LAUTERBACH
DEVELOPMENT TOOLS

ENABLING THE AUTOMOTIVE FUTURE

# Debugging the Software Defined Vehicle (SDV) – How to be "SDV Ready"

*With the transition from traditional, domain-based E/E vehicle architectures to the software-defined vehicle (SDV), the requirements for development tools are increasing to a considerable extent. New heterogeneous multi-core high-performance chips and a highly complex software stack in a virtualized environment limit the choice of suitable debug and trace tools. However, there are already perfect "SDV ready" solutions for these complex architectures.*

## Introduction

The term "software-defined vehicle" (SDV) describes a vehicle whose features and functions are mainly enabled by software. This is the result of the progressive transformation of the automobile from a mainly hardware-based product to a software-centered electronic device on wheels.

Premium vehicles today can already contain up to 150 million lines of software code, distributed across up to 100 electronic control units (ECUs) and a growing number of sensors, cameras, radar and lidar devices. Mass market vehicles are not far behind. Three powerful trends – electrification, automation and connectivity – are changing customer expectations and prompting manufacturers to increasingly rely on software to meet them.

In the past, vehicle manufacturers differentiated themselves through mechanical features such as power and torque. Today, consumers are increasingly looking for software-defined features, such as driver assistance functions, infotainment innovations and intelligent connectivity solutions.

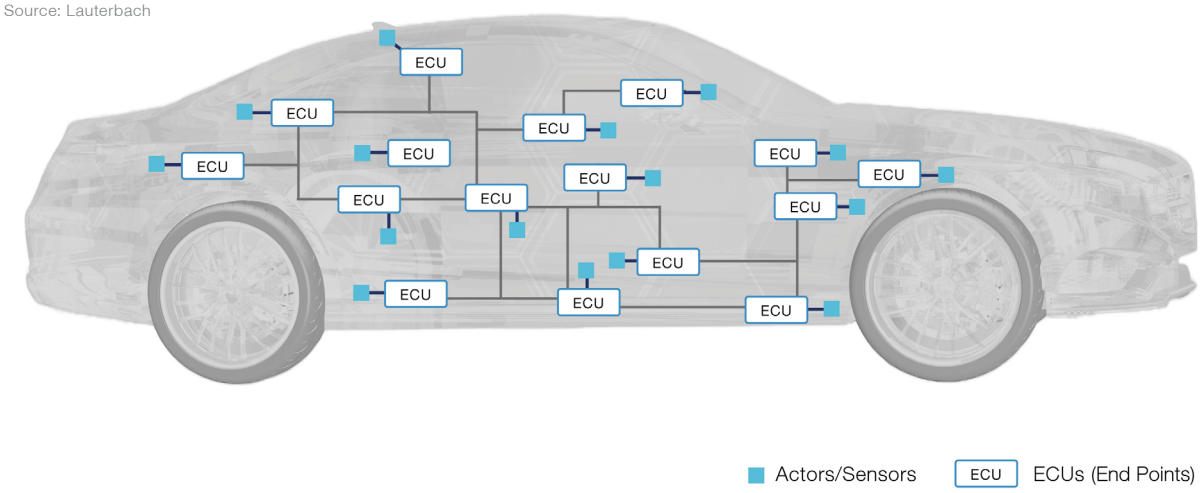
As driver assistance functions grow towards automated and fully autonomous driving, so does the need for more software. As consumers expect more rich content in their infotainment systems, the amount of digital content that the vehicle needs to manage is also increasing. And as vehicles become part of the Internet of Things (IoT) and transfer large amounts of data to and from the cloud, software is needed to process, manage and distribute all this data.

The software-defined vehicle not only offers new safety, comfort and convenience functions, but also has a number of other advantages over its hardware-defined predecessor.

Today, software upgrades for infotainment, telematics or vehicle diagnostics systems require a trip to the dealership. With a software-defined vehicle, customers can not only receive over-the-air (OTA) updates that include safety patches, infotainment enhancements and monitoring and tuning of core vehicle functions such as powertrain and driving dynamics, but can also download new features from an app store.

ECUs will send and receive vast amounts of data to and from sensors and actuators, giving vehicle manufacturers insight into every aspect of a vehicle, its performance and its place in the connected ecosystem. This gives vehicle manufacturers the opportunity to improve lifecycle management and develop revenue-boosting features to offer their customers – all of which will lead to deeper, more connected relationships with customers.

Figure 1: Traditional E/E architecture in the vehicle.
Source: Lauterbach




## From Hardware to Software

Figure 1 shows a traditional domain-based vehicle architecture. It divides the vehicle into different domains for the purposes of structure and hierarchy. Similar functions for the chassis, powertrain, body or infotainment are grouped together in these domains. If ADAS is subsequently added, for example, a new domain must be created and its control unit connected to the central gateway. Each feature or function is represented by its own control unit (ECU). The microcontroller of the ECU is selected precisely for the software mapping this function and together with the hardware, the ECU is set up for this function, e.g. as an "ECU for electric windows" or "ECU for airbag activation" or whatever. In line with the limited functionality, the computing power requirements and memory capacity of the microcontrollers are kept within narrow limits compared to SoCs (system-on-a-chip), such as those found in smartphones.

