2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 14-15, 2025

Functional Twin: A Framework for ReusableVirtual Electronic Control Units

Sacha Loitz, Torsten Hermann, Martin Hruschka, AUMOVIO SE, Frankfurt a.M., Germany
(sacha.loitz@aumovio.com , torsten.hermann@aumovio.com, martin.3.hruschka@aumovio.com)

***Abstract*— Virtualization is a key enabling technology to develop applications in the context of a Software-defined Vehicle (SDV). Closed loop applications, e.g.: trunk lift, rely on sensor information (Inputs) and actuator controls (Outputs) which are provided by real-time systems like Classic AUTOSAR [1]. The development of real time applications for SDV require an early development and validation environment which also provides the target related Basic SW (BSW) including the connection to a virtual environment. Such a virtual environment allows a hardware independent SW development as well as making the development process more efficient and streamlined.**

The development of a digital twin with the focus on embedded SW testing is quite costly. Due to these costs today's virtualization setups for virtual ECU testing and (sub-) system testing of connected ECUs often start with an L3 virtualization level. And here it again often starts with pure networking test based on the typical communication technologies like CAN, ethernet, etc. In such a pure networking virtual ECU setup the functionality is abstracted away, and the virtual ECUs are degraded towards traffic generators or consumers on the communication network. While this is helpful to validate the overall cars electrical/electronic (EE) architecture, the correct functionality of an application / system cannot be validated without a complete model of the ECU.

In this paper we show our approach to stepwise setup a digital twin for an L3 virtualization which provides an embedded SW function related virtual environment with respect to virtual peripherals and wiring up to the virtual connector. The innovative aspect of our approach is that we setup the interfaces of the ECU component models to be developed in such a way that the same ECU component model can be reused in an L3 as well as in an L4 virtualization system thus reducing the need to recreate these models when moving to lower abstraction levels. Together with wrappers tailored towards different L3 / L4 simulation tools we enable a high reusability and by that enable a higher return on the initial invest to create such a virtual ECU. A key aspect for this is the formal description of the wiring that can be automatically processed and allows for an automatic creation of a virtual ECU tailored towards a specific simulation tool.

In the result section we demonstrate how we use our approach to create an L3 and an L4 virtual ECU from the same source and give an indication on the accuracy and the real time factor we achieve.

Keywords*— *Digital Twin, Virtualization Levels, Electronic Control Unit, Software-defined-Vehicle, Classic AUTOSAR

I. MOTIVATION

The new Electronic and Electrical (E/E) architecture supporting the Software Defined Vehicle concept features a centralized design built on high-performance processors for computing centralized vehicle control functions. Despite this centralization, access to sensors and actuators remains essential, typically managed through Classic AUTOSAR services. The emerging E/E architecture integrates High Performance Computers (HPC) connected via high-speed Ethernet links to Zone Control Units (ZCU). While Adaptive AUTOSAR offers flexibility with services tailored for Linux and Hypervisor-based HPCs, Classic AUTOSAR continues to be relevant for local control and I/O interfaces, ensuring fast and secure vehicle control.

While virtualization of hardware resource and network is a state-of-the-art technology for server applications, it is still evolving for setting up vehicle “Digital Twins” with HPC computers connected to their environment, facilitating system development and early validation. These solutions include Ethernet network virtualization, enhancing connectivity between HPCs and ZCUs. Various commercial solutions [2, 3, 4] enable the abstraction of basic software through the Classic AUTOSAR Microcontroller Abstraction Layer (MCAL). Typically, these solutions virtualize the OS and communication services to simulate a ZCU at a functional level. However, MCAL abstraction is often 1) vendor-specific, 2) requires specific development for I/O control, and 3) is not suited for more accurate simulators. Additionally, OEM requirements and use case driven testing often require the setup of different virtualization solutions compatible with different vendors. Combined with the number of peripherals to be supported individual solutions for each setup become overwhelming [5].

This is why AUMOVIO implemented a Proof of Concept (PoC) to enable a configurable single virtualized concept for different virtualization abstraction levels, including I/O control diversity for Classic AUTOSAR real-time systems. This setup provides a flexible simulation environment, incorporating ECU-level peripheral simulation, various virtualization solutions, and multiple abstraction levels. In this paper, we explain the virtualization concept using an extract from I/O based ECU, comparable to a Controller Platform as a ZCU instance and demonstrate the performance test use case requirements.

