Ref. Ares(2026)3561246 - 07/04/2026

Hardware Abstraction Layer for a European Software Defined Vehicle Approach

D3.4 – Modelling and simulation activities for the HAL4SDV tools

30.03.2026

Co-funded by the European Union
Chips JU

Deliverable	D3.4 – Modelling and simulation activities for the HAL4SDV tools
Work Package(s)	WP3
Task(s)	Task 3.4
Dissemination Level	Public
Due Date	31.03.2026
Actual Submission Date	30.03.2026
WP Leader	Dirk Slama
Task Leader	Giuseppe Tagliavini
Deliverable Leader	Giuseppe Tagliavini
Contact Person	Giuseppe Tagliavini
E-mail	giuseppe.tagliavini@unibo.it

Document History		
Revision	Date	Description
V0.1	12.12.2025	Initial version
V0.2	28.01.2026	First complete draft
V0.3	31.01.2026	Ship out for internal revision
V0.4	23.02.2026	Version ready for PSB review
V0.5	16.03.2026	Version for PGA review
V1.0	30.03.2026	Final version

Authors	
Name	Partner Short Name
Giuseppe Tagliavini	UNIBO
Daniele Jahier Pagliari	POLITO
Vittorio Zaccaria	POLIMI
Nenad Petrovic	TUM
Max Scheerer	FZI
Sebastian Weber	FZI
Florian Pölzlbauer, Mario Driussi	VIF

Co-funded by the European Union

Chips JU

This document and the information contained within may not be copied, used or disclosed, entirely or partially, outside of the HAL4SDV consortium without prior permission of the project partners in written form.

Acknowledgement

The project is co-funded by the Chips Joint Undertaking (Chips JU) and National Authorities under grant agreement n° 101139789.

Disclaimer

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or National Authorities. Neither the European Union nor the granting authorities can be held responsible for them.

Table of Contents

1 Summary 8

2 Modelling and simulation activities for the RISC-V integration 10

2.1 GVSoC Simulation Modules 10
2.1.1 Contributing Partners 10
2.1.2 Category 10
2.1.3 Description and Achievements 10
2.1.4 Current Release 12
2.1.5 Interactions with other tools 13

2.2 Analysis of performance and safety properties during design time 13
2.2.1 Contributing Partners 13
2.2.2 Category 13
2.2.3 Description and Achievements 13
2.2.4 Current Release 15
2.2.5 Interactions with other tools 16

2.3 Model-based Representations Tools and QA Plugin 16
2.3.1 Contributing Partners 16
2.3.2 Category 16
2.3.3 Description and Achievements 16
2.3.4 Current Release 23
2.3.5 Interactions with other tools 23

2.4 Hypervisor support 23
2.4.1 Contributing Partners 23
2.4.2 Category 23
2.4.3 Description and Achievements 23
2.4.4 Current Release 25
2.4.5 Interactions with other tools 25

2.5 Middleware for Performance (DDS, Rust) 25
2.5.1 Contributing Partners 25
2.5.2 Category 25
2.5.3 Description and Achievements 25
2.5.4 Current Release 26
2.5.5 Interactions with other tools 26

2.6 VPSim Support 26
2.6.1 Contributing Partners 26
2.6.2 Category 26
2.6.3 Description and Achievements 26
2.6.4 Current Release 27
2.6.5 Interactions with other tools 28

2.7 Build- and Deployment-Pipeline for Xen-based Systems 28
2.7.1 Contributing Partners 28
2.7.2 Category 28
2.7.3 Description and Achievements 28
2.7.4 Current Release 29
2.7.5 Interactions with other tools 29

3 Conclusion 30

4 References 31

Co-funded by the European Union

List of figures

Figure 1: Overview of the HAL4SDV modelling and simulation activities (T3.2 and T3.4) 8

Figure 2: Modelling and simulation activities related to the modelling and simulation tools (T3.4) 9

Figure 3: GVSoC timing and events. 11

Figure 4: GVSoC architecture..... 12

Figure 5: Multi-level simulation approach..... 15

Figure 6: GenAI-driven test generation workflow leveraging VSS specification and Gherkin test cases. 16

Figure 7: Regulatory compliance ADAS test scenario modelling using LLMs. 20

Figure 8: Workflow for LLM-driven SDV behaviour modelling relying on formal methods. .. 21

Figure 9: VPSim platform overview. 27

Figure 10: Build- and deployment-pipeline for Xen-based systems. 29

List of tables

Table 1: VSS Signal Mapping Results for LLMs in case of CPDS 18

Table 2: VLM Performance on Diagram-Based VSS Signal Extraction (CPDS)..... 19

Table 3: Integrated Results - Prompting techniques across pipelines for regulatory simulation scenarios..... 21

Abbreviations

ADAS	Advanced Driver Assistance Systems
AI	Artificial Intelligence
API	Application Programming Interface
ARM	Advanced RISC Machines (company)
BB	Building Block
CARLA	CAR Learning to Act (autonomous driving simulator)
CE	Conformité Européenne
CPU	Central Processing Unit
CPDS	Child Presence Detection System
COT	Chain of Thought
DDS	Data Distribution Service
DMA	Direct Memory Access
DoA	Description of Action
DTF	Digital Test Framework
ECU	Electronic Control Unit
E/E	Electrical/Electronic
FMI	Functional Mock-up Interface
FPGA	Field-Programmable Gate Array
GenAI	Generative Artificial Intelligence
GPU	Graphics Processing Unit
HAL	Hardware Abstraction Layer
HPC	High-Performance Computing
HW	Hardware
ICL	In-Context Learning
IPC	Inter-Process Communication
IPI	Inter-Processor Interrupt
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JU	Joint Undertaking
KPI	Key Performance Indicator
LLM	Large Language Model
MCS	Mixed-Criticality System
MCU	Microcontroller Unit
MPU	Memory Protection Unit
NIC	Network Interface Controller
OEM	Original Equipment Manufacturer
PCM	Palladio Component Model
QA	Quality Assurance
QEMU	Quick Emulator
RAG	Retrieval-Augmented Generation
RISC-V	Reduced Instruction Set Computer – Version V (standard)
SDV	Software-Defined Vehicle
SoC	System on Chip
SPMP	Supervisor Physical Memory Protection
SOP	Start of Production

Co-funded by the European Union

Chips JU

SW	Software
TLM	Transaction-Level Modelling
TPR	True Positive Rate
UC	Use Case
UML	Unified Modelling Language
UN	United Nations
VIF	Virtual Interface
VLM	Vision–Language Model
VSS	Vehicle Signal Specification
WP	Work Package
XAPI	Xen API
XRCE-DDS	eXtremely Resource Constrained DDS

Co-funded by the European Union

ChipsJU

1 Summary

This deliverable reports on the activities conducted within Task 3.4 (T3.4), focusing on modelling and simulation tools that support the development, integration, and validation of software-defined vehicle (SDV) platforms. The work presented provides higher-level, cross-platform tools and methodologies for performance analysis, safety assessment, quality assurance, virtualization, and deployment workflows.

