SYNOPSYS®

WHITE PAPER

Accelerating Development of Software Defined Vehicles with Virtual ECUs

Author

Filip Thoen
Ph.D., Synopsys Scientist—Virtual Prototyping Product Architect

Overview

The automotive industry is going through a revolution. To adapt to new customer demands such as convenience, safety, autonomy, and electrification, the automotive industry is moving to software-driven vehicles. These require new, more powerful electrical/electronic (E/E) architectures and significantly increase the vehicle software content. They also force the industry to move from lengthy development cycles and complex maintenance schedules to a more agile development approach allowing for continuous over-the-air (OTA) updates over the vehicle lifespans.

This white paper addresses one of the key changes: the move from software development and validation using physical electronic control units (ECUs) and benches to digital twins using simulation of ECUs, called virtual ECUs (vECUs). vECUs bring important benefits such as front-loading of the validation effort, greater testing scalability, and better ecosystem collaboration. This paper includes a comprehensive definition of the various vECU types, their abstraction levels, and their applicability during software development and validation.

Automotive Industry Challenges

Delivering to the expectation of consumers while achieving the expected return on investment is a key objective of the automotive industry transition. This transition is challenged by several factors, including the electronic supply chain disruption, the overhaul of E/E architectures for software-defined vehicles, the complex software development and deployment, increasing security challenges, and accelerating time to market to compete. Each of these items has a direct impact on the electronic systems (hardware and software) development.

The supply chain disruption ranges from chip shortages compressing the supply chain to new entrants addressing computing requirements with new system-on-chip (SoC) designs. This requires software development to be decoupled from hardware development such that these two development tracks can proceed in parallel. Digital twins for electronics system development enable this decoupling.

The overall E/E architectures for software-defined vehicles are driving the need for new compute platforms that are more powerful, more power conscious, and more scalable. Moving from domain controller to zonal controller and central compute is directly impacted. This also requires new vehicle operating systems that need to be developed earlier. Digital twins enable development to start sooner.

Millions of lines of code ensure that the vehicle operates correctly. Electrification, advanced driver assistance, and in-vehicle infotainment require more software that operates interdependently to serve the desired function. The validation of these systems cannot happen in isolation, late in the development process and using expensive test benches and mule vehicles. Most of the functional problems need to be addressed earlier and in a more productive environment. Digital twins provide a highly productive validation environment for such task.

In addition to functionality, safety and security are key concerns for automotive companies. New vulnerabilities need to be accounted for, and compliance with safety standards such as ISO 26262 creates additional pressure on already compressed development cycles. Transitioning to digital twins for electronic system development enables developers to take into account these aspects earlier than before and to accelerate time to compliance.

New entrants are challenging traditional development, creating increased competitive pressure and the need to accelerate time to market. Achieving start of production (SOP) with accelerated development is not sufficient anymore; the ability to continuously validate and deliver software updates and new functionality via OTA updates has become a key differentiator and business opportunity. Automotive companies need to ensure that these update cycles are fast. Digital twins accelerate the validation of OTA updates through faster turnaround. Collaboration across the new automotive ecosystem is now, more than ever, key to future success. Digital twins provide an essential collaboration platform to foster innovation, accelerate development, and deliver better products faster.

Digital Twins for Electronic Systems Development Are the Key to Success

The introduction of digital twins and use of virtualization-based testing with vECUs is a key element to address the challenges associated with ECU development. As shown in Figure 1, there is a clear trend in the industry to transition from traditional physical bench testing to virtualization and simulation-based testing. This has several major benefits:

- Early software bring-up before physical ECUs or benches are available
- More productive system testing, due to deterministic reproduction of issues and a high degree of debug visibility
- Support for the rapid pace of agile development and integration into continuous integration/continuous deployment (CI/CD) systems
- Functional safety and security validation, by using advanced features such as fault injection difficult to realize with physical systems
- Easy supply chain collaboration due to the software nature of vECUs

Figure 1: Change in automotive electronics development

One of the reasons that pre-silicon software development is so critical is the amount and complexity of code that must be written and validated. Automotive software stacks are growing due to several trends. At the vehicle level, a new zonal architecture is being introduced, with complex hardware based on advanced SoCs and a centralized high-performance computing (HPC) engine. At the same time, there is a move towards a more service-oriented software architecture, with easy discovery and updateability, and less monolithic applications.