This article is structured as follows: Section II provides an overview for the system set-up and the different virtualization levels (L0 to L4). Section III details signal flows and virtualization, by describing signal flow at ECU level of the system setup and how this signal flow is modelled at different levels of abstraction. In detail the flow is described for a L3 and L4 signal flow virtualization level. Based on these signal flows we outline the gaps in existing commercial solutions and the needs for implementing virtual peripherals. Section IV presents the virtual peripheral concept setup, with the tool setup, the virtual peripheral architecture, and finally the PoC implementation. To wrap up, section V summarizes the conclusion and suggests directions for future works.

II. STATE OF THE ART

A. *System set-up*

The system setup for the demonstration of the proposed methodology performed as a PoC is based on an embedded real-time ECU for automotive applications that typically sense and steer parts of the vehicle. Our example is the control of the trunk of a car, where the ECU controls a motor which finally moved the trunk lid as depicted below in Figure 1.

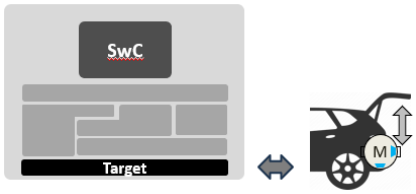

Figure 1 System setup

The ECU is built on the top of Classical AUTOSAR (C-AR) architecture, and the trunk lid is controlled by a hardware driver piloting the trunk lid motor. Three sensors are integrated, one for measuring the current of the motor and a second and third as information when the trunk is opened / closed. The communication with the hardware driver is performed with a serial communication for its configuration. The environment for the experiments is based on commercial tools, one for L3 virtualization and another one for L4 virtualization.

B. *Virtualization Levels*

The state-of-the-art industry wide smart System Engineering Prostep IVIP association agreed on different levels for virtualization of Electronic Control Units (V-ECU) applicable for interoperability of simulation standards. The white paper [6] defines V-ECU simulation levels applicable to AUTOSAR systems that goes from Level 0 aka controller model at algorithm level to Level 4 aka target binary to simulate (close to real-time behavior) and verify/pre-validate the target binary software with representativity of hardware signals.

In between Level 1 contains software code application executed in communication simulation level (typically RTE for AUTOSAR) to perform integration of application software component for functional verification at communication level. The Level 2 simulation BSW provides an environment simulating the BSW services with no software driver dependencies to complete functional software verification with abstracted basic software services (typical communication or memory abstracted level). The Level 3 Production BSW integrates the Hardware Abstraction Layer with simulation of low-level software drivers. Hardware drivers are simulated (e.g. MCAL for AUTOSAR) and application can be verified including BSW features (still with Micro Controller hardware abstracted). Finally, Level 4 only emulates the hardware. The unmodified and target compiled BSW can directly execute on the L4 virtual platform. See Figure 2 for graphical explanation of the levels.

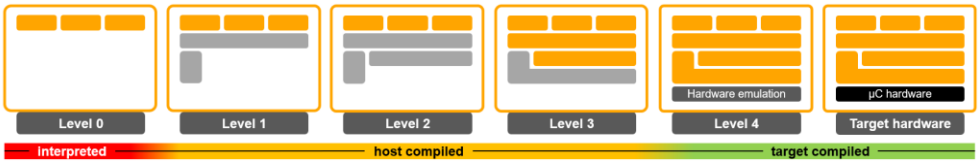

Figure 2 Virtualization levels

The proposed methodology aims to focus on Level 3 and Level 4 V-ECU virtualization levels. The paper proposes to design a methodology capable to abstract hardware signal level and propose a configurable implementation capable to support both level of simulation.

III. SIGNAL FLOW AND VIRTUALIZATION

A. Signal Flow in an ECU

The driving of the motor is implemented with a HW brick called ‘power output’. This power output gets signals / commands from the ECU and drives the power for the motor accordingly. After the motor is enabled, it consumes some current which is read back by the ECU. With the evaluation of the current usage a sticking trunk lid can be detected, and reactions can be derived.

The implementation of the motor and trunk lid simulation, aka. plant model will receive the provided power and provide the actual current. If such a plant model shall be reused from a HW in the Loop (HiL) setup in a virtual environment, the virtualization must also provide the information of the power and read the used current. Figure 3 shows the system setup. The power output peripheral is soldered on the PCB and thus also called “PCB soldered peripheral”. These kinds of peripherals are described here; they get commands from the ECU, drive a line to an external device and read back the used current.