The modelling and simulation activities addressed by T3.4 span multiple abstraction layers, ranging from virtual platforms and architectural modelling to AI-assisted validation and system integration tooling. Figure 1 provides a high-level overview of the entire modelling and simulation landscape covered by T3.2 and T3.4, organized across abstraction layers and functional domains. The activities of T3.4 are strongly related to those of T3.2, which focuses on modelling and simulation activities specific to the open-source RISC-V ecosystem. Some tools are greyed out because they are not actively developed inside the project, but higher-level tools use them.

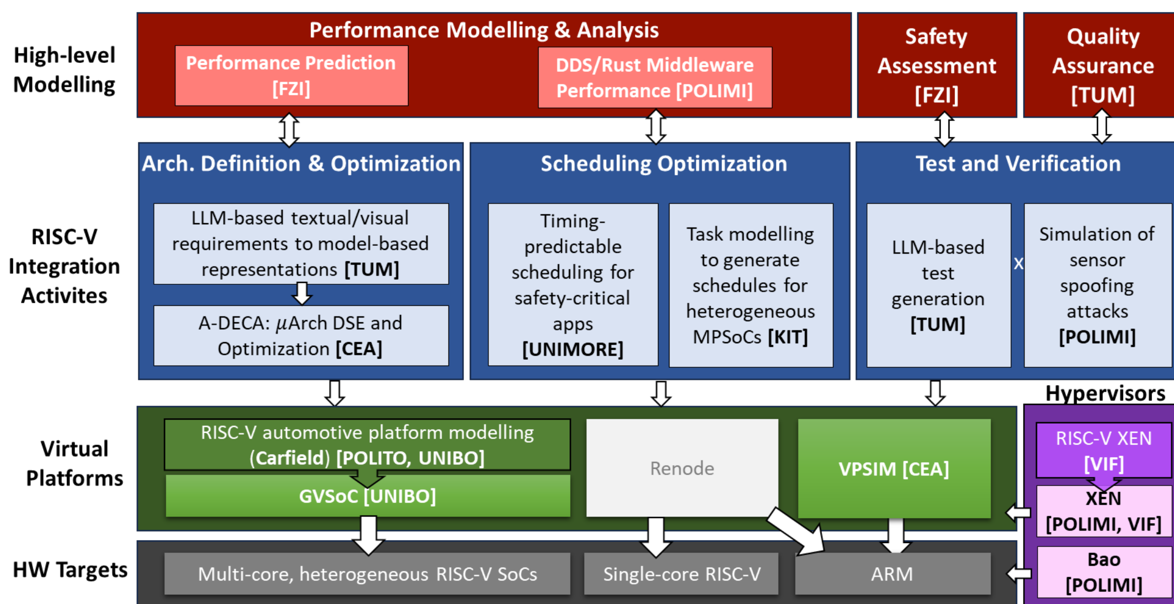

Figure 1: Overview of the HAL4SDV modelling and simulation activities (T3.2 and T3.4).

Co-funded by the European Union

Chips JU

Figure 2 highlights the subset of tools and activities that fall specifically within the scope of T3.4 and are therefore documented in this deliverable.

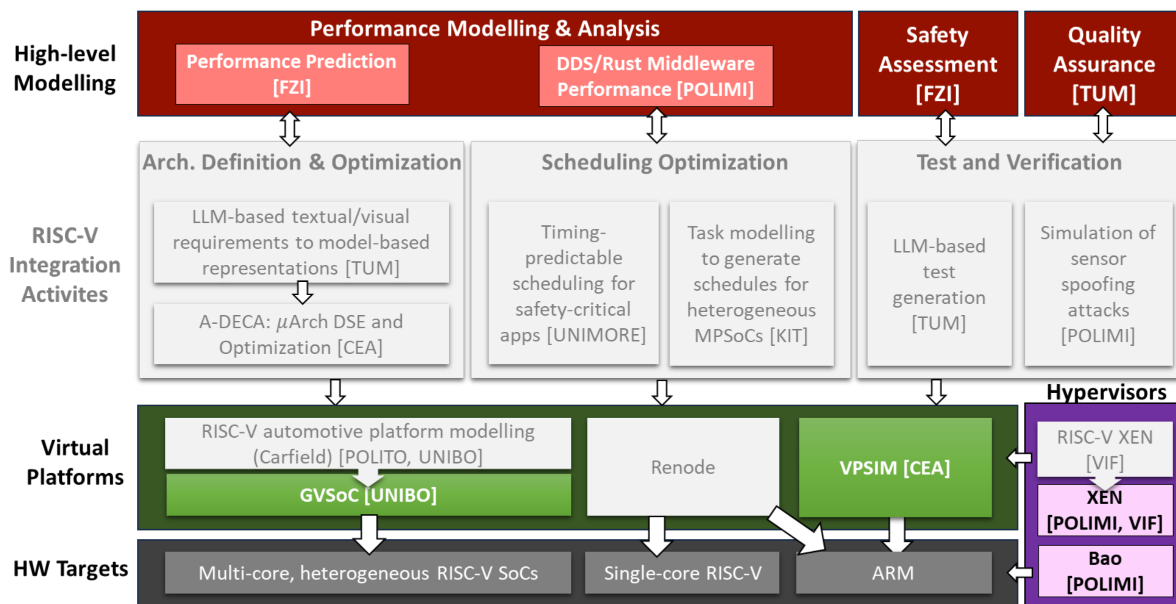

Figure 2: Modelling and simulation activities related to the modelling and simulation tools (T3.4).

