Reliable Validation of Highly Automated Driving Functions by Increasing the Virtualization Level of High Performance Computing Platforms and Smart Sensors

Dipl.-Inf. Stefan Wonneberger, M.Sc. Sinan Balci, CARIAD SE, Wolfsburg

Abstract
The increasing automation level of driving and parking functions in upcoming vehicle generations also leads to a centralization of functions on high performance computing platforms. Introduction of IP-based networks renders the vehicle bus networks more and more into a classic data center architecture bridging into a cloud backend and reducing the communication traffic over traditional automotive bus systems.

However, the high safety level of automated driving functions require both dimensions of a vehicle computing network – reliable deterministic communication achieved in combination of different bus systems together with microcontroller levels up to ASIL-D and a system of high performance processors with a high-bandwidth interconnection.

For validation of those systems, the amount of scenarios and test-cases requires the usage of data center infrastructure and reduction of real hardware, HiL-based testing to only those test-cases targeting hardware level and SiL-validation purposes.

In our work we analyze the architecture of high-performance control units, e.g. hosting the primary driving functions, and the sensor and actor components in combination with a view on the safety and performance subsystem parts. We demonstrate an innovative approach to transfer both subsystems at once into a virtualized, software-in-the-loop environment allowing a higher testing coverage and offloading HiL-based testing into an easier data center scalable SiL framework.

Beside the feasibility demonstration we identify in our work methodical challenges in the goal of a higher virtualization level over all subparts of an automated driving system and encourage for standardization efforts in this field.
Introduction and Motivation

Today's advanced driver assistance systems and automated driving functions (ADAS/AD) are one of the most complex and widely integrated vehicle functions in a vehicle's software and electronics architecture. The rising ADAS SAE Levels up to 3 and 4 of automated driving functions in consumer vehicles lead to a tremendous increase in test requirements for function validation. Covering the software complexity can only be achieved by an early and continuous integration and testing process over all development phases.

With increasing software complexity, processing units (electronic control units, ECUs) get more and more powerful and changing to complete High Performance Computing Platforms (HCPs). Microcontrollers are incorporated for software parts of highest automotive safety levels while powerful system-on-chip groups (SoCs) are integrated for specific sensor-, deep learning based data or high-definition graphics processing. Multi-processors on those platforms already exceeded the performance of other consumer electronic devices, such as smartphones [1]. Validation and development testing especially for end-2-end tests thus covers software parts across a complete HCP and due to the already motivated distributed software for a customer function, also multiple HCPs and peripheral ECUs over the full vehicle E/E architecture.

The high amount of test cases and a demand for testing already at early development stages render even the most efficient and scalable HiL (hardware-in-the-loop) test benches in cost intensive parts over all development areas. Especially in early development phases with only initial hardware samples a bottleneck arises quickly. Long HiL-update-cycles also conflict with a continuous integration and testing approach on build level basis for the full vehicle software stack. Moving the complete HCP environment and required peripheral software into a virtual integration and test environment with a maximum of possible test coverage is a logical consequence and motivated by the cost reduction and faster test cycles. Also the fast scalability of the test environment across generally available data centers or cloud environments depending on a daily based test amount with easier to maintain processor hardware is a further motivation. For a good test and function performance coverage a high amount of virtualization is desired to reduce the amount of emulation and model-based abstraction of the full software stack to be tested. Oel et.al. already motivated our activities in this field in [2] to seamlessly connect virtual integration with connected real hardware testing in our test & integration facilities.

In the following we give a closer look to HCP-based vehicle E/E architectures in general before we investigate in this work the current virtualization possibilities of a ADAS-specific HCP. We assumed here a generic HCP architecture derived from literature for a general state-of-the-art
Discussion of E/E Architectures for High Performance Computing Platforms

The trend to centralized, in-vehicle High Performance Computing Platforms (HCPs) has been widely motivated in the past. In [3], a good overview of E/E architecture generations can be found. All software is bundled on centralized High Performance Computing Platforms, typically with a specific domain usage such as body control, vehicle motion, infotainment and connectivity or - as focus of this work - as central host for the ADAS and automatic driving functions including the necessary sensor processing. With respect to entry-segment passenger vehicles, the ADAS systems may split to so called Smart Sensors, which can be themselves considered as small HCPs, for example a camera with combined safety and comfort functions or a control host for parking and surround view functions. The Volkswagen Group already implements its E/E architecture in this way within its ICAS (In Car Application Server) systems [4] and the distributed software framework VW.OS [5] for vehicle-wide connected software services. For ADAS/AD, the Volkswagen Group early introduced with the zFAS [5] a domain specific HCP in its premium vehicles. Figure 1 shows a generic brief example of a High Performance Computing Platform (HCP) based E/E architecture with domain specific processing nodes.