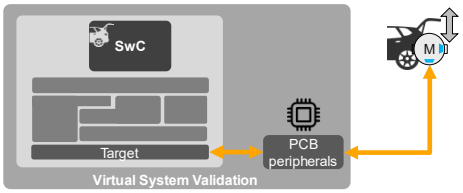

Figure 3 System setup of an ECU with a connected plant model

In the example the trunk lid control application SW component controls the trunk lid plant model. To open the trunk by driving the move of the motor with 50% speed a trigger is generated on interface (1) as shown in Figure 4. The power output driver, located in C-AR Basic Software (BSW) as Complex Device Driver (CDD), converts this functional request at interface (2) with a Pulse-Width Modulation (PWM) control, driven by a hardware driver part of the Micro Controller Abstraction Layer (MCAL). This module configures the Micro Controller (µC) PWM peripheral to toggle a digital output pin of the µC at the requested frequency – see interface (3).

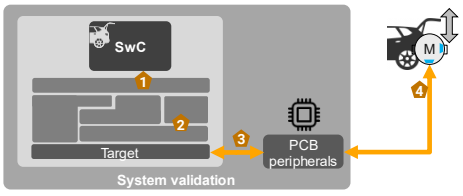

Figure 4 ECU based system with main interface references

This µC output pin acting on related voltage level, 0…3.3V or 0…5V, controls the power output input pin of the peripheral. The power output peripheral switches the 12 V motor command at the requested frequency. If the SWC requests 50% of the maximal speed, the external line is switched for 50% to on and 50% to off. This leads to a theoretical power output of 6V and the motor runs with only 50% of the maximal speed.

While the motor is running its current consumption is measured. The measurement is implemented in or next to the peripheral. This circuit setup converts the consumed current (in Ampere) into a voltage value image of the current. The conversion is performed several times during trunk movement to provide e.g. anti-pinch protection.

The additional sensors report the opened / closed position of the trunk lid. This information is digitalized by the µC ADC and routed to the application SW component to switch off the motor.

B. Signal flow in a L3 & L4 Virtualization

L3 and L4 virtualization is used for functional and integration testing with a wide range of test capabilities from basic communication and diagnosis tests up to closed loop testing. The test cases must be executed with minimum impacts on the virtualized solution and the embedded SW must not be adapted to virtualization needs – except mentioned differences in virtualization levels. To enable closed loop tests in a L3 and L4 virtualization the signal flow as on a real hardware must be respected. Otherwise changes in embedded SW are mandatory and lead to the need of an implementation of virtual peripherals for L3 and L4 virtualization.

IV. REQUIREMENTS ON VIRTUAL PERIPHERAL IMPLEMENTATION

A. Virtual peripheral needs

Virtual peripherals are the connector between the physical interface of the ECU and the µController in- and outputs. There are different peripheral types used in real time projects. A categorization can be done as follows:

- Less complex like (de-)multiplexer or simple logical AND/OR gates, Inverter
- Medium complex, like H-Bridges, High Side Drivers (HSD), Low Side Drivers (LSD), eFuses.
- Highly complex, like ethernet switches RF antenna, Bluetooth, …

For each of the peripheral categories different hardware features must be managed like the communication between µController and peripheral (DIO, SPI, …) and the supported voltages and currents. A mid-range ECU project embeds around 60 peripherals and around 20 different categories and kinds.

To be able to efficiently handle these peripherals while achieving a good simulation performance the impact on the embedded SW needs to be analyzed for each peripheral with the functional test objective in mind. Most ECU soldered peripherals have no direct impact on the embedded SW and can thus be ignored. Other components offer registers that are directly accessed by the SW but are not connected to the plant model (e.g. temperature sensor or PMIC) and can be represented by a simple register stub. Most interesting in the context of this presentation are the remaining peripherals that require a functional model. Especially for these the functional requirements of the model need to be analyzed e.g. the communication interface to the MCU is for both L3 and L4 level models often established for performance reason not at a pin level but at an API level and e.g. missing retry mechanisms. Still the effort to implement such peripherals is high and it cannot be afforded to reimplement these peripherals for different abstraction levels or simulation tool vendors. The next section presents our approach to cover several tools at both L3 and L4 level with the same peripheral model. Please note that for these peripherals the functional requirements to the model are besides the communication interface to the MCU model / MCAL model equivalent for both L3 and L4 level simulations thus that usually the L4 simulation benefits from the higher speed of reusing an L3 model.