A primary focus of the contributions is on **virtual prototyping and platform simulation**. This includes virtual platforms that facilitate early hardware/software co-design and architectural exploration of heterogeneous computing systems. Notably, extensions to GVSoC allow for the execution of realistic software workloads while preserving key timing effects. Additionally, enhancements to VPSim improve benchmarking, profiling, and bare-metal execution capabilities, enabling thorough analysis of real-time constraints and task-level interactions in SDV architectures.

At a higher level of abstraction, this document introduces a **model-based performance analysis methodology for heterogeneous, AI-enabled automotive systems.** Building on the Palladio Component Model and the Slingshot simulator, the proposed approach enables quantitative predictions of both component-level and end-to-end timing properties during the design phase. A multi-level simulation strategy is also presented to balance scalability and accuracy, combining abstract and detailed timing models while ensuring consistency in functional states.

The deliverable further addresses **quality assurance and validation through GenAI-enabled modelling and test generation tools.** These contributions include automated extraction of relevant vehicle signals from natural-language and diagram-based specifications, LLM-driven generation of test artifacts, and automated modelling of regulatory-compliant Advanced Driver-Assistance Systems (ADAS) test scenarios in realistic simulation environments. The document illustrates these workflows through a series of pipelines that integrate retrieval-augmented generation, vision-language models, and formal validation techniques.

Additional efforts focus on **virtualization and middleware support for mixed-criticality SDV platforms.** The deliverable reports on the evaluation of hypervisor-based architectures using Bao and Xen, addressing issues of isolation, performance determinism, and efficient I/O

virtualization across both safety-oriented microcontrollers and high-performance application processors. Complementary middleware contributions emphasize deterministic data distribution and memory-safe system software components to ensure predictable and scalable SDV execution environments.

Finally, the deliverable outlines **a build-and-deployment pipeline for Xen-based systems that enables automated DevOps-style workflows for building, testing, deploying, and monitoring virtualized automotive applications**. This pipeline supports deployment across development, laboratory, and production environments.

Overall, the activities described in this deliverable illustrate how designers can systematically integrate modelling and simulation tools at various abstraction levels to support early design space exploration, performance and safety analysis, automated validation, and scalable deployment of software-defined vehicle architectures. The results of T3.4 provide a foundation for integrated toolchains within HAL4SDV and contribute to the project's objective of enabling robust, efficient, and reusable workflows for SDV development.

2 Modelling and simulation activities for the RISC-V integration

2.1 GVSoC Simulation Modules

2.1.1 Contributing Partners

UNIBO, POLITO

2.1.2 Category

Virtual Platform

2.1.3 Description and Achievements

GVSoC [1] is an event-driven virtual platform that simulates full computing systems, including CPU cores, memory hierarchies, interconnects, and peripherals. It is specifically designed for early-stage design and software development while preserving important timing effects, such as contention, peripheral latency, and I/O behaviour, which are essential for system integration. In practice, GVSoC facilitates hardware and software co-design by allowing developers to boot and run realistic software workloads on a configurable platform model, even before silicon or FPGA prototypes are available.

2.1.3.1 GVSoC timing model

GVSoC utilizes an event-driven simulation model where all time-dependent activities are represented as discrete events linked to a future simulation time. Instead of advancing the simulation one cycle at a time, GVSoC executes the next scheduled event. This method reduces simulation overhead while maintaining the temporal relationships between events. Each event in GVSoC consists of three key elements:

i. a reference to a callback function implementing the component behaviour,

ii. a payload carrying contextual information (e.g., request metadata or internal state),

iii. a time offset that specifies when the event must be executed relative to the local clock domain.

To manage event scheduling efficiently, GVSoC utilizes a two-level queuing mechanism. This mechanism is based on a fixed-size circular buffer and one or more external queues. Each clock domain defines a time window (Tw) that determines the horizon for near-future events. Events scheduled for execution within this window are encoded directly into the circular buffer, using the relative position in time as an index. This data structure allows for constant-time insertion and quick iteration, facilitating the efficient execution of closely spaced events while minimizing scheduling overhead. Figure 3 (a) shows the structure of the circular queue.

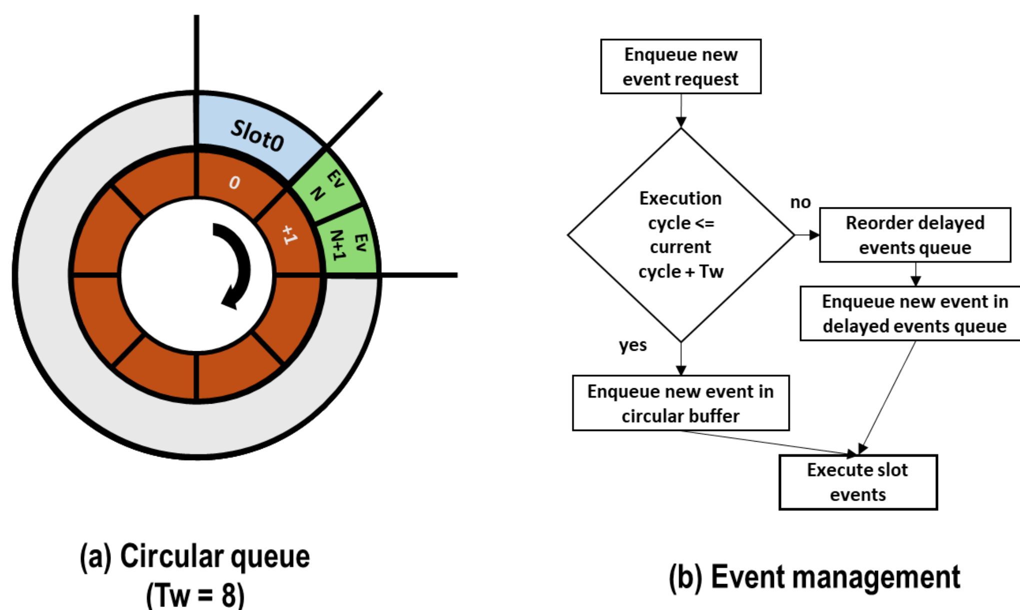

Figure 3: GVSoC timing and events.