In an SDV, features and functions are consolidated on fewer, but significantly higher-performance chips. Figure 2 shows an example of a proposal for the gradual transition from a domain-based to a zone architecture with a central computer.

In a zonal architecture, the functionality shifts from domain controllers to zonal control units and central processors. Instead of dividing similar functions into domains, they are assigned to a zone depending on their location in the vehicle. This approach significantly reduces the complexity of the vehicle electrical system and facilitates standardization. The devices in a zone are controlled by one or more ECUs, which in turn communicate with a local host or zonal gateway. Several data streams and functions thus share a single piece of hardware.

A simpler car could consist of three or four zones in this sense, and six zones would be conceivable for more complex vehicle classes.

The design of the zones is variable – including the elements in the hierarchy. In addition, the changeover does not have to happen abruptly. For example, the three development stages shown in Figure 2, which are considered likely by various manufacturers, are possible.

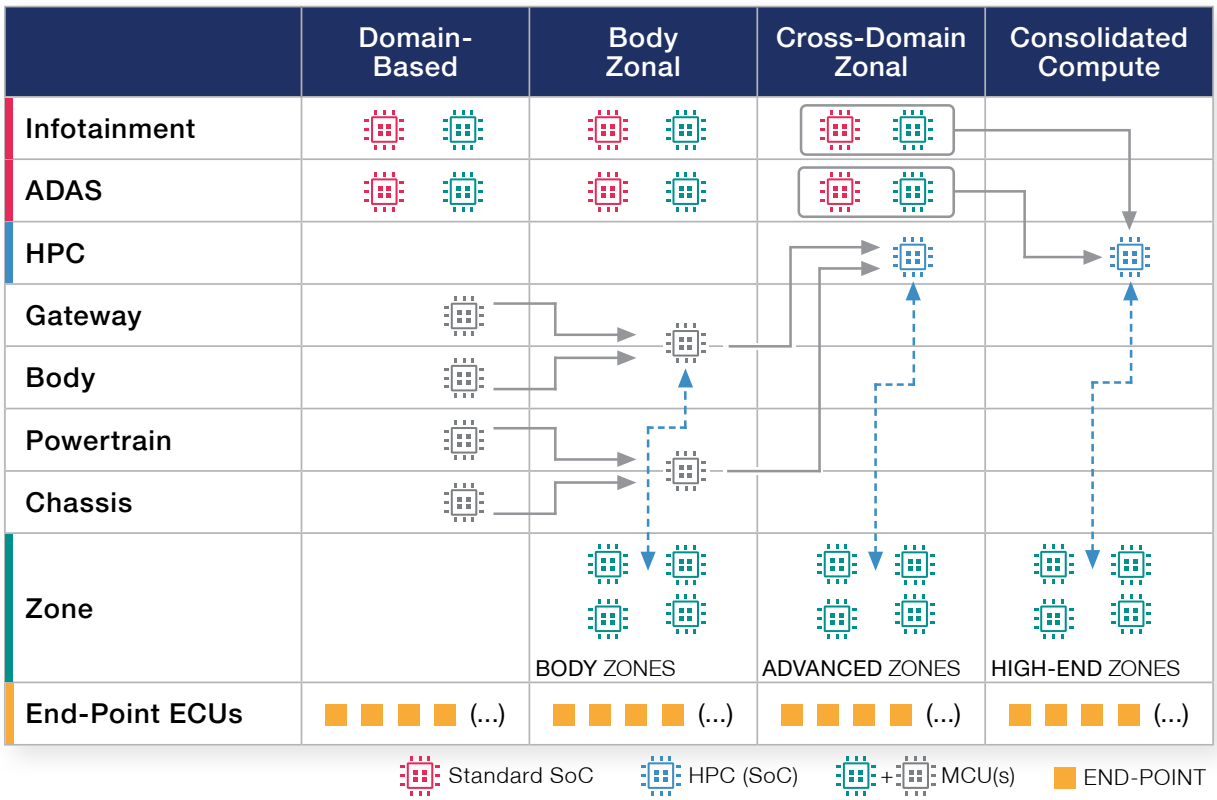



Figure 2: Consolidation of chips in the SDV.

Source: Lauterbach

## From the Domain-based Architecture to the Software Defined Vehicle (SDV)

The body domain, with its many distributed actuators and sensors as well as sensor ECUs, will be one of the first domains to transition to a zone architecture. This first approach reduces the complexity of the wiring harnesses within the vehicle. Devices are connected to a zonal gateway locally, reducing the number and length of data and power cables. The zonal gateway functions not only as a processing center, but also as a power distribution module. The communication links between the gateways and the central computing cluster can be established with just a few high-speed network connections. This means that the redundancy required in safety-critical systems is also possible with less material input.

The second phase is about much more than "just" reducing weight through less complex cable harnesses. Here, more and more computing power is required in the individual control units and the central computer unit. The number of ECUs is consolidated and the central computer becomes the brain of the vehicle. Instead of outsourcing the chassis control together with the drivetrain via a motion domain controller, for example, it could become part of the zones.

The ultimate goal of the zone architecture is a fully software-defined vehicle that ideally combines standardized components for sensors, actuators, zonal modules and data connections. The zonal aggregator is responsible for traffic management

and real-time processing. It combines different types of data with different criticality, safety and security requirements. It must ensure both access to and exit from the data highway and finally the reconversion of Ethernet packets into CAN or LIN bus signals without compromising the above-mentioned requirements of each individual bit of information.