Even if the software is partitioned by a domain-centric concepts, todays customer functions already include software parts and vehicle data distributed over all domains. This applies not solely but with a high impact to the ADAS/AD functions. Within the SAE Levels 3-4, where...
partially automated driving is already part of the function, a high amount of test coverage for function performance and safety validation is required. Driver interaction and information is here still part of the function, which includes the infotainment and connectivity domain into the test system. For example for take over interaction and correct system state observation possibilities to the driver. The vehicle motion is typically the low level motion control endpoint for all driver assistance systems and is the second critical domain which is required in most test setups. Finally, all vehicle base functions are located in the body control domain and required for base software operations of all domains. That shows that, not only but especially, for the ADAS/AD domain typically a complete set of HCPs is required for efficient function testing and validation. In addition, with increasing complexity of the software modules on each HCP, setting up simulation environments for single HCP-isolated domain tests get more and more complex even with an optimal software partitioning and interface alignment between the domains. Another reason for non-strict software partitioning is variant scaling over vehicle classes where basic functionality of all domains can be integrated in a central control unit, combined with Smart Sensors such as Radar and Camera with Level 2 functionality for the ADAS domain as mentioned before.

As motivated in the beginning and with this discussion in mind, especially for efficient end-to-end testing and early testing during development, a scalable-, easy to maintenance test environment including the complete HCP-layer of the E/E architecture is required. A Software-in-the-loop (SiL) environment on data center or developer hardware level is here an efficient extension to a limited HiL environment in terms of scaling, maintenance costs and also partial deployment to developer workplaces. The key challenge here arises in the virtualization of the individual HCPs themselves since they are already internally complex multi-processor environments with typically multiple chips, a local network interconnection and domain specific co-processors for sensor or graphics processing. Multiple processor isles are typically partitioned to different software safety levels. Figure 1 also shows a typical sub-structuring within the HCPs.

While most tools designed for Virtual ECU emulation are very good suited for classic software environments in the micro-controller area, the micro-processor area already tends more and more to typical consumer hardware where base software stack emulation is not needed. In the following we analyze a reference HCP for virtualization possibilities and current state-of-the-art for the individual HCP parts. We selected the ADAS/AD domain for our analysis as it is the key target of our test focus but also provides a comparable basis for other domain units like

https://doi.org/10.51202/9783181023846-17
Generiert durch IP '54.190.42.255', am 06.11.2021, 07:05:44.
Das Erstellen und Weitergeben von Kopien dieses PDFs ist nicht zulässig.
the infotainment domain. Our evaluation is based on a theoretical HCP for the ADAS/AD domain based on reference designs from literature and state-of-the-art in the market.

**Evaluation of HCP Virtualization**

For our virtualization analysis we considered a typical architecture of a modern ADAS/AD focused domain processing unit (Domain ECU). For safety critical software components a microcontroller with an up to ASIL-D capable operating system and software stack like an AUTOSAR Classic conform RTE is integrated. Software parts with higher performance demands run on an additional microprocessor system. For service-oriented communication with adjacent vehicle domains the operating system and software framework we considered to be Adaptive AUTOSAR compatible. Especially for the important high performance sensor pre-processing in the ADAS/AD domain a coprocessor for highly parallel computing is integrated. In our evaluation target we assumed a CUDA-compatible general-purpose GPU. Since the ADAS/AD domain is rather stream oriented and time triggered this subsystem in our evaluation runs the NVIDIA base system Drive OS [7] combined with a proprietary AUTOSAR compatible runtime framework derived from the zFAS project [6].

For variant scaling flexibility we assumed the sensor-processing subsystem and function processing subsystem each in different processors on the HCP system connected with a high-bandwidth interface. For resource partitioning and safety-isolation the microprocessor subsystems can host multiple container partitions using standard kernel virtualization technology. Figure 2 shows the resulting conceptual architecture of the reference ADAS/AD HCP used for our evaluation as sketched above.

