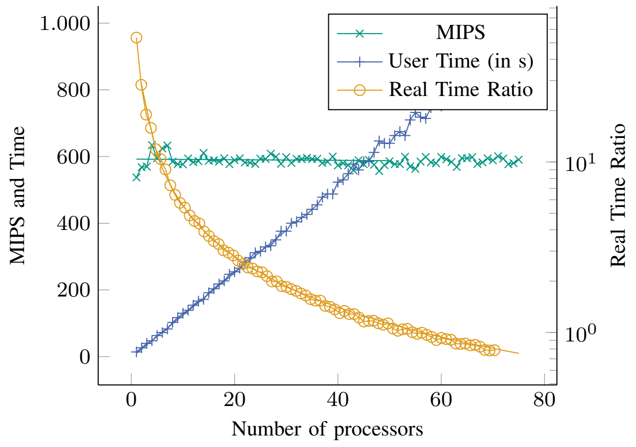

Fig. 9. Real time ration over number of processors

Also shown in Figure 9 are the million instructions per second (MIPS) which stays almost constant independent of the processor count. This means the simulation overhead is not increasing with the numbers of processors.

VII. Evaluation

For ACC testing, only the relevant ECUs were simulated (see Figure 5 and Figure 6).

The testing is based on the cross-compiled code for the target platform. I. e. the instructions order and the behavior is the same as on the real platform, since no optimization of the compiler will be done. Further, the executable and linkable format (elf) file used for the testing can be flashed to the target device without any additional changes.

Running the simulation for a testbench with a simple stimuli of three intervals of constant velocities of a lead car delivers the results represented in Figure 10 for the physical hardware and the virtual platform.

As Figure 10 shows, the adaption of the distance to the lead car at the simulation time $t = 39\,\text{s}$ was successful by

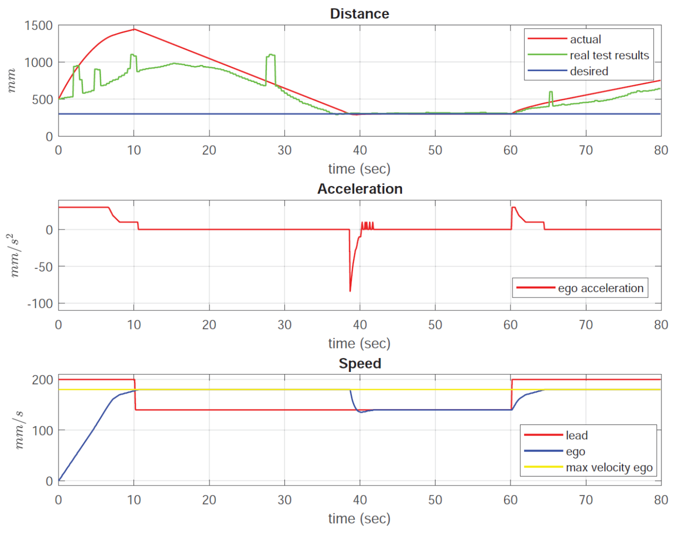

Fig. 10. Results of the virtual platform based test of the ACC system

respecting the given safe distance $d_s = 300\,\text{mm}$. Although some important physical effects like the influence of temperature or weather conditions are not included in the used simulation environment, the behavior of the virtual ACC system is very similar to the one of the real hardware system, as shown by the green distance curve representing real world results. The real test has been conducted by configuring the same velocity and distance profile on a small lead car controlled by a micro-controller and logging the distance information each $40\,\text{ms}$. The higher jumps in the green curve represent measurement errors of the used sensor. Also the real world results were not exactly reproducible.

On the other hand, the virtual system is less abstract than a pure Simulink simulation, which does not consider hardware aspects. However, Simulink or other tools can add more depth to the simulation by adding realistic models for the sensors, actuators and dynamic environment.

After running the test for the basic variant of the system, which is described in Figure 5, we run different other tests for the remaining variants. These tests are supposed to be targeting the deltas to the basic variant. We considered the case of a different sensor, which can give additional information to the system. This is for example the case for a sensor measuring not only the distance d to, but also the velocity v_{lead} of the lead car. This change is representing one of the variants of Figure 3 of the SUT. It has been configured and tested on an equivalent virtual platform by changing the software of the ACC ECU and the sensor model in the test-bench.

In this work, we didn't explicitly define the delta-tests to the basic variant, but showed the feasibility of the novel testing method for variant-rich systems. The scalability evaluation using OVP models showed the potential of this approach to virtually test complex ECU networks.

VIII. Conclusion and Future Work

Increasing complexity and variant-rich systems (such as SOTA) lead to a huge number of different variants that have to be tested. Using state of the art methods, the different variants need to be tested with the corresponding hardware on a HiL. However, taking the increasing number of variants and shortening life cycles, hardware tests are becoming very costly and time-consuming.