Traditionally, the software running on ECUs has followed the Automotive Open System Architecture (AUTOSAR) Classic standard. Recently, operating systems such as Linux and QNX complaint with the Portable Operating System Interface (POSIX) standards have become increasingly popular. A mix of non-POSIX and POSIX software is now commonplace at the vehicle level, within a single ECU, and even running on a single microcontroller (MCU) or SoC.

Figure 2 shows an example of such a converged software architecture, in which ECU consolidation causes multiple mixed-criticality software stacks to share the same ECU or MCU hardware. The example shows a combination of POSIX and non-POSIX (AUTOSAR Classic in this case) stacks, for instance for in-vehicle infotainment (IVI) and a driver assistance function (ADAS). This multiplexing is typically enabled through use of a software hypervisor. These stacks might have quite different safety levels, presenting the additional challenge of developing and validating software with mixed criticality.

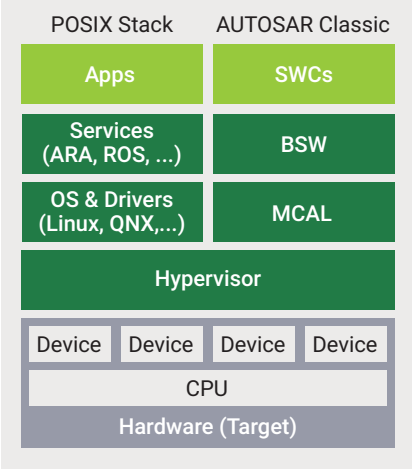

Figure 2: Converged ECU/MCU software architecture

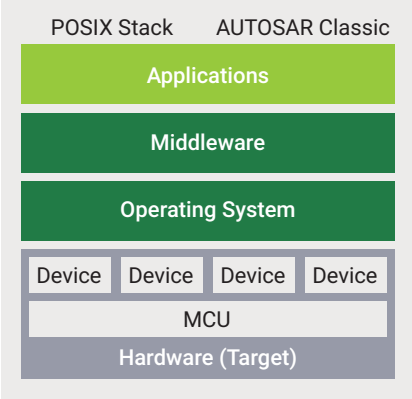

Figure 3: Levels of automotive software stacks

Another emerging trend related to POSIX is the use of the Virtual I/O (VIRTIO) device open interface promoted by the OASIS standards consortium. Though originally originating from the HPC market, VIRTIO-compliant device drivers are making their appearance in automotive software stacks, making it easier to move the software stack between different hardware and reuse software stacks.

vECU Classification: Types

The simulation of ECUs as vECUs has found rapid adoption in several phases of automotive development. However, a comprehensive definition of the various vECU types, their abstraction levels, and their applicability has been missing in the industry. Existing virtual ECU classifications focus on AUTOSAR Classic and use its terminology of Software Components (SWC), Basic Software (BSW), and Microcontroller Abstraction Layer (MCAL).

As POSIX operating systems have made their way into vehicles, a more generic vECU terminology is needed. Synopsys recommends the terms Applications, Middleware, and Operating System (OS). For simplicity, the hypervisor is considered part of the OS. Broadly speaking, there are two distinct types of vECUs: host compiled and target compiled.

In a host compiled vECU, the ECU software is cross compiled to the simulation host (typically an Intel x86 server or desktop). For example, if the host is an x86-based machine then the x86 compiler is used rather than the compiler for the MCU that will run the software in the physical ECU. A target compiled vECU uses the compiler for the specific target automotive MCU or SoC, containing CPUs such as Arm, RH850, or TriCore.

The host compiled vECU contains no model of the ECU hardware. Instead, simulation of software APIs is used at various cut points in the software stack, to remove lower, hardware dependent software layers. Thus, the vECU is not running full production code and software modification is required for certain layers. Each modification is a focused replacement of a specific software layer not part of the system under test (or SUT) with a simulation equivalent.

In contrast, a target compiled vECU uses detailed hardware simulation models to represent the ECU hardware. This allows the production software to be run with no modifications. The software is compiled exactly as it would be for the physical ECU in the actual vehicle. This requires detailed hardware modeling of the complete ECU, its SoCs and/or MCUs, and their internal CPU cores, components, and even board-level devices. With Arm hosts now readily available and the Arm architecture being increasingly adopted as automotive ISA architecture, host and target are closing in on parity and new opportunities open up for vECU virtualization.