The zonal controller is responsible for extended network processing. Sensors, actuators and sometimes functions are connected locally to the zoned controller. These can also be regarded as mini-gateways, as they combine CAN or LIN with Ethernet. Finally, the zone ECUs are connected to the central brain of the vehicle via a high-speed Ethernet network backbone.

In addition to all these functions, the zone processor is able to perform application processing and advanced gateway services. This architecture places high demands on data transmission, the microcontrollers used and the edge intelligence. However, the advantages are obvious: the controllers are reusable and can be easily updated, as the necessary functions are available together in the central computer.

Figure 3 shows a proposal that contains four zone controllers in addition to the endpoint MCUs on the sensors and actuators. The requirements description above makes it clear that these must be high-performance microcontrollers with real-time capability and features for functional safety. Chips suitable for this are being developed by various semiconductor manufacturers or are already on the market.

At the top of the hardware list for the central computer in Figure 3 are three so-called high-performance SOCs, which are high-end chips such as Qualcomm's Snapdragon, Nvidia's Drive Orin/Thor, NXP's S32N55 Super Vehicle Integration Processor or Renesas' 5th generation R-Car. Other semiconductor manufacturers are currently developing similar SoCs for HPC. These include massive multicore architectures with the most powerful Cortex-A/R cores currently offered by Arm®, as well as additional cores for accelerating special tasks (e.g. Hexagon DSPs in Snapdragon, CEVA-X DSPs in certain R-Car derivatives).

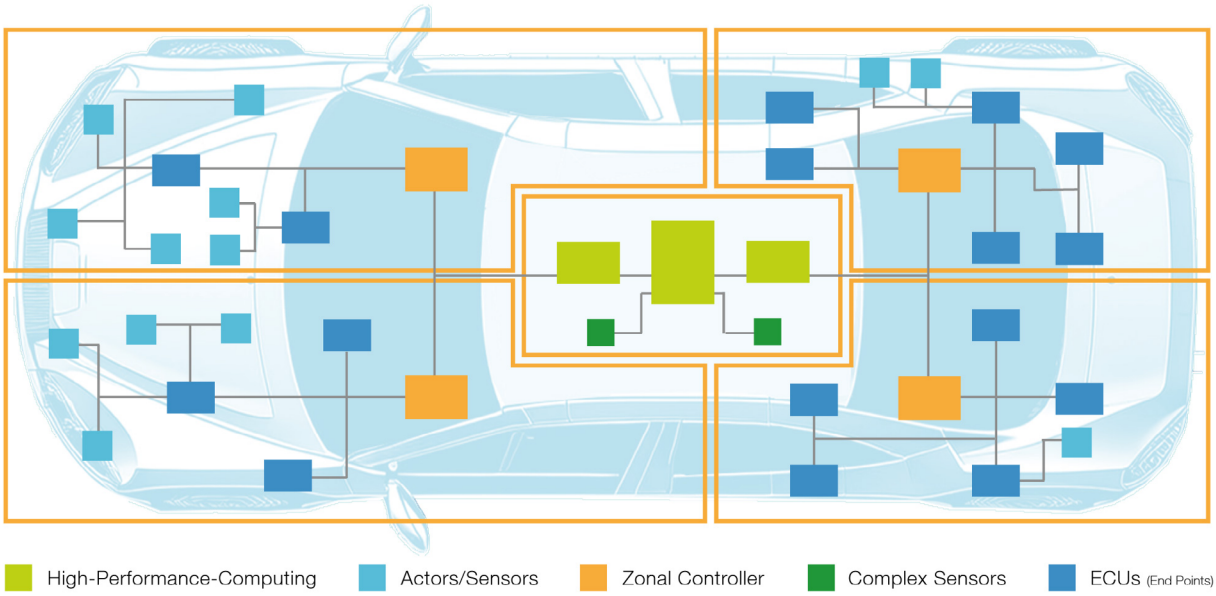



Figure 3: Typical SDV vehicle architecture.

Source: Lauterbach

## Hardware Requirements for Debug and Trace Tools

Even if there is still no standard definition of "the" SDV architecture, there is general agreement on the type of hardware and software stack that will be found in SDVs.

It sounds trivial, but a debug and trace tool must of course support all chips installed in the SDV as a first step, primarily meaning effective multicore debugging, especially for heterogeneous chip architectures with a large number of different cores. These so-called AMP systems have several cores, each of which fulfills its own task.

Lauterbach's TRACE32® PowerView software provides a consistent user interface and feature set for any mix of core architectures. You can debug all types of multicore configurations with up to 16 synchronized PowerView instances via a single debug probe [1]. Within your system, each core can have its own core architectures, memory models, operating systems, address translations and debug symbols, as well as its own physical address space. You can even combine several separate multicore SoCs for debugging via a daisy chain (JTAG), a star topology (cJTAG/SWD) or using two whiskers with a TRACE32® CombiProbe.

Sometimes traditional stop-mode debugging is not sufficient, e.g. for heisenbugs that only occur in real time or if errors only occur occasionally or have complex causes. This is where program flow data provided by trace extensions can help, showing exactly which instructions were executed when and how long the execution took, without affecting the application under test. Code coverage measurements and timing analysis also require the collection of trace information over a long period of time.