Figure 3 (b) illustrates the event management algorithm. When a component schedules an event whose execution time lies beyond the current time window, the simulator places it into an external ordered queue. This data structure organizes events based on their scheduled execution times and serves as a deferred pool for activities involving long latencies, such as direct memory access (DMA) transfers, interactions with peripherals, or cross-domain synchronizations. As simulation time progresses and the circular buffer completes a full rotation, GVSoC checks the external queue and moves the events that fall within the upcoming time window into the circular buffer. This approach allows the simulator to maintain a limited active scheduling structure while still accommodating delays of any length. Consequently, the complexity of event management remains proportional to the number of imminent events rather than the total number of outstanding events in the system.

Events scheduled for the same simulation instant are executed one after the other, without imposing an arbitrary total order unless explicitly required by the dependencies between components. This decision acknowledges that, at the selected level of abstraction, concurrent hardware activities can be considered temporally equivalent if the causal order is maintained. This approach ensures deterministic execution while avoiding unnecessary serialization constraints.

Co-funded by the European Union
ChipsJU

An additional layer of event handling supports multiple clock domains. Each domain operates with its own event queues and clock counters, while a global time engine maintains global synchronization. When a request crosses a domain boundary, a specialized interface converts the event timestamp to match the target frequency and re-encodes it into the appropriate queue. This mechanism allows GVSoC to model heterogeneous platforms, where cores, interconnects, and peripherals operate at different clock rates, without forcing the simulation into a single global cycle.

Overall, all these features enable GVSoC to scale to full-platform simulations while maintaining a favourable trade-off between speed and timing fidelity. This design is a key factor in allowing the simulator to remain generic and extensible, independent of the specific instruction set or platform configuration instantiated on top of it.

2.1.3.2 GVSoC architecture

GVSoC adopts a modular architecture that separates platform structure from component behaviour. A set of Python generators and configuration files includes parameters such as processing elements, memory hierarchy, interconnect properties, clock domains, and peripherals. These configuration artifacts serve as a declarative description of the target platform and can be modified at runtime without recompiling the simulator.

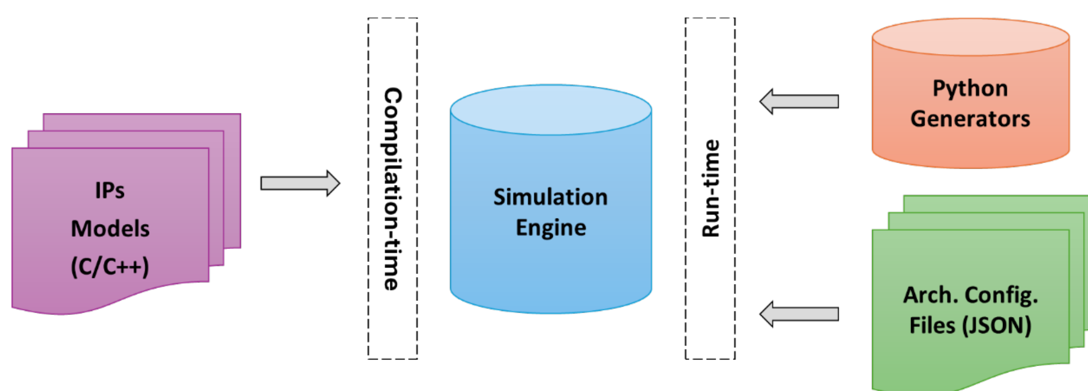

Figure 4: GVSoC architecture.

The functional and timing behaviour of individual components is implemented in C/C++ models, which form a reusable library of simulation blocks. Each component manages its own internal state, timing rules, and interaction interfaces, with clearly defined ports for communication. This design allows the simulator to efficiently manage complex system interactions while maintaining a clear distinction between architectural descriptions and behavioural implementations.

Separating configuration from execution logic allows for quick exploration of various platform instances. GVSoC facilitates efficient design-space exploration and architectural what-if analysis across the entire platform, including processors, memory, and I/O subsystems. This modular structure also allows for easy incremental expansion of the simulator.

2.1.4 Current Release

GVSoC is distributed as an open-source software project and is publicly available through a set of repositories hosted on GitHub: https://github.com/gvsoc/gvsoc

Co-funded by the European Union
ChipsJU

The GVSoC core is released under the **Apache License 2.0**, making it suitable for both academic and industrial use. Additional repositories provide platform descriptions, processor models, and integration tools, forming a comprehensive virtual prototyping environment.

User and developer documentation is available online and details the installation procedures, platform configuration methods, and guidelines for enhancing the simulator with new components or architectural features. Updated links are available in the README.md file on the GitHub repository.

2.1.5 Interactions with other tools

The current version of the simulator features an instruction-set simulator for RISC-V processors, along with models of typical System on Chip (SoC) subsystems. Within the context of the HAL4SDV project, GVSoC has been used as a foundation to define an open-source reference architecture for the RISC-V ecosystem. Further details about this architecture can be found in D3.2.

2.2 Analysis of performance and safety properties during design time

2.2.1 Contributing Partners

FZI

2.2.2 Category

High-level modelling

2.2.3 Description and Achievements

The architectural design of AI-enabled driving functions for software-defined vehicles requires quantitative performance assessment before implementation, despite substantial platform heterogeneity and complex interference effects. In such systems, end-to-end performance cannot be inferred reliably from isolated component runtimes. It is an emergent property of deployment decisions, shared-resource contention, scheduling policies, and communication latencies across distributed computing units. The inclusion of AI inference further increases variability since execution time depends on workload characteristics as well as the selected accelerator and software stack. Simultaneously, the design process demands rapid and repeatable feedback to enable systematic comparison of architectural alternatives. This yields a methodological trade-off: uniformly high-fidelity simulation is typically too costly for iterative exploration, whereas highly abstract prediction models may omit the dominant contributors to latency and performance.

To overcome this trade-off, we develop a model-based simulation tool to support architectural decision-making by predicting performance-related quality attributes prior to implementation. The tool targets heterogeneous AI-enabled systems in the SDV context and focuses on quantitative prediction of timing and performance, explicitly accounting for AI

inference time as part of end-to-end execution chains. Uncertainty-related safety assessment is treated as a complementary probabilistic analysis and detailed in D3.3.