B. Comparison of virtual peripheral implementation for L3 and L4

When modeling ECU (Electronic Control Unit) peripherals in the context of ProSTEP IVIP simulation levels, the level of detail and realism in how these peripherals behave and interact with the rest of the system varies significantly between Level 3 (L3) and Level 4 (L4). The following sections primarily focus on the software intended to run on the virtual ECU and the interaction between microcontroller and external peripherals on the PCB. We do not focus on the actual implementation of the virtual microcontroller and do thus not compare the internal implementation for e.g. registers, on-chip interconnect, caches, etc.

In a Level 3 simulator, ECU peripherals are typically functionally modeled without incorporating timing behavior. This means the simulation focuses on the logical correctness of the software interacting with the peripherals but does not account for how long operations take or how they are scheduled in real time. The models

are abstract and simplified, which allows for faster simulation and early-stage software testing but limits the ability to detect timing-related issues.

In contrast, a Level 4 simulator introduces a more detailed and loosely time-accurate representation of ECU peripherals. While it still does not simulate exact hardware timing, it includes approximate timing behavior to better reflect real-world execution. This is often achieved using Transaction-Level Modeling (TLM), which abstracts communication between components at a higher level than signal-level simulation, enabling faster execution while still capturing essential timing characteristics. This allows for more realistic integration testing, including the detection of potential race conditions or scheduling conflicts. A comparison between the characteristics of L3 and L4 is captured in Table I.

Table I. Comparison of L3 and L4

Aspect	Level 3 (L3)	Level 4 (L4)
Simulation Fidelity	High fidelity; close to real ECU behavior, excluding hardware-dependent drivers	Very high fidelity; includes hardware abstraction and near-complete ECU behavior
Software Integration	Includes all basic software above hardware drivers	Includes full software stack including hardware abstraction layers. Binary compatible.
Timing Behavior Simulation	Capable of simulating timing with real-time OS integration	Simulates precise timing and scheduling, suitable for virtual hardware-in-the-loop (vHiL)
Use Cases	Software-in-the-loop (SiL), early integration testing	vHiL, final validation, safety-critical testing
HW Dependency	Minimal; excludes hardware-specific drivers	Abstracted; simulates hardware interfaces
Complexity	Moderate to high	Very high
Toolchain Requirements	Requires real-time OS and simulation environment	Requires full simulation stack and possibly co-simulation tools
Validation Scope	Functional and timing validation	Full system validation including fault injection and robustness testing

In a typical ECU development flow, both Level 3 (L3) and Level 4 (L4) simulators are essential because they serve complementary purposes at different stages of the development and validation process. L3 simulators are used early in the development cycle for functional testing and software integration, offering fast simulation speeds and enabling developers to validate logic without being constrained by hardware or timing. L4 simulators, on the other hand, are used later for system-level validation, where timing behavior, hardware abstraction, and integration with real-time systems become critical. As L4 simulators are still an abstract representation especially of the timing the final system-level validation always needs to be done hand-in-hand with real hardware. The binary compatibility here allows a transition between L4 simulator and real hardware.

However, these simulators are rarely reusable across levels due to several key challenges:

- Tool Incompatibilities: L3 and L4 simulators are often developed using different toolchains or simulation environments, which are optimized for different abstraction levels. This makes direct reuse technically difficult or even impossible without significant adaptation.
- Lack of Standardization: There is no universally adopted standard for model interfaces or timing semantics across simulation levels, leads to inconsistencies in how models are structured and executed.
- Different Modeling Objectives: L3 simulators prioritize speed and functional correctness, while L4 simulators emphasize timing accuracy and hardware interaction. As a result, the internal architecture and assumptions of the models differ significantly.
- Performance Trade-offs: L4 simulators often use Transaction-Level Modeling (TLM) and include loosely timed behavior, which adds complexity and overhead not needed in L3. Reusing an L4 simulator in an L3 context would unnecessarily slow down simulation, while using an L3 simulator in an L4 context would lack the required fidelity.