With TRACE32® trace tools, developers can capture real-time traces on any multicore SoC that offers a trace interface or an on-chip buffer. It doesn't matter what type of multicore system you use: Symmetric Multiprocessing (SMP), Asymmetric Multiprocessing (AMP), Integrated Asymmetric Multiprocessing (iAMP) or any mixture. Modern multi-core SoCs merge the trace data of all cores so that you only need a single trace sample to capture the off-chip trace of all cores together.

Before deciding on a debugger or trace tool in an SDV project, questions such as "Does the tool actually support all chips including HPC SoCs?" and "Can I debug all cores including accelerators such as the Hexagon DSP on Qualcomm's Snapdragon?" are more than important.

## Software Requirements for Debug and Trace Tools

On the side of an SDV software stack, we are experiencing a significantly greater paradigm shift compared to hardware. Several consortia have been formed to develop a standardization here. According to a study by the renowned analysis company WardsAuto™, the SOAFEE consortium founded by Arm® has the best prospects of being successful here [2].

Figure 4 shows the highly complex software stack as envisioned by the SOAFEE consortium (www.soafee.io). The aim is to significantly decouple the previous co-development of hardware and software. The components are developed independently of each other. The integration of all modules, which was previously carried out by the OEM, is now carried out by the car itself during operation.

There are two crucial points here in particular: Firstly, the environment is virtualised. As we have discovered with data centres, this is a more efficient use of the computing power available. With the help of hypervisors, so-called virtual machines (VMs) are set up, each of which runs not only applications but also its own operating systems, strictly separated from other VMs. Secondly, container technology can be used to install new applications from the cloud without affecting existing workloads.

What does this mean for debug and trace tools? First of all, they must offer broad hypervisor and OS awareness. This means that users can debug the entire software stack from the user application to the device driver and query and display all operating system objects such as threads, message queues, etc. In virtualized systems, all VMs and their applications must be debuggable at the same time in order to create complete transparency (Figure 5).

On virtualized systems where multiple operating systems are controlled by one hypervisor, Lauterbach's TRACE32® Hypervisor-aware Debugging enables simultaneous OS-aware debugging for each guest operating system/virtual machine (VM) and the display of an overview of the entire system. In addition to static hypervisors, dynamic hypervisors that dynamically allocate memory resources and cores to the VMs are also supported – a unique TRACE32® feature in the entire embedded industry [3].

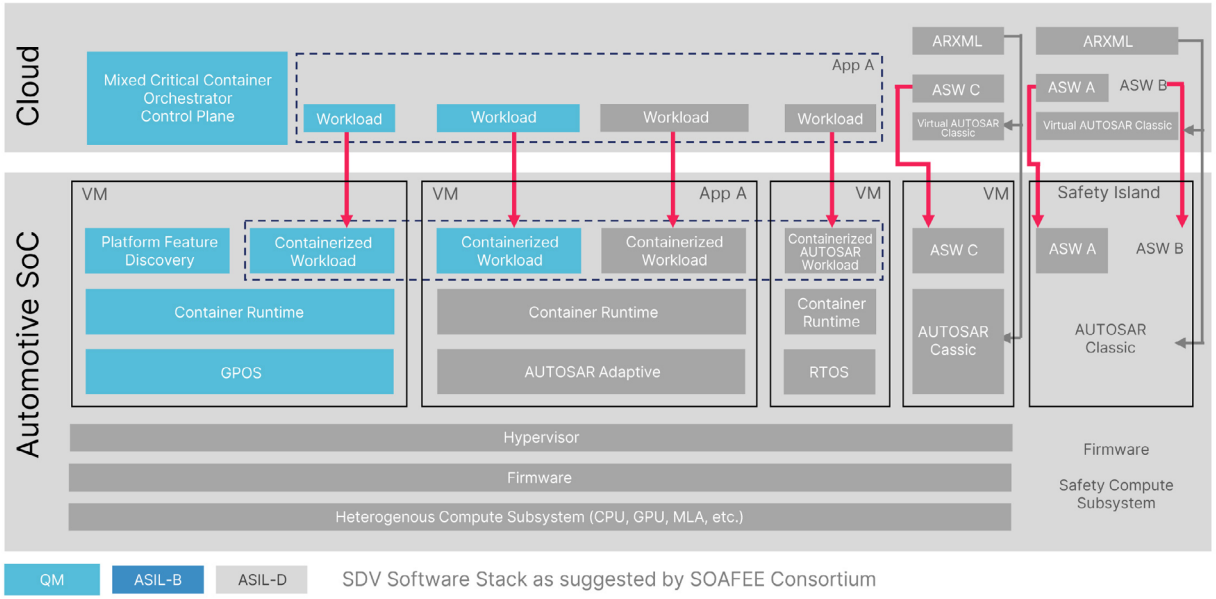



Figure 4: Software stack with virtualized environment and containers for SDVs.