2.2.3.1 The Palladio approach

The proposed tool is based on the Palladio approach, which combines the modelling and analysis of systems with respect to different quality properties. The Palladio Component Model (PCM) serves as the modelling formalism for system architecture and captures components, interactions, usage scenarios, and resource demands. Based on PCM models, simulation is executed and coordinated by the simulator Slingshot, which provides the analysis backbone to derive quantitative performance insights at design time. The tool produces predictions for component response times and end-to-end latencies, accounting for the impact of deployment and platform specifications.

2.2.3.2 Multi-level performance prediction and simulator coupling

A key goal is a multi-level timing strategy that balances scalability and accuracy. Instead of enforcing a single abstraction level for the entire system, the tool supports interchangeable timing simulator backends and allows selective increases in granularity where precision is important (e.g., critical pipelines or suspected bottlenecks), while keeping the remaining parts efficient for rapid variant exploration. This is essential for heterogeneous platforms, where accelerator behaviour, bus contention, and scheduling effects can dominate latency but are impractical to simulate in full detail for every component and scenario.

Multi-level simulation introduces a consistency problem when switching between abstraction levels, especially when moving from abstract to detailed timing models. The tool resolves this by anchoring timing prediction to a functional simulation backbone that maintains a coherent system and hardware state across simulation cycles. Conceptually, a system emulator such as QEMU[1] can serve as an example for functional simulation and state progression, while a detailed timing simulator such as gem5[2] can serve as an example for microarchitecture-aware timing prediction. In each simulation cycle, the PCM-derived execution flow determines the next functional step and its workload context. Slingshot coordinates the execution so that the functional simulator advances the system state consistently, while the selected timing backend contributes the corresponding timing estimates. This architecture avoids pairwise state translations between every combination of timing simulators and instead provides a coupling concept that supports extensibility as additional timing backends are integrated.

Figure 5 summarizes the approach: PCM-based software execution model simulation with Slingshot orchestrates the run, timing prediction is supplied by pluggable timing simulators, and a functional simulator preserves state continuity, enabling consistent switching between timing abstraction levels.

[1] https://www.qemu.org/

[2] https://www.gem5.org/

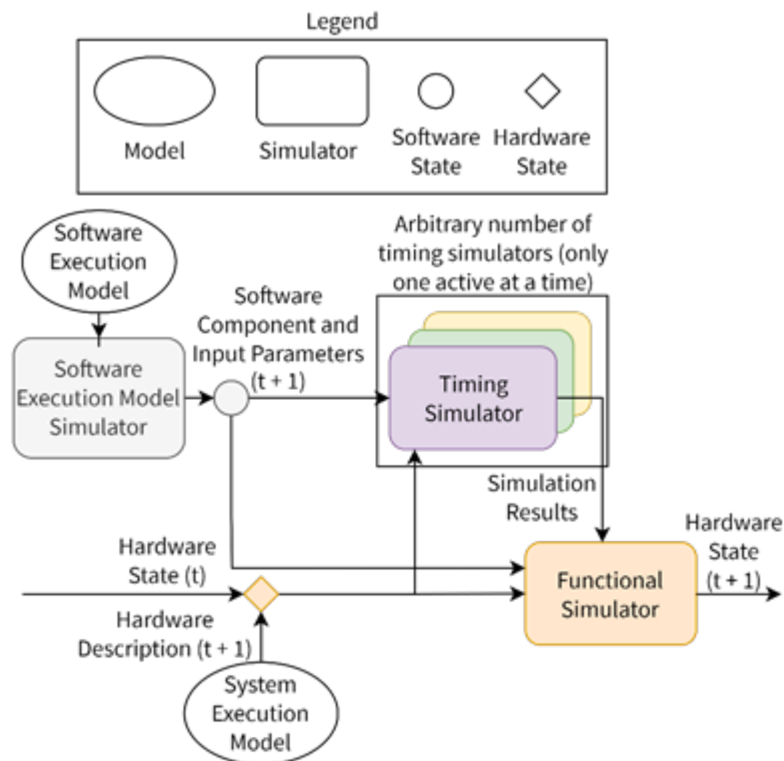

Figure 5: Multi-level simulation approach.

2.2.3.3 Summary

The presented Palladio-based approach enables design-time performance prediction of heterogeneous AI-enabled automotive systems. This tool establishes a PCM-driven workflow for quantitative performance analysis before implementation, explicitly integrates AI inference time into end-to-end timing predictions, and implements a multi-level timing strategy that makes analysis practical for iterative architectural exploration. The coupling via a functional simulation backbone provides a principled solution for state continuity and reduces integration effort when extending the tool with additional timing backends. Uncertainty-related safety assessment contributions are detailed in D3.3.

2.2.4 Current Release

The tool is intended to be released as open-source under the Palladio ecosystem at the PalladioSimulator GitHub organization[3]. This reflects its role as an extension of the Palladio approach: it reuses PCM-based modelling, integrates with existing Palladio analysis workflows, and is structured to allow community-driven extensions, particularly additional simulator adapters for emerging heterogeneous platforms.

[3] https://github.com/PalladioSimulator

Co-funded by the European Union
Chips JU

2.2.5 Interactions with other tools

We are exploring possibilities to integrate the tool into the workflows currently developed in WP3 and BB-E to ensure compatibility and usefulness of inputs and outputs.

2.3 Model-based Representations Tools and QA Plugin

2.3.1 Contributing Partners

TUM, Mercedes-Benz AG and Ferdinand-Steinbeis-Institut der Steinbeis-Stiftung, StatInf

2.3.2 Category

Quality Assurance

2.3.3 Description and Achievements

2.3.3.1 GenAI-empowered test scenario modelling for SDV platforms

Summarization, analysis, and text generation capabilities based on word sequence predictions by large language models (LLMs) make them suitable for tasks such as signal mapping based on a catalogue of Vehicle Signal Specification (VSS) signals. On the other side, vision–language models (VLMs) are leveraged in order to extract relevant information from UML sequence and state diagrams, building upon prior work presented at IntelliSys2025 conference *[2]*.
Before test or target platform code generation, we aim to reduce hallucinations by extracting only scenario-relevant VSS signals. The VSS mapping phase starts from a hierarchical JSON catalogue, where each entry contains fields such as name, datatype, and description. A preprocessing script recursively traverses the hierarchy and produces a flattened list in the form *signalName*, sensor/actuator, datatype (one per line). This list serves as the authoritative reference for valid signals used in downstream prompts. The driving scenario is provided as natural-language text, such as “On entry to S1, CPDS shall activate direct and indirect sensors within 5 s.”, or as a flowchart-like diagram.