There is a clear trade-off: host compiled is faster but less accurate, while target compiled is slower but more accurate. These characteristics dictate their primary use. Host compiled is best for tackling the higher software layers and target compiled is best applied for hardware dependent software and full-stack validation of all software layers together. Figure 4 shows both types of vECUs and introduces the abstraction levels they support. The color-coding highlights production software, simulation replacement, or bypassing of a software layer.

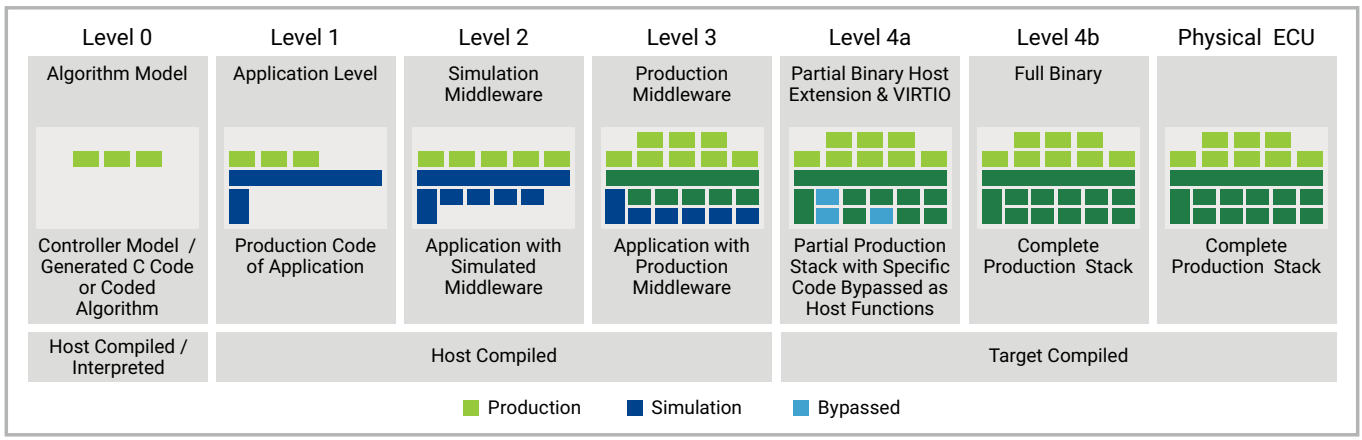

Figure 4: vECU types and abstraction levels

vECU Classification: Abstractions

Level 0—In Figure 4, Level 0 focuses on the design of the algorithm or (control) model. Programmers can either use model-based design, for example with the MathWorks MATLAB environment, or develop manually, such as C/C++ code or even machine learning models for ADAS or Autonomous Drive (AD) applications. The code is simulated on the host as host compiled code or via an interpreter. This falls outside of the scope for this paper, as does the physical ECU (right side of Figure 4), which logically requires target compiled code.

In the host-compiled vECUs, Level 1 to Level 3 is a logical progression of adding more and more production code at lower software layers. In Figure 4, the green represents production software and dark blue represents simulation substitution. In the target-compiled vECUs, Level 4a and Level 4b run the actual target binaries, enabled through detailed hardware models. Level 4a differs from Level 4b in that certain software drivers or complete layers are bypassed on the target and executed as host functions rather than using a detailed hardware model.

Level 1—This level is called the Application Level, with the testing focus on the production application or a module of it. The middleware software layer is removed, and then supplemented with basic run-time and I/O simulation code, allowing the application to be scheduled and to perform I/O communication. For example, in the case of AUTOSAR Classic, this simulation code consists of the Run-Time Environment (RTE), Operating System, and I/O code to send and receive signals.

The application or module is compiled into a host executable. The communication of the code and its compiled vECU is typically at the signal level, as opposed to the bus or network level (i.e., packet level). This type of vECU enables early module testing without target availability, and this type of vECU simulation can be easily integrated into a CI/CD flow.

Level 2—A level 2 vECU uses Simulation Middleware to expand the scope to a complete application and bring in the interfaces from the application to the middleware. The production code of the complete application is used, but the middleware is replaced by a simulation equivalent as indicated by the blue color in Figure 4. Again, the application is typically compiled into a host executable together with the Simulation Middleware.

The communication of the vECU can be optionally modeled at the signal level or at the bus/network level using packets. Level 2 enables a broader validation scope, including not only the functional basics of the application, but also extended functionalities such as bus monitoring and diagnostics. It also enables validating networks of ECUs, including the complex definition and configuration settings of these networks. Integration within CI/CD flows is possible at this level too.