Source: SOAFEE consortium

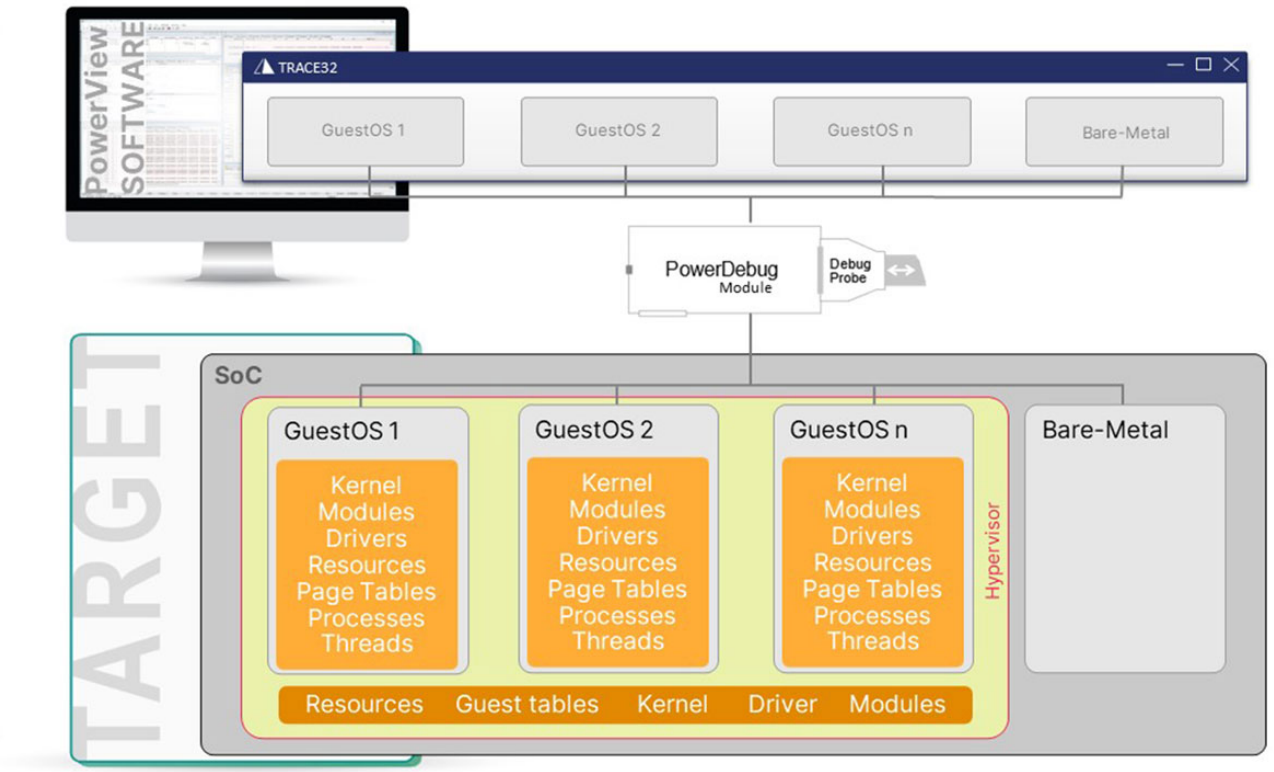



Figure 5: Parallel debugging of several guest operating systems on a hypervisor.

Source: Lauterbach

## Supporting Heterogeneous Operating Systems and AUTOSAR

TRACE32® OS Awareness is available for more than 80 embedded operating systems of all kinds. It supports Rich OSes as well as RTOSes and all common open source and commercial operating systems used in embedded applications.

To be able to read the task list or to enable process or module debugging, OS awareness accesses the internal structures of the OS kernel via the kernel symbols. TRACE32® OS awareness and TRACE32® MMU support enable seamless debugging across process boundaries: it is possible to directly access the virtual address space of each process and display the current register set and stack frame for each individual process. After loading an operating system-specific extension, the TRACE32® PowerView software offers additional commands, options and displays that simplify the debugging of the operating system.

OS awareness is of course also important when using AUTOSAR. The AUTOSAR Classic platform uses a static operating system that is based on the OSEK OS and has been extended by AUTOSAR with a range of functionalities. TRACE32® OS-aware debugging for AUTOSAR Classic based operating systems is activated by loading the ORTI file or the AUTOSAR ARTI description file. The TRACE32® PowerView software is extended by a dedicated menu that gives developers access to AUTOSAR OS resources such as tasks, alarms, stack coverage and much more. When running multiple AUTOSAR Classic operating systems on multicore SoCs, the TRACE32® tools also support simultaneous debugging of these systems. This also includes hypervisor debugging of virtualized systems.

In contrast to the Classic Platform, AUTOSAR uses a POSIX operating system (POSIX Profile PSE51) in

the Adaptive Platform. This means that Linux, for example, which has long been used in the automotive sector for infotainment systems and other applications, is also available as an operating system for the Adaptive Platform. In addition to Linux, the TRACE32® tools support all relevant POSIX-compatible operating systems such as QNX, PikeOS or eMCOS.

If several AUTOSAR Adaptive Platform-compliant operating systems are used on multicore SoCs, TRACE32® also supports the simultaneous debugging of these systems. This also includes hypervisor debugging of virtualized systems.

As can be seen in Figure 4, mixed AUTOSAR Classic/Adaptive components are used in the SOAFEE reference software stack. The TRACE32® debug and trace tools also support various configurations for the simultaneous debugging of AUTOSAR Classic-based and AUTOSAR Adaptive-compliant operating systems thanks to the almost unlimited support of any multicore configurations. A typical configuration is shown in Figure 6.