The workflow of the initial prototype presented at the DTF symposium [3] is depicted in Figure 6.

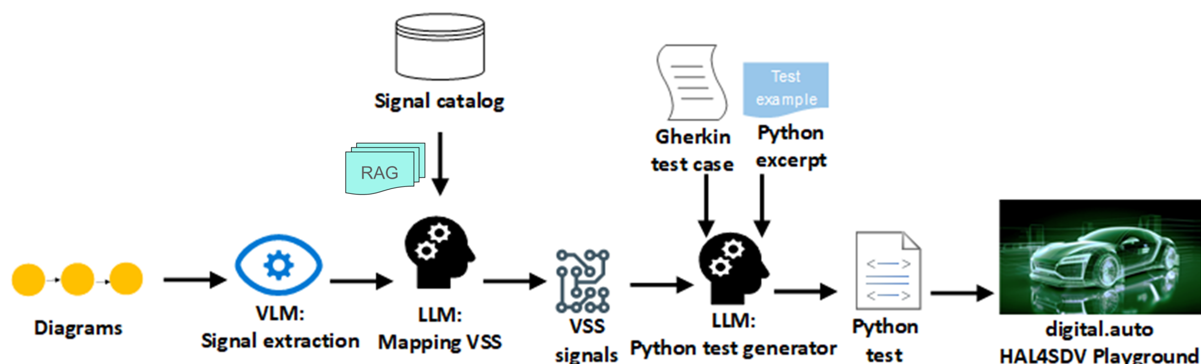

Figure 6: GenAI-driven test generation workflow leveraging VSS specification and Gherkin test cases.

Co-funded by the European Union
Chips JU

We apply Retrieval-Augmented Generation (RAG) to manage the vast VSS catalogue and reduce hallucinations. RAG is needed because the catalogue may contain thousands of entries that exceed the model's context window. The VSS hierarchy and signal naming evolve, so many names are absent from the model's pretraining data. Relying solely on internal training data also makes the model susceptible to changes in terminology and specifications over time, as well as to the use of ambiguous synonyms. In our retrieve-and-re-rank setup, the hierarchical VSS JSON is first flattened into a one-entry-per-line list that serves as an external knowledge base. Each entry and the scenario text are embedded as vectors, and the top-k most similar entries are retrieved. A re-ranking step using a cross-encoder or lightweight rules prioritizes candidates that match the scenario context, for example, "accelerating" implies speed, pedal, and torque signals. The selected candidates are chunked to fit the model's input limit. The LLM is then queried with the scenario, along with each chunk, and the outputs are normalized and merged, removing duplicates to form the final set of signals.

This retrieval-first workflow confines generation to predefined, valid entries, improving precision and scalability without retraining. For the retrieve-and-re-rank pipeline, two models are crucial: (1) a *SentenceTransformer* model [all-MiniLM-L6-v2] converts sentences (or phrases like VSS signal names) into dense vector embeddings, where each vector captures semantic meaning, so similar concepts lie close together in embedding space; and (2) a cross-encoder model [ms-marco-MiniLM-L-6-v2] does not embed inputs independently but looks at pairs of texts together. For each candidate (scenario, VSS signal), it jointly encodes both and predicts a relevance score, which allows it to capture finer contextual relationships (word-by-word alignment), unlike simple cosine similarity. In the re-ranking process, we take the top-k retrieved candidates from the first stage of retrieval. The cross-encoder scores each one based on its relevance to the scenario. The results are then sorted by this score. The top N highest-scoring signals become the final ones selected for the given prompt.

Moreover, a structured natural-language prompt is created that combines the scenario with the available signals. In our case, the list of available signals is based on the output of the retrieve-and-re-rank workflow. The prompt explicitly instructs the model to use only the provided signals and to avoid introducing extraneous signals. This is an example of such a prompt, based on the Child Presence Detection scenario developed as a collaboration among project partners:

Based on the following driving scenario: [On S2 entry, CPDS transitions to S3 and notifies the driver via mobile app push plus key fob (if available) within 10 s, including vehicle location and a countdown for T_ACK_1 (5 minutes).]

Extract all relevant Vehicle Signal Specification (VSS) signals that could describe this scenario.

The full list of available VSS signals is:

***[Vehicle.Cabin.ChildPresenceDetection.IsChildDetected: boolean,
Vehicle.Cabin.Infotainment.DriverAppNotification.IsDriverNotified: boolean,
Vehicle.Cabin.Infotainment.DriverAppNotification.HasDriverAcknowledged: boolean]***

Return only the subset of signals from the list that are relevant, as a comma-separated list (no explanations).

The selected LLM is queried with the previously constructed prompt. During generation, the model matches the scenario context to the signal descriptions in the list and returns a filtered, comma-separated subset of signals. The model output is cleaned to remove formatting artifacts (for example, Markdown elements) and normalized by trimming whitespace and standardizing delimiters, producing a validated list of relevant VSS signals. The extracted signals can be compared against ground truth for evaluation, using exact match or fuzzy matching techniques to measure selection correctness. By using this workflow, the RAG-empowered extraction ensures that the model selects only predefined, valid signals from a potentially large list of available signals (in practice, often more than a thousand) specific to a particular scenario, allowing the extraction of VSS signal subsets for testing, prototyping, or simulation without manual inspection of large signal inventories. Once the VSS signals are identified, we proceed with the next steps in the test generation flow.

While LLMs support summarization, analysis, and text generation, making them suitable for tasks in the target pipeline requires to map requirement-level concepts to VSS paths and generating code from test case specifications. For visual artifacts, we use vision–language models (VLMs) to extract relevant information from UML sequence and state diagrams. We evaluate both hosted commercial models, such as GPT-4o-mini and GPT-5, and locally deployable models with fewer parameters, including Llama-based open-source variants.