![Fig. 2: Architecture Concept of a reference ADAS/AD HCP used as basis for the Virtualization Evaluation](image_url)
As motivated in the beginning and in E/E architecture discussion the virtualization shall assist as a powerful test environment for end-2-end function testing over the full vehicle electronics architecture with special respect to the ADAS/AD functions. That means we postulate the following basic requirements to our environment:

- **Commonly Available Hardware / HCP Board Independence**
  In order to perform early testing and allow an easy scaling of test execution we concentrate on typically available data-center or cloud available processor hardware to reduce the HiL farm size for testing to only those tests requiring the exact HCP Board environment.

- **Maximum Software Stack Inclusion**
  To find a maximum amount on function errors with respect to HiL or in-vehicle tests we demand a high inclusion rate of the target software lines-of-code. This especially includes runtime frameworks and operating system with only a minimal drop-out of the hardware / micro controller abstraction layer (MCAL).

- **Environment Scaling**
  To allow an easy setup and fast scaling of the environment for parallel test execution the distribution shall be easily and based on state-of-the-art virtualization and deployment technologies.

- **High Real-Time Factor**
  For fast test responses we further demand an simulation factor close to one (real-time) and thus exclude processor or partial instruction-set emulation. This automatically cuts the possible test space from those tests requiring the HiL and HCP target environment.

Beside those requirements for the function validation of new software releases we consider the environment to be used for development and integration tests. This means that the virtualization shall also scale down to a normal developer workplace and allow a closed debugging of software spawned across the multi-processor multi-partition environment.

**Generic Virtualization Concept**

The requirements above also define our targeted virtualization level. Here we follow the definition given by the recently distributed white paper on Virtual ECUs [8]. For all subsystems and partitions we are targeting the V-ECU Level 3, which means that application software and basic software (runtime framework, operating system) excluding the MCAL (micro-controller abstraction layer) is the exact production code. Depending on the available execution
processor and coprocessor types also binary code can be considered at least for the application software. Since this is not exactly defined as an V-ECU Level we simple targeting our V-ECU Level as 3+.

For our concept we started with some general decisions before we individually analyzed virtualization challenges and possibilities of the individual subsystems. Special aspect is given to the network system on the HCP board between the processor subsystems and the vehicle network communication layer for HCP-to-HCP communication.

Since the performance and sensor processing subsystem are natively based on UNIX-like operating systems we choose this as starting point for our virtualization. For the beginning we started with a basic Linux distribution. We flattened the subsystems and concentrated on the already given partitioning possibility. That lead us to the decision to use the native container technology for direct mapping of the target partitions to Linux containers. In our evaluation environment we used Docker for the container partitioning. If the target operating systems is available for normal processor systems it can be directly used as basis, QNX [9] can be given here as an example. As most datacenters already provide compatible ARM-based processors this is an alternative if the operating system is only available for this instruction set.

![Fig. 3: Virtual HCP Environment](image)

For the communication layer we differentiated between classic automotive network connections, IP-based internal and external communication and inter-processor communication. For each IP-network we add a Linux virtual network between the connected partitions. For inter-processor communication or external sensor data input we add an additional middleware layer between all partitions based on our in-house Functional Engineering Platform (FEP) [10] utilizing the DDS protocol for generic frame based communication. This platform also acts as our external co-simulation middleware which
provides the central simulation synchronization to all underlying time-based software stacks [11]. With this setup we were already able to abstract all inter-subsystem communication. An eye on external vehicle network communication is given in a later section. Figure 3 shows our resulting virtual HCP environment as described above. In the following we will address individual aspects of the different subsystems and the remaining communication layer parts.

Performance Subsystem
We used the reference Adaptive AUTOSAR stack and an available implementation of our internal vehicle operating system. Since this stack was available cross-compiled for standard x86 systems it easily fits in our virtual environment. We ported the stack to our Docker configuration and adjusted the network setup accordingly. Since this partition communicates only to the IP-based internal and external network, this subsystem does not need any emulation software at all and, assuming an available ARM-subsystem, could run nearly close to target system binary code considering a data center for validation testing.

Sensor Processing Subsystem
Our test environment basically used CUDA and stream oriented data transfers in the target architecture. We used a data center hardware environment which provides the sufficient CUDA standard, the necessary components of the target Drive OS [7] where available in the standard SDKs and could easily be installed in our container environment as well. Our proprietary ADAS stack was already available for generic processor Linux Environments which connects the NVIDIA stack to the application software and the internal IP-network for transmitting result data to other partitions in AUTOSAR-compatible formats. While sensor raw-data processing like image processing is directly performed in the NVIDIA subsystem, the ADAS stack schedules the time triggered parts such as data fusion and path planning. Since our environment stubs external sensor input we connected our platform middleware [9] to the partition and rewrote the sensor input that it streams images from our middleware into the system.