Before starting an SDV project, you should therefore ask yourself, for example, "Does the tool actually support all hypervisors and OSes, including all AUTOSAR-compliant OSes for my project?", "Can I debug all active and non-active VMs in virtualized environments at the same time and track the memory conversion from virtual to physical addresses via MMU (Memory Management Unit)?" and "Can I debug applications in containers, e.g. from Docker?". If the answer is "no", it is likely to become difficult at some point in the course of an SDV development project.

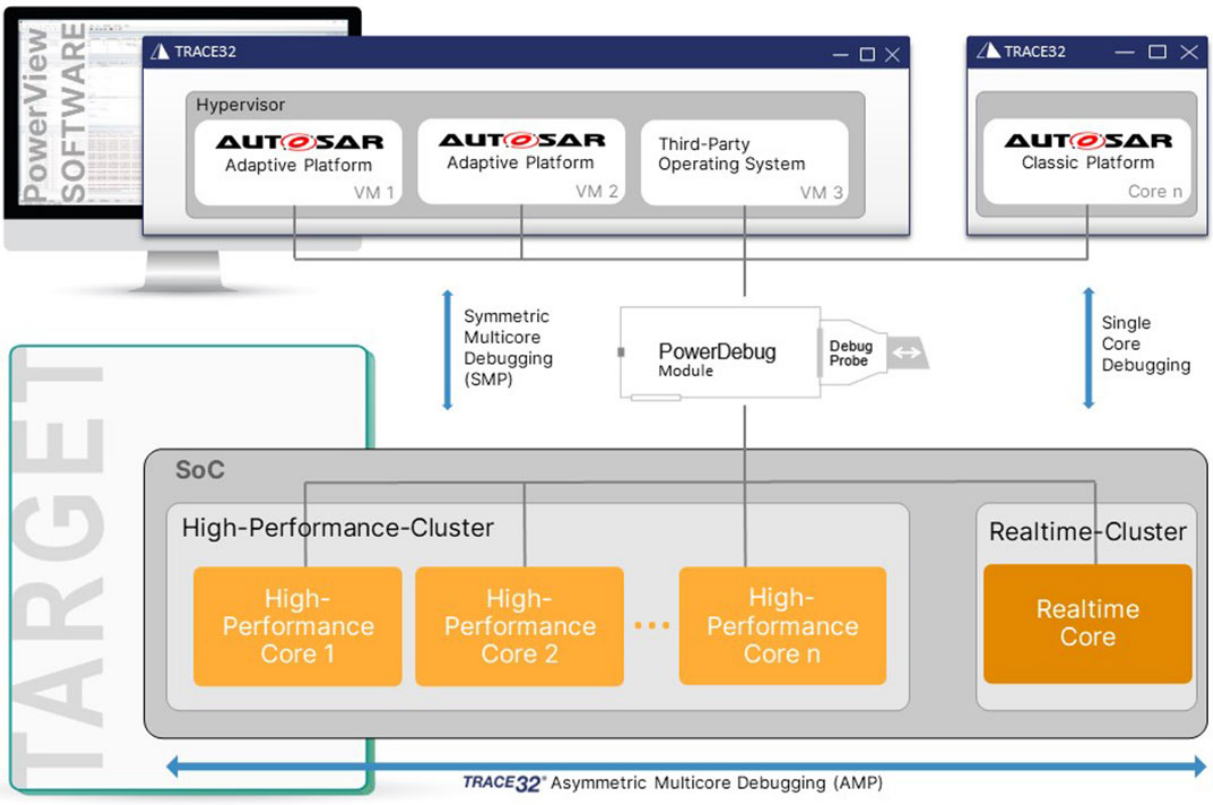



Figure 6: Simultaneous debugging for AUTOSAR Classic and AUTOSAR Adaptive Platform. Source: Lauterbach

## Development in the Cloud thanks to "Shift Left"

In order to be able to start software development as early as possible before real chips are available (in sufficient numbers), development is increasingly being started on virtual targets in the cloud. Providers such as ASTC, Correlium, Synopsys and others offer entire chips or even automotive platforms "in software" for this purpose. In a standard white paper [4] from 2020, five levels of virtual and real ECUs are described, in which the real production environment is increasingly approximated with ascending levels.

From the customer's point of view, it is of course desirable to be able to debug all levels of an ECU with one toolchain and one GUI so that there is no break in the development process. Lauterbach's TRACE32® supports the "Shift Left" approach by not only supporting real chips in silicon, but also virtual targets and emulators from various partner suppliers via different interfaces, on which software development is already possible if no real chips in silicon are available (Figure 7). The functions and the GUI of the TRACE32® PowerView software do not differ from real targets, so that the user experience is the same throughout the entire development cycle [5].

With its reference platform RD-1AE (Fig. 8), Arm has made a concrete proposal for the architecture of an SDV. The platform combines high-performance Arm® Neoverse V3AE application processors (primary compute) with an Arm® Cortex®-R82AE-based so-called security island for scenarios where additional system security monitoring is required. The system also includes a Runtime Security Engine (RSE) realized on an Arm Cortex-M55, which is used for the secure start of the system elements and the Runtime Secure Services. In addition, an open source software package is supplied with the Xen hypervisor and the Zephyr OS real-time operating system, among other things.