We evaluate four LLMs for VSS signal mapping and Python test script generation. The evaluation is conducted using the Child Presence Detection System (CPDS) case and targets the *digital.auto* playground for software-in-the-loop testing, with Python as the execution language, as described in [3]. Specifically, we compare GPT-4o-mini with a locally deployed Vicuna model under two candidate-pool conditions: a curated subset of sixteen commonly used signals and the full in-vehicle catalogue of 982 signals. Decoding parameters are fixed across all runs, with temperature set to zero, a maximum of 1024 tokens, and top-k set to 25. The results are given in Table 1.

Table 1: VSS Signal Mapping Results for LLMs in case of CPDS

Model	Candidate Signals	Correct / Expected	Additional (False Positives)
GPT-4o-mini	16	4 / 4	0
Vicuna 7B (local)	16	4 / 4	7
GPT-4o-mini	982	2 / 4	4
Vicuna 7B (local)	982	4 / 4	19

The results show that restricting the candidate signal pool significantly improves mapping quality. With a shortlist of sixteen candidates, GPT-4o-mini correctly identifies all ground-truth signals without false positives, while the Vicuna model also achieves full recall but introduces additional incorrect signals. When using the full catalogue of 982 signals, performance degrades for both models. Vicuna maintains recall but produces false positives, while GPT-4o-mini misses half of the required signals. These results highlight the importance of prefiltering the VSS catalogue to a small, scenario-relevant shortlist. Without retrieval or candidate filtering, neither model reliably selects the correct signals. While commercial

models achieve higher precision, locally deployable models remain important for sensitive or proprietary scenarios, and our pipeline therefore supports local deployment.

Additionally, we also evaluate VLMs for diagram-based VSS signal extraction using a UML state diagram with eight states and eleven transitions, together with a candidate pool of 16 VSS signals. The results are given in Table 2.

Table 2: VLM Performance on Diagram-Based VSS Signal Extraction (CPDS).

Model	Precision	Recall
Gemini 2.5 Pro	0.956	0.956
GPT-4o	0.780	0.867
Grok 3	0.762	0.778
Qwen 2.5 VL 72B (local)	0.843	0.822
Llama 4 Maverick 17B (local)	0.787	0.822
InternVL 3 78B (local)	0.655	0.711

Among hosted models, Gemini 2.5 Pro achieves the highest precision and recall. Among locally deployable models, Qwen 2.5 VL 72B performs best. For specifications that contain sensitive or proprietary information, local deployment avoids sharing artifacts with external providers.

More details about this contribution are contained in the publication entitled „Req2Road: A GenAI Pipeline for SDV Test Artifact Generation and On-Vehicle Execution“, submitted for CAiSE2026 conference, in collaboration with Mercedes-Benz AG, while the initial works can be found in [3].

2.3.3.2 LLM-driven regulatory compliance test scenario modelling for simulation platforms

TUM also contributes an LLM-driven framework for automated modelling of automotive test scenarios aiming at ADAS capabilities in a full-fledged, realistic 3D environment like CARLA, building upon our previous work from [4] [5]. This way, we aim to complement the modelling capabilities and bridge the gap when it comes to the evaluation of automated hardware abstraction and integration, especially when it comes to sensors and actuators relevant for autonomous driving.

Using regulation-compliant test scenarios extracted from UN regulations like Regulation No. 152 [UN152], together with similar natural language requirements from other safety standards, this component automatically generates simulation-ready configuration code for a CARLA based virtual test environment. The objective is to transform abstract, human-readable descriptions of safety-critical behaviour into a fully specified set of parameters that can be executed and evaluated without manual coding. This allows regulatory scenarios, which are typically published as descriptions and tables, to be reproduced at scale inside an autonomous driving simulator.

The system decomposes each scenario into three complementary requirement categories that collectively cover the crucial aspects of simulation. The first category, vehicle definition, captures properties such as vehicle make and model, physical footprint, actuator behaviour, and integrated or externally mounted sensor suites (such as cameras, radars, and lidar). Descriptions of sensor placement, nominal field of view, and update rates are included so the

simulator can reproduce perception capabilities that are consistent with the intended test conditions. The second category, pre-conditions, expresses how the virtual world should be initialized, including road geometry, lane topology, weather and lighting, and the spawn placement, orientation, and initial motion of all relevant agents. This category defines the spatial and temporal context in which the behaviour specified by UN Regulation 152 should unfold. The third category, post conditions, codifies the measurements that must be collected during the run and the criteria used to judge whether the observed vehicle behaviour satisfies the expected outcome. These post conditions include telemetry streams such as vehicle pose, velocity, acceleration, and sensor data, as well as derived safety metrics (e.g., minimum distance to other agents or time to collision).

Each category is processed by a specific large language model-driven pipeline based on GPT-4o. These pipelines accept natural language requirements as input and translate them into structured configuration objects, implemented as JSON documents that can be consumed directly by downstream software. The approach builds on the methodology presented in [5], splitting the translation task into three targeted sub-processes that can be improved independently. By separating vehicle configuration, scene initialization, and evaluation logic, the system enables modular reuse and reduces the likelihood of cross-category errors in the generated configuration. Once the three generated configuration products (vehicle, pre-condition, and post-condition) are available, they are merged into a single simulation configuration bundle. This unified bundle contains all information necessary to instantiate the scenario inside CARLA without further manual editing. The CARLA runtime then consumes the configuration, constructs the synthetic environment, spawns, and parameterizes all actors, and executes the simulation while logging the specified telemetry.

Figure 7 summarizes the overall architecture and data flow between the three pipelines and the simulation engine. This design enables consistent reproduction of regulatory scenarios directly from textual descriptions, creates a rapid pathway for validating new or revised regulations, and on the other side, also supports scalable scenario generation for autonomous vehicle testing and benchmarking.