In this work, a novel method was introduced to reduce the hardware test effort by using virtual platforms for delta testing. It uses virtual platforms for automated delta testing to handle the abundance of system configurations. Virtual testing was used to reduce the amount of physical hardware tests and allow efficient variants verification. As a proof of concept, an adaptive cruise control (ACC) composed of two ECUs has been implemented both in real hardware and with a virtual platform. Considering the increasing amount of ECUs and processors in a vehicle we investigated the scalability and showed that solutions with less than 60 processors can be simulated by a desktop computer faster than real time. The results of delta testing correlates between virtual platform and physical hardware. This is achieved by cross-compiling the software to run on both virtual and physical hardware.

Future work will focus on the implementation of co-simulations and the automatic generation of test cases for delta testing. Besides, the evaluation process will be carried out with a distributed system including more than two ECUs operating in various use cases. Further research topics are the investigation of automatable parts of the Delta product line development (from the requirement to the test) and whether the release of functions by virtual tests is sufficient and can replace HiL tests.

References

[1] E. Sax, R. Reussner, H. Guissouma, H. Klare, and H. Guissouma, "A survey on the state and future of automotive software release and configuration management," *Karlsruhe Reports in Informatics*, 2017.

[2] J. Schäuffele and T. Zurawka, *Automotive Software Engineering*. ATZ/MTZ-Fachbuch, 2016.

[3] J. Quigley and K. Robertson, *Configuration Management: Theory, Practice, and Application*. CRC Press, 2015.

[4] M. Staron, "Automotive software architectures : An introduction," 2017.

[5] P. Giusto, A. Ferrari, L. Lavagno, J. Y. Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli, "Automotive virtual integration platforms: why's, what's, and how's," in *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 370–378, 2002.

[6] D. Abel and A. Bollig, *Rapid Control Prototyping*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2006.

[7] A. Lauber and E. Sax, "Testing Security of Embedded Software through Virtual Processor Instrumentation," in *Online engineering & internet of things // Online engineering & Internet of Things* (M. E. Auer and D. G. Zutin, eds.), vol. 22 of *Lecture notes in networks and systems*, pp. 85–94, Cham, Switzerland: Springer and Springer International Publishing, 2018.

[8] R. L. Bucs, L. G. Murillo, E. Korotcenko, G. Dugge, R. Leupers, G. Ascheid, A. Ropers, M. Wedler, and A. Hoffmann, "Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface," in *2015 Forum on Specification and Design Languages (FDL)*, pp. 1–8, Sept 2015.

[9] D. Andreas Junghanns, R. Serway, D. Thomas Liebezeit, and M. Bonin, "Building virtual ecus quickly and economically," vol. 7, 06 2012.

[10] S. Apel, D. Batory, C. Kstner, and G. Saake, *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013.

[11] J. Zhou, Y. Lu, K. Lundqvist, H. Lnn, D. Karlsson, and B. Liwng, "Towards feature-oriented requirements validation for automotive systems," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pp. 428–436, Aug 2014.

[12] E. Sax, ed., *Automatisiertes Testen eingebetteter Systeme in der Automobilindustrie*. München: Hanser, 2008.

[13] "Automotive open system architecture (autosar)," 2018. https://www.autosar.org/, last accessed: 23.02.2018.

[14] S. Jeong, Y. Kwak, and W. J. Lee, "Software-in-the-loop simulation for early-stage testing of autosar software component," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 59–63, July 2016.

[15] S. Werner, A. Lauber, J. Becker, and E. Sax, "Cloud-based Remote Virtual Prototyping Platform for Embedded Control Applications: cloud-based infrastructure for large-scale embedded hardware-related programming laboratories," in *Proceedings of 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, IEEE, 2016.

[16] "GXemul," 2018. http://gxemul.sourceforge.net/, last accessed: 12.02.2018.

[17] Wind River Systems Inc., "Wind-River-Simics: Product-Overview," 2015.

[18] "QEMU: the FAST! processor emulator," 2018. https://www.qemu.org/, last accessed: 12.02.2018.

[19] Synopsys Inc., "Virtual Prototyping," 2018. https://www.synopsys.com/verification/virtual-prototyping.html, last accessed: 12.02.2018.

[20] Imperas Software Limited, "Open Virtual Platforms: The source of Fast Processor Models & Platforms Open Virtual Platforms - the source of Fast Processor Models & Platforms," 2016.

[21] A. Leitner, R. Mader, C. Kreiner, C. Steger, and R. Weiß, "A development methodology for variant-rich automotive software architectures," *e & i Elektrotechnik und Informationstechnik*, vol. 128, pp. 222–227, Jun 2011.

[22] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *Tests and Proofs* (A. D. Brucker and J. Julliand, eds.), (Berlin, Heidelberg), pp. 67–82, Springer Berlin Heidelberg, 2012.

[23] T. Instruments, "Arm cortex-m4f based mcu tm4c123g launchpad evaluation kit," 2016.

[24] S. Werner, L. Masing, F. Lesniak, and J. Becker, "Software-in-the-loop simulation of embedded control applications based on virtual platforms," 2015.

[25] F. Poppen and K. Grüttner, "Co-simulation of c-based soc simulators and matlab simulink," 01 2012.