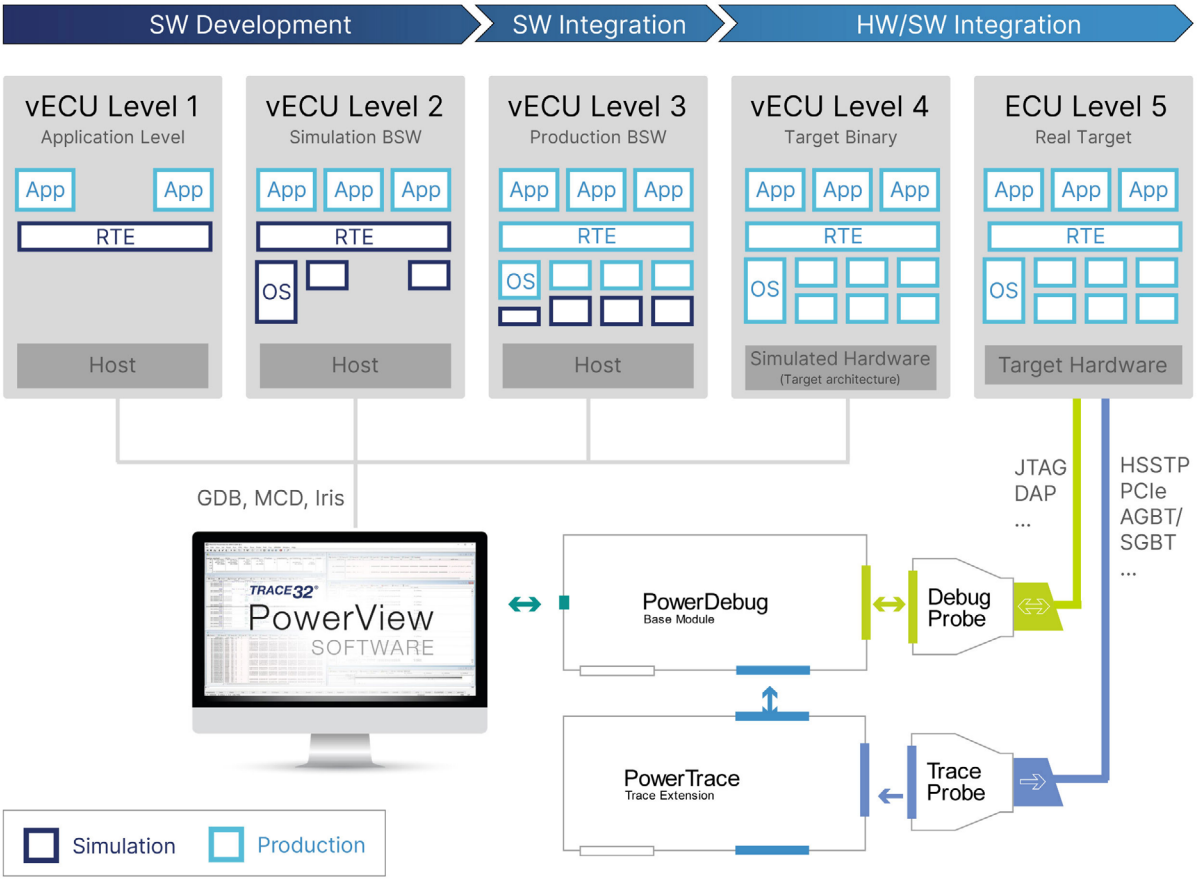



Figure 7: Debugging virtual and real ECUs.