Safety Subsystem
The safety subsystem executes all safety-critical software parts and routes safety-critical communication to reliable or redundant vehicle networks such as CAN or FlexRay. Since the real software stack was not available for x86 systems we were required to use a generic emulation of Classic AUTOSAR RTE to integrate our application. This made basic framework parts such as the diagnosis and configuration layer unavailable and we need to adopt our software accordingly using static compiled parameters. Furthermore it took us some effort to
setup the emulation layer accordingly to the target environment. While the virtual network for IP-based inter-processor communication could easily be connected, we needed additional effort for the classic vehicle networks as described below.

**Communication & Network Layer**

As already mentioned the IP-based communication between the partitions was easy to adapt and replaced by the virtual LAN modules in our Linux distribution and the container stack. Also time synchronization between the partitions and an external simulation clock could be achieved by this network layer. For the removed sensor inputs and peripheral components we used our internal middleware platform as described above and where able to stream even high resolution image data in real-time in our sensor-preprocessing subsystem.

For the classic automotive communication we needed the highest adaptations. Since we want to connect our Virtual HCP to our existing virtual integration environment [2] we implemented a bridge between the Safety subsystem and our existing vehicle network simulation. Here any virtual ECU Emulation Software like Synopsis Silver, dSPACE VEOS or Network Simulations like Vector CANoe could be considered. Since VEOS was available, we used this as endpoint for the network bus simulation for the beginning. In the connected virtual integration framework then all the residual bus simulation, connection to other Virtual HCPs and the virtual environment, sensor and traffic simulation is connected to a holistic simulation environment.

**Results & Conclusion**

Bringing basically all microprocessor software parts of a HCP into our virtualization environment was rather simple. This can be easily explained as microprocessors already share the same architecture to common consumer hardware and both worlds are getting closer by the massive performance increase microprocessors gain over the past years. Since the real-time operating systems on those platform are mostly UNIX or at least POSIX based, cross compilation of the higher software frameworks is also easy achievable and in most cases already delivered as part of the development kit with the target environment. Missing peripheral interfaces such as hardware sensor frontends and raw data processors could be stubbed and the data input could be established via our internal simulation middleware [10]. For future HCP generations this is a very positive trend since at least with first microprocessor operating systems being ASIL-D certified [9] most of the relevant application software to be tested in end-2-end tests will move there.

Until this evolution has been completed and legacy software exists our evaluation shows a big conceptual virtualization gap between the microcontroller with classic software stacks and the...
microprocessor subsystems. Bringing both worlds together required a lot of conceptual implementation. Most of the available virtualization software is used by emulating the base software stacks which adds an additional error component to the test environment itself. At time of our evaluation most target implementations where only available for typical microcontrollers and could not be run natively on x86 systems or comparable, compiled in a single execution tree for example. This leads to the problem that third-party modules from different Tier-1 or Tier-2s must be made available for the Emulation Software starting with the specific run time environment implementation. Otherwise the integrator and tester is left to redevelop the base software stack for application testing. An alternative is close alignment over software tools and processes over the complete software supply chain right from development beginning. The second high conceptual implementation effort was spend to the classic network systems. In order to combine a third party vehicle network simulation with our container based HCP virtualization. Since most of the tools require a complete development of virtual ECU and vehicle network definition within the closed toolchain our bridge approach overcomes this limitation which we would have rather like to avoid. We propose the introduction of a new open standard for distributed simulation of multi-partition and vehicle network communication as a solution for this issue.

Summary & Outlook
Our evaluation has shown the gap of HCP architectures consisting of modern high performance microprocessors and classic microcontroller parts as they still currently take an important role for safety critical function software parts. This gap has shown fundamental impacts on the virtualization of those systems to meet the requirements of highly scalable test environments. With these results we want to encourage further work on two aspects. First, continue our effort in standardization groups especially in integration of complete partition based HCP virtualization into Virtual ECU frameworks from the various tool vendors and second to introduce an open standard for distributed network communication simulation between different tools. Furthermore we will continue to encourage automotive runtime software distributors especially for the microcontroller stacks to deliver compatible environments for full HCP virtualizations at a virtualization level of 3 or higher as defined by the V-ECU Whitepaper [8].

Special thank is given to XTRONIC GmbH, which supported us especially in the implementation of the conceptual parts of our evaluation work.
References