Because of these differences, development teams typically maintain separate simulator versions for L3 and L4, each tailored to its specific role in the development pipeline [7, 8, 9]. Adding different tool suppliers at each level that all need to be supported due to different customer requests we get a huge zoo of different flavors that need to be supported by the different peripheral models that are part of our simulators. Opening the question on how far we can combine these different models.

C. *Unique implementation of virtual peripherals for L3 and L4*

While L3 and L4 simulators are often developed independently due to toolchain and fidelity differences, reuse of individual L3 peripheral models at L4 is not categorically impossible — it largely depends on the required fidelity and simulation objectives. In scenarios where the goal is to execute unmodified software on an L4 virtual ECU without validating full ECU functionality, a compromise on fidelity is acceptable. This is particularly relevant for control-oriented software, such as our example of the trunk window lifter, where the interaction with PCB peripherals is the primary concern. Since this interaction is central to the software's behavior, models of these peripherals are required at both L3 and L4 simulator levels.

The main distinction between L3 and L4 simulation flows lies in the presence or absence of timing annotations. While both levels can simulate the same software logic, their treatment of timing and peripheral interaction differs significantly. The impact of the peripheral timing on the actual simulation depends on who is the initiator of a transaction. Table 2 shows a comparison for the L3 and L4 features for transactions initiated by the MCU. While Table 3 presents the behavior if a peripheral is the source of a transaction.

Table 2 Transactions Initiated by Software (MCU → Peripheral)

Aspect	Level 3 (L3)	Level 4 (L4)
Timing	No timing information is modeled.	TLM allows attaching timestamps to transactions, enabling loosely timed behavior.
Peripheral Behavior	Assumes static, passive behavior. Immediate response expected.	Also assumes static behavior but allows modeling of response delays.
Use Case Fit	Sufficient for validating control logic (e.g., trunk/window lifter).	Adds realism for integration testing, but not strictly necessary if timing is not critical.

Insight: Since PCB peripherals (e.g., switches, sensors) are typically passive and their behavior doesn't change dynamically, L3 is often sufficient for validating software logic. L4 adds timing realism but may not be essential unless integration with other timed systems is required.

Table 3 Transactions Initiated by Peripherals (Peripheral → MCU)

Aspect	Level 3 (L3)	Level 4 (L4)
Peripheral Activity	Peripherals cannot actively initiate transactions. No interrupts.	Peripherals can initiate transactions and raise interrupts.
Event Handling	External events (e.g., from plant model) are polled or triggered via callbacks.	Interrupts can be raised and handled by the simulated MCU core.
Use Case Fit	Acceptable for most control SW, as polling is the dominant pattern.	Necessary for testing interrupt-driven behavior or fault injection.

Insight: In most real-world ECU applications, polling is preferred over interrupts for safety and determinism. Interrupts are mainly used for fault management, which in simulation is often script-controlled and directly injected into the MCU model – making L3 sufficient in many cases.

D. *Known limitations of the selected approach*

While the selected approach – reusing L3 models in L4 contexts with relaxed fidelity – offers efficiency and practicality, it has clear limitations:

- Timing-Critical Applications: The approach is not suitable for systems with tight real-time constraints, where precise timing and synchronization between components are essential.
- Example ESC Control: In an Electronic Stability Control (ESC) system, the microcontroller must:
  - Measure vehicle motion via an inertial sensor,
  - Apply brake pressure,
  - Monitor wheel speed and inertial feedback,
  - Release brake pressure accordingly.

This closed-loop control involves multiple tightly coupled components and real-time feedback, making it infeasible to simulate accurately without precise timing and interrupt handling, thus out of scope for the current abstraction level.

V. EVALUATION OF THE PRESENTED APPROACH

To show the proof of the concept AUMOVIO has implemented prototypes in commercial simulation tools for Level 3 and Level 4. Each prototype uses the same virtual peripheral C++ implementation for the eFuse VNF9D5F. The embedded SW was, except for the different MCAL modules, almost identical. In Level 3 simulation a multiplexer was additionally integrated. Figure 5 explains the setup of the virtual PCB with the wiring from the MCAL over the virtual peripheral up to the virtual connector where measurement- and test-tool was connected. Figure 6 shows the almost identical setup for L4 but with the "Virtual Adapter" to embed the virtual peripheral to the L4 environment.