Level 3—This level adds even more production software to the bottom of the stack: Production Middleware is included, and the device drivers are substituted with a simulation equivalent. Typically, the vECU interfacing is modeled at the bus/network level. In Level 3, only the hardware dependent parts of the ECU software (such as the driver layer) are missing but all the hardware independent software layers can be validated. Again, integration with CI/CD is possible.

For target compiled vECUs, Level 4b is the classic Full Binary case. In Figure 4, all boxes in this level are green, meaning that production code is used for all the software layers: Application, Middleware, and OS. No simulation substitutions are required. The software stack is compiled with the same compiler as the physical ECU and produces the identical binary target executable.

This binary compatibility is enabled through a complete set of hardware models covering all the internals of the ECU, including MCUs, DRAM, timers, and network controllers (e.g., CAN or Ethernet). Communication modeling is typically done at the hardware bus/network level.

Level 4b—This level enables the widest use cases, all the way from application and middleware to OS and drivers, and even the hypervisor. Users can validate complex system boot scenarios, which typically involve certification validation, and actions such as ECU re-flashing, for instance as part of an OTA update. This level's completeness allows to it perform functional safety testing, for example by injecting hardware faults in simulation, as well as security testing. CI/CD can be used to continuously test the full stack.

Level 4a—This level is a unique approach introduced by Synopsys. Its principles are the same as Level 4b, with the difference that certain software functions in the full stack can be bypassed. They are executed as host functions rather than the normal combination of a target software layer and its underlaying hardware models. Two technologies can be used for the bypass step: Host Extension and VIRTIO driver substitution.

Host Extension, a CPU instruction set simulator (ISS) technology, traps API calls to the software function or layer to be bypassed, passes control to the host function, and patches results back into the target software stack. The other technique, VIRTIO driver substitution, leverages the increasing adoption of this standard in automotive software. If the stack already supports VIRTIO, it is easy to change the hardware driver to a simulation driver.

Level 4a offers two important advantages over Level 4b. The set of required hardware models is reduced, enabling full stack validation earlier while models are being readied or eliminating the need to model everything. This level also speeds up the vECU simulation by the use of faster host functions. Leveraging a compute intensive host GPU rather than a slow GPU hardware simulation model is a common example.

Mixed Abstraction Levels—vECU levels are not required to be identical throughout a domain/zone or vehicle level simulation. Figure 5 illustrates mixing vECU levels in a single simulation. Note there is a sliding scale between Level 2 and Level 3. In the same software stack, some of the middleware might be implemented as Level 2 (simulation code) while others might be at Level 3 (production code). It is also possible to include Level 4a in the mix so that some lower levels of software can be executed.