Source: Lauterbach

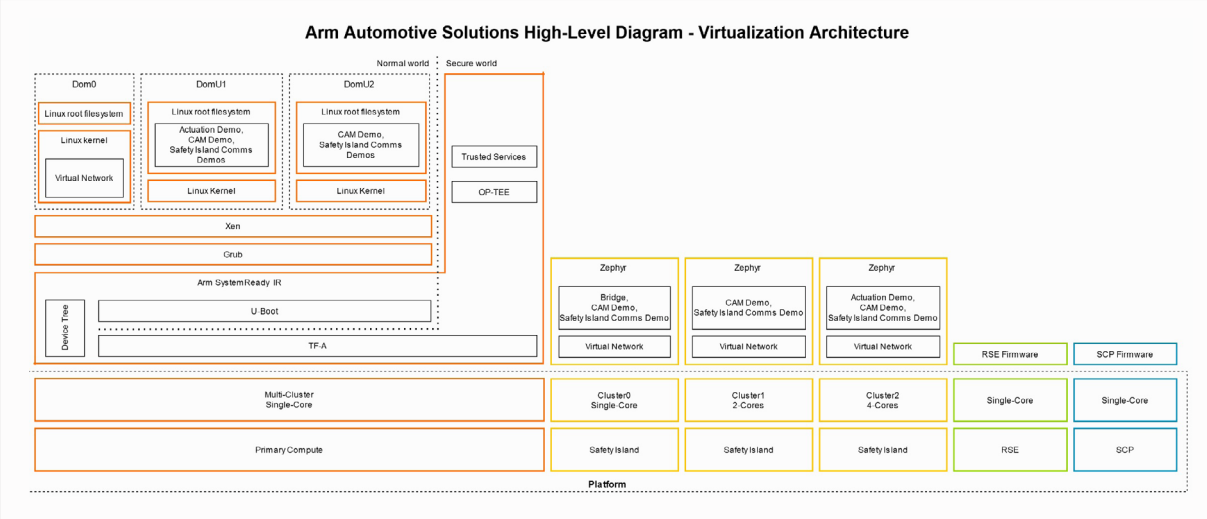



Figure 8: Arm Automotive reference platform RD-1AE. Source: Arm

## Debugging Arm's Automotive Reference Platform in the Cloud

The company Corellium has fully virtualized this platform and brought it into the cloud [6]. The special feature is that the virtual platform runs on AWS instances at Amazon Webservices with real Arm hardware (specifically: Amazon's Graviton Arm Server), so that the Arm instructions of the virtual Arm cores do not have to be migrated to x86, as is the case with many cloud solutions. Performance is therefore significantly higher.

TRACE32® is already capable of debugging the entire Corellium platform in the cloud, both the Neoverse V3 and the Cortex-R82AE and Cortex-M55 clusters (Figure 9).

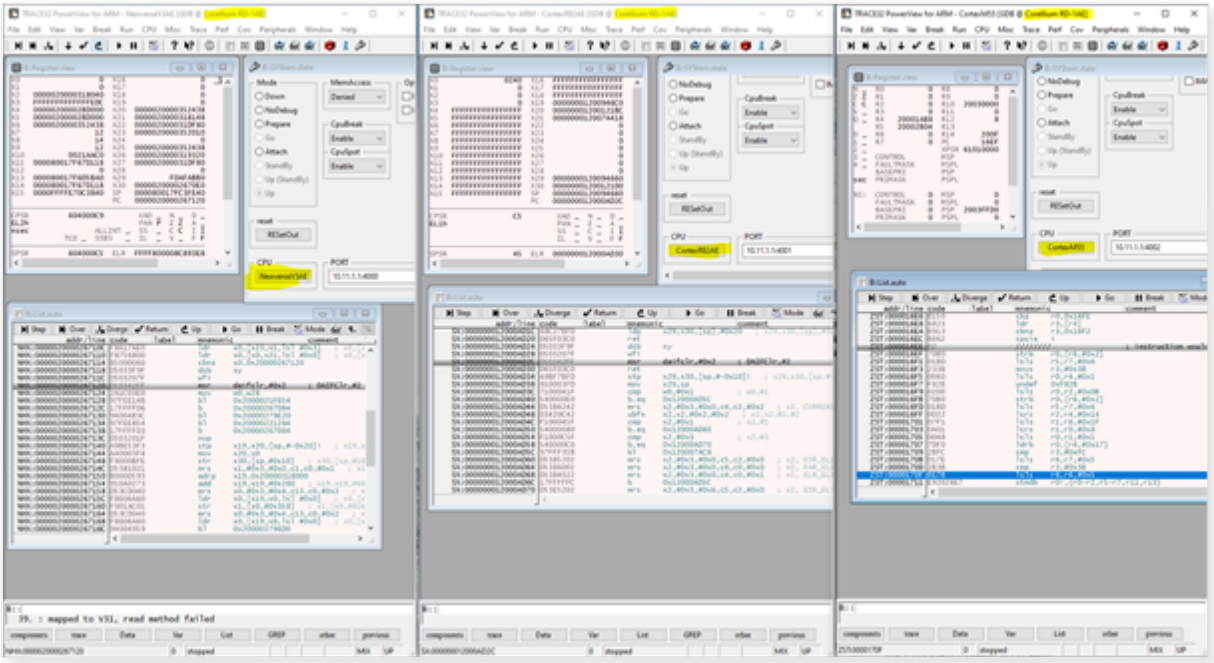


Figure 9: Debugging the Corellium Arm Automotive reference platform in the AWS cloud with TRACE32®. Source: Lauterbach

CONCLUSION:

## "SDV Ready" Tools make Life easier for SDV Developers

To summarize, the world for developers of SDV architectures is becoming more complex compared to the traditional approach. Not all tools support all new challenges from highly complex multicore SoCs to virtualized environments, containers and virtual targets. The good news is that Lauterbach's TRACE32® debug and trace tools are already "SDV Ready" today: With TRACE32®, developers can debug your entire SDV software stack on all current and future automotive SoCs and across the entire lifecycle from virtual ECU to real silicon.

REFERENCES:

[1] Unlimited multicore debugging with Lauterbach TRACE32®: https://www.lauterbach.com/features/multicore-debugging-and-tracing

[2] WardsAuto Study "Unveiling Tomorrow's Ride: A Deep Dive Into Software- Defined Vehicles", page 19. Downloadable from https://www.nxp.com

[3] Hypervisor and OS-aware debugging with Lauterbach TRACE32® : https://www.lauterbach.com/features/os-awareness

[4] Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs): https://www.ps-ent-2023.de/fileadmin/prod-download/WhitePaper_V-ECU_2020_05_04-EN.pdf

[5] "Shift Left Debugging" with Lauterbach TRACE32®: https://www.lauterbach.com/supported-platforms/toolchain/emulators-and-virtual-targets

[6] Corellium Arm Virtual Automotive Platform: https://www.corellium.com/blog/introducing-rd-1ae

[7] Debug SDVs with Lauterbach TRACE32®: https://www.lauterbach.com/sdv