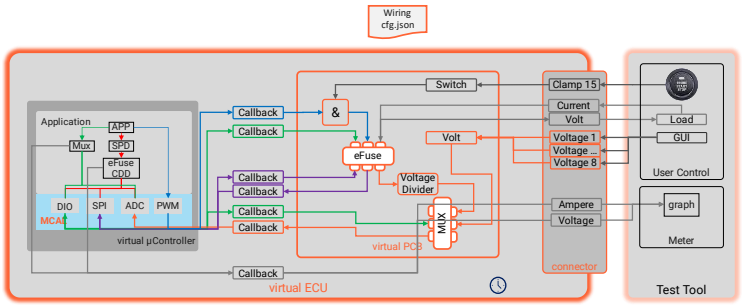

Figure 5 Setup Level 3 simulation environment

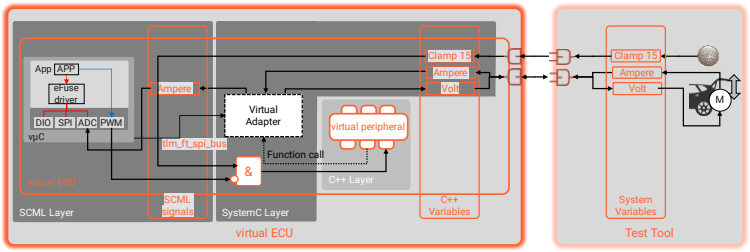

Figure 6 Setup Level 4 simulation environment

The SPI initialization sequence of the eFuse, shown in Figure 7, was measured at the L3 setup. Here it is shown, that by the eFuse internal state change to "Normal", triggered by SPI commands, the Outputs are disabled and must be explicitly enabled via SPI. The PWM ramp-up and Voltage changes on the eFuse outputs in Figure 8 have been taken from the L4 setup. Channel 0 is disabled, and Channel 1 provides a Voltage of 2.5V

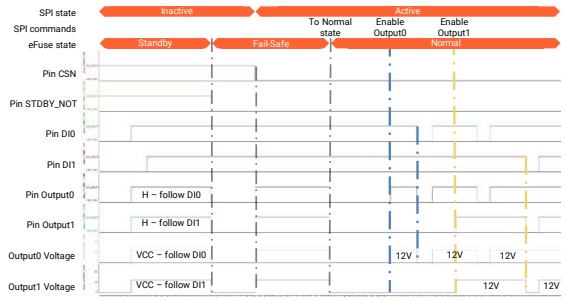

Figure 7 SPI based eFuse initialization sequence

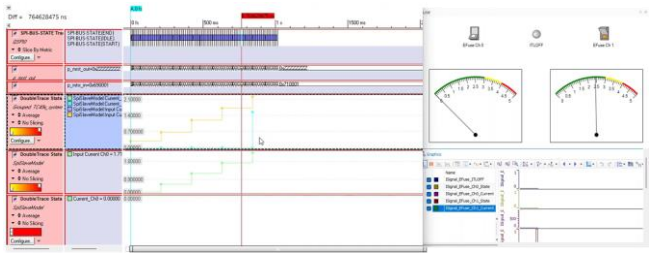

Figure 8 PWM ramp-up and voltage change on the eFuse outputs

VI. RESULT / CONCLUSION

A. *Result*

While L4 offers more accurate timing and interrupt handling, L3 can often be reused in L4 contexts if the simulation goals are limited to software execution and not full ECU validation. The key enabler is a clear understanding of the required fidelity: if timing and peripheral-initiated events are not critical, L3 models can be adapted or reused with minimal compromise. To be able to reuse our L3 peripherals we defined an API according to which our peripherals are developed. This API focuses on the commonalities of virtual peripherals: PINs. Every kind of virtual peripheral has a set of PINs which can be connected or not. In case they are connected, they transfer a certain data type (current, voltage, DIO, PWM, ...). In addition, each PIN has a defined direction (input or output) which allows safe and consistent wiring. This generic API is then wrapped in a SystemC wrapper interfacing to SystemC signals and TLM transactions.

According to this API we implemented an eFuse model for use in an L3 simulation. To reuse this model in an L4 simulation we only had to implement a wrapper that is due to our API generic for different peripherals and add the according wiring which is here the TLM transaction from the microcontroller to the peripheral. We tested this setup first with dummy initiator in an IEEE1666 SystemC environment. In a later stage the TLM was converted into the TLM flavor used by a proprietary tool supplier. In this setup we were able to successfully run a SW example switching the eFuse and reading back the resulting voltages. With this we have shown that we only need to adjust the connection (aka wiring) between the respective microcontroller model and the peripheral and don't need to touch the peripheral implementation.