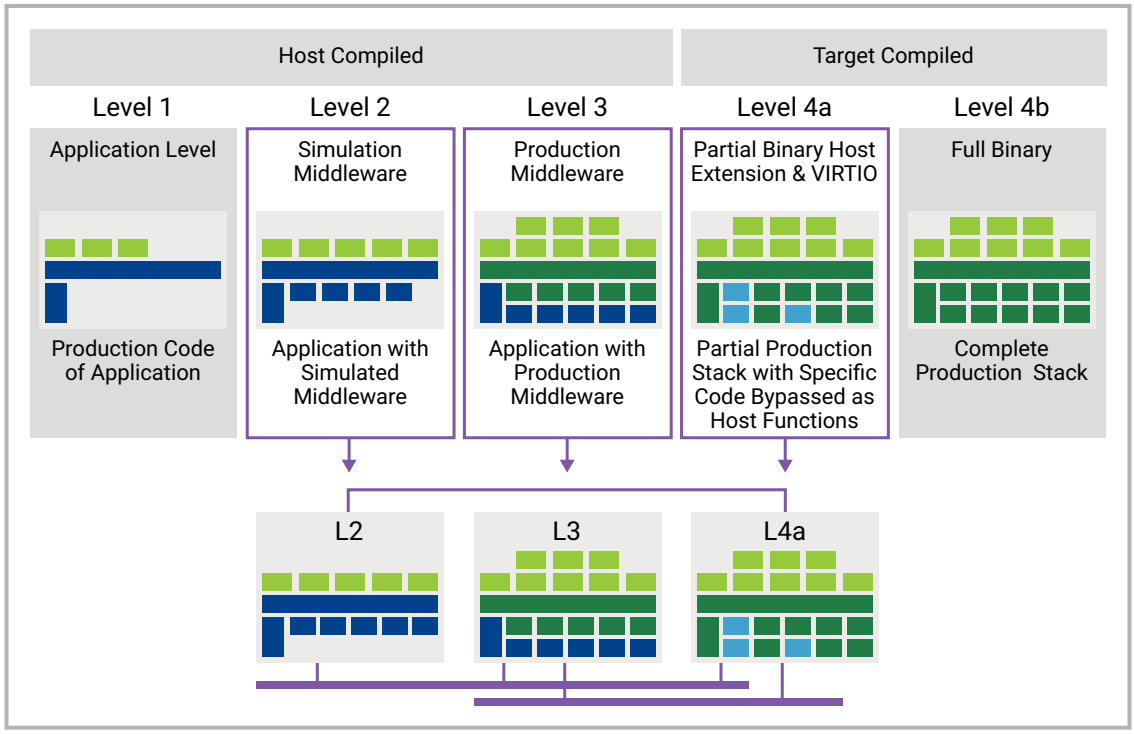

Figure 5: Mixed-abstraction vehicle level simulation

vECU Classification: Comparison

As previously discussed, each of the vECU levels has its own advantages. Some levels may be more appropriate and useful than others at specific stages of an ECU development project. Figure 6 contrasts and compares the different levels. The left side plots the number of test cases, representing the validation scope, against the degree to which the vECU models the actual ECU hardware. Of course, other aspects including vECU (modeling) effort and resulting simulation speed should be considered on a multi-dimensional plot.

The graph shows how the higher levels are more realistic and support more test cases. However, the effort involved and the simulation speed are inversely proportional. The gray areas between Level 2 and Level 3, and between Level 4a and Level 4b, reflect the increased scale of applying middleware substitution and host bypassing.

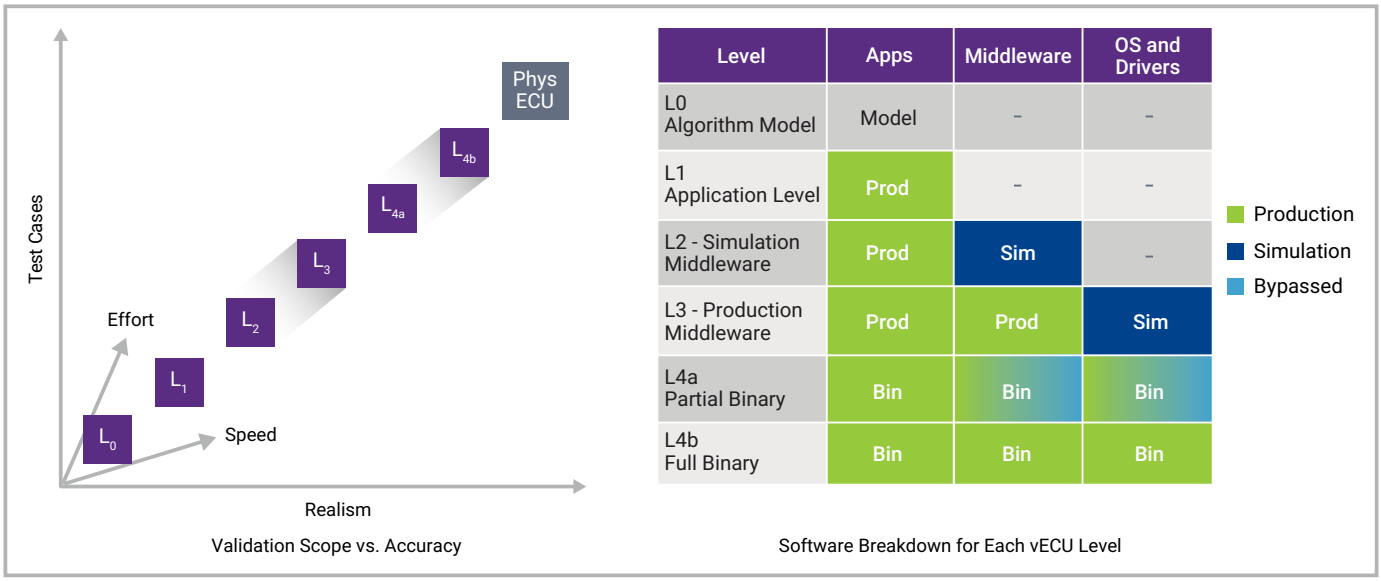

Figure 6: vECU types and abstraction levels

The right side of Figure 6 summarizes how the different levels support the three main software layers of Applications, Middleware, and OS (including drivers). The downwards staircase function shows the growing inclusion of more production software between Level 1 to 4. The columns show how the layers move from substituted simulation layer to production software to binary code.