B. *Conclusion*

With the ability to implement virtual ECU (vECU) peripherals for different virtualization levels we developed a tooling to connect the virtual ECU peripheral implementation to different virtualization solutions. By using this tooling, the virtualization tools which – out of the box – mainly support a virtual Microcontroller (vµC) with communication interfaces like CAN, Ethernet, LIN, etc. can be exchanged. The wiring between the vµC and the vECU peripherals are setup outside the tool solution and can been taken over.

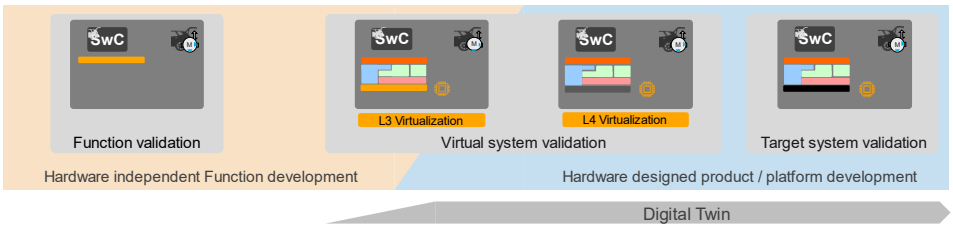

Figure 9 Streamlined development with multiple virtualization level support

This means that the setup of a virtual ECU which was developed in one virtualization solution (e.g. L3) the configuration can be taken over to generate a vECU in a L4 virtualization solution.

Figure 9 shows the approach of the streamlined development with virtualization support. For the application development (not in focus of this paper) the L1 virtualization is used. The outcome of this step, the embedded SW component (SWC) and the test cases / plant model / environment simulation, can be reused in the L3 virtualization to perform functional integration tests. With the re-use of the virtualized ECU from L3 to L4 more detailed tests like embedded SW multicore and safety tests can run in a virtual environment. The final validation on a real target probably including real external peripheral (e.g. motor, ..) setup is still mandatory but in a reduced amount of HiL.

With the general capabilities of virtualization like multi-instances, easier SW tracing, error injections, the overall setup enables a streamlined and cost-efficient way for shorter time to market development approach.

VII. REFERENCES

[1] AUTOSAR GbR, "AUTOSAR Classic https://www.autosar.org/standards/classic-platform.

[2] Synopsys Silver tool solution, https://www.synopsys.com/verification/virtual-prototyping/silver.html

[3] ASTC, VLAB Works, https://vlabworks.com/

[4] Vector, vVirtualTarget tool solution, https://www.vector.com/at/en/products/products-a-z/software/vvirtualtarget

[5] T. Hermann, P. Cuenot, S. Loitz, "Digital Twin for Classic AUTOSAR ECU," CESA 2025, Versaille, France, 2025.

[6] prostep ivip association, "Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs), White Paper, 2020, https://www.prostep.org/en/medialibrary/translate-to-english-detail?ai%5Baction%5D=detail&ai%5Bcontroller%5D=Catalog&ai%5Bd_name%5D=wp_smartse_vecu&ai%5Bd_pos%5D=.

[7] Ruehl, M., Bronner, F, "Creation and Usage of Virtual Control Units (V-ECUs) in SIL and HIL for Development and Validation especially for Software-Defined-Vehicles (SDVs)." In: Kulzer, A.C., Reuss, HC., Wagner, A. (eds) 2024 Stuttgart International Symposium on Automotive and Engine Technology. I{SSYM 2024.

[8] dSPACE GmbH, "(R)evolution in the exchange of V-ECUs," *dSPACE Engineers' Insights*, Paderborn, Germany. [Online]. Available: https://www.dspace.com/en/pub/home/news/engineers-insights/v-ecu-standard.cfm

[9] dSPACE GmbH, "Combining Level 3 and Level 4 virtual ECUs in one simulation," *dSPACE Learning Bits*, Paderborn, Germany. [Online Video]. Available: https://www.dspace.com/en/inc/home/learning-center/recordings/learning-bits/video_lb_l34virtualecus.cfm