Synopsys vECU Solutions

As part of providing the industry's broadest and most robust automotive solution, Synopsys fully supports the complete range of vECU levels discussed in this white paper. Figure 7 shows the scope of vECUs supported by Synopsys. This solution offers rich integration with the rest of the automotive ecosystem, by providing OS and semiconductor models, and integrating with environment, sensor, and vehicle dynamics simulators, test automation tools, and many more.

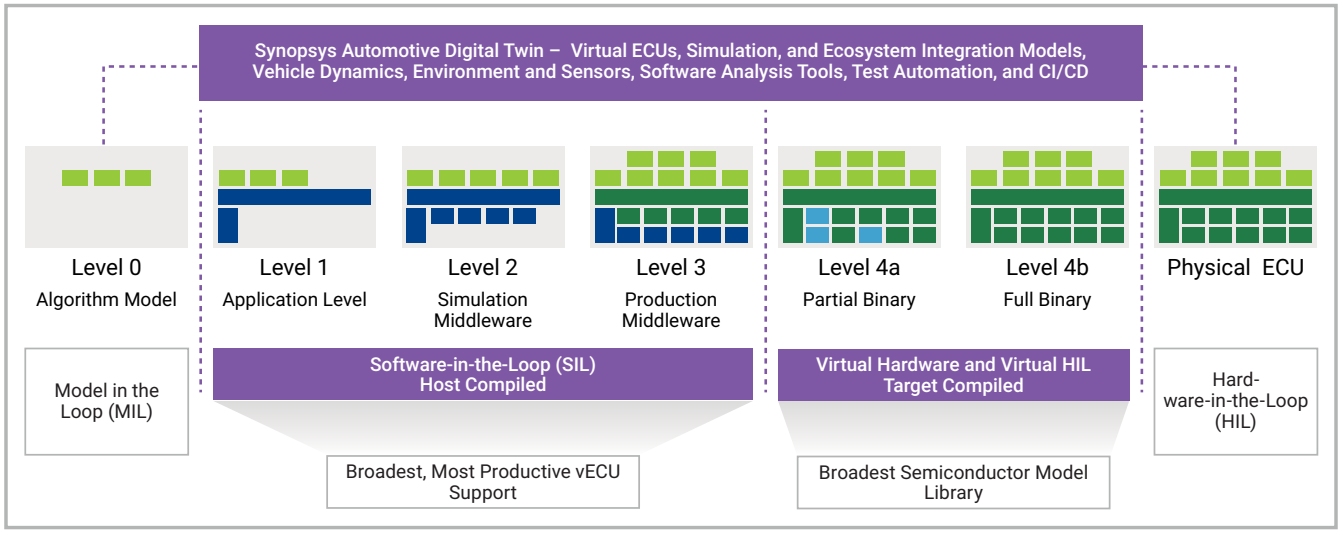

Figure 7: Broadest digital twin/vECU solution for automotive companies

Conclusion

There are many new challenges associated with the development and validation of automotive electronics, and of ECU software in particular. The old method of developing software and validating the system in physical bench setups is obsolete. The introduction and use of virtualization-based development and testing with vECUs is a new approach gaining rapid adoption.

This white paper has introduced a comprehensive vECU classification that covers both POSIX and non-POSIX software stacks. This includes a new type of vECU, Level 4a, which is target compiled level but allows selective bypass of specific target software functions or layers, executing them as host functions instead. This level offers opportunities for vECU simulations difficult to achieve otherwise.

There is no single vECU level that addresses all the challenges. Synopsys supports all the levels in single framework. This allows users to tune the abstraction levels to their use cases, mix vECU levels in a single simulation, and scale up to complete vehicle level simulations. There is no better way to develop functionally correct, robust, safe, and secure automotive software.

04/07/23.CS1078157768-SDG-vECU-2023-SL2031523-WP.

Pub: March 2023