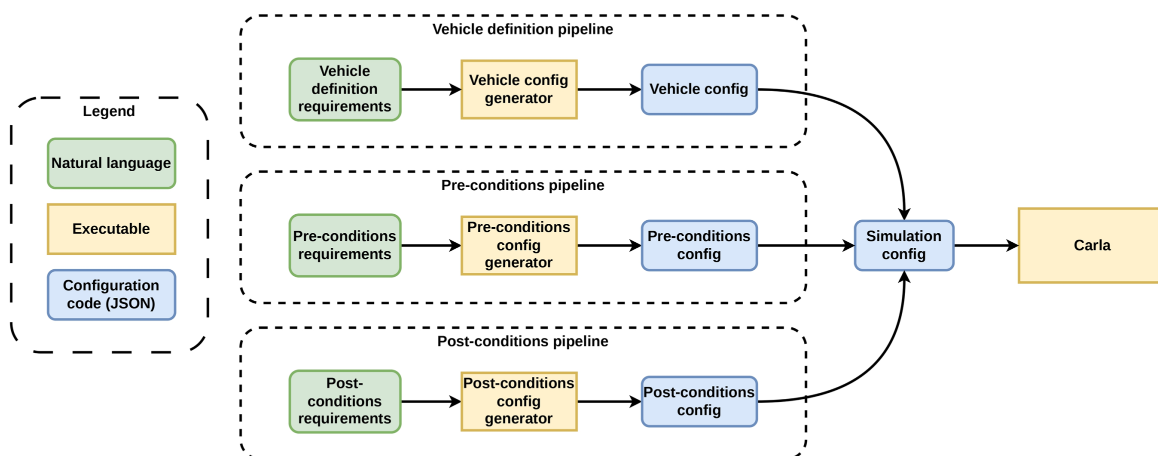

Figure 7: Regulatory compliance ADAS test scenario modelling using LLMs.

In addition, we have evaluated several prompting techniques for LLMs (ICL – In-Context Learning, COT – Chain of Thought), as summarized in Table 3.

Table 3: Integrated Results - Prompting techniques across pipelines for regulatory simulation scenarios.

Pipeline	Metric	Simple	ICL	COT
Pre-conditions	Avg TPR	0.78	0.95	0.95
Pre-conditions	Pass@1	0.0	0.65	0.65
Pre-conditions	Pass@5	0.0	1.0	1.0
Pre-conditions	Pass@10	0.0	1.0	1.0
Post-conditions	Avg TPR	0.84	0.98	1.0
Post-conditions	Pass@1	0.45	0.9	1.0
Post-conditions	Pass@5	0.97	1.0	1.0
Post-conditions	Pass@10	1.0	1.0	1.0

2.3.3.3 LLM-empowered event-chain based SDV behaviour modelling

The aim of this workflow (depicted in Figure 8) is to enable that developer-provided behaviour descriptions in either textual or visual form are transformed into validated and optimized automotive platform code through iterative analysis and LLM-driven processing, ensuring correctness by design, functional safety, and alignment with timing constraints thanks to synergy with model-driven engineering and formal validation methods.

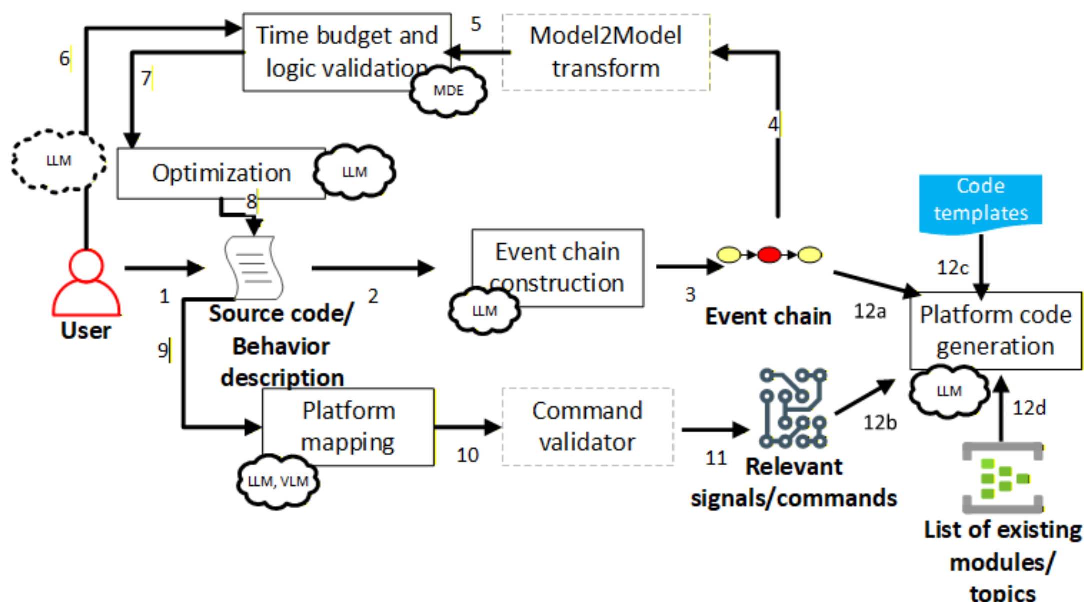

Figure 8: Workflow for LLM-driven SDV behaviour modelling relying on formal methods.

Co-funded by the European Union

Chips JU

The process can be summarized in the following stages:

User Input (Step 1): The user provides a natural-language or source-level behaviour description expressing the intended function of the system. This specification captures timing, sequencing, and expected reactions to system events.

Initial Processing and Interpretation (Step 2): An LLM parses the input and extracts task semantics, producing an intermediate representation suitable for transformation into event-based logic.

Event-Chain Construction (Step 3): The interpreted behaviour description is converted into an explicit event chain model. The resulting model represents stimuli, reactions, and dependencies as an ordered chain that can be validated and further analysed.

Transformation to Platform-Relevant Structures (Step 4): The event-chain specification is consumed by a model transformation process, which produces a model aligned with the constraints of the target automotive software platform or specific external tools used for profiling of time-related and other aspects.

Time Budget and Logic Validation (Step 5): The transformed model is checked against timing constraints and logical correctness requirements. Violations detected at this stage indicate that the design is either unsafe or incompatible with platform assumptions.

Optimization Loop (Steps 6-8): Validation feedback is used to correct and improve the model.

- Step 6: Issues are reported back to the user and LLM-based reasoning processes.
- Step 7: The system proposes or applies refinements, which may include simplifying event connectivity, reshaping timing chains, or redefining execution triggers.
- Step 8: Optimized requirements are regenerated, and the revised behaviour description is fed back into the modelling chain.

Platform Mapping (Step 9): Once validated, the refined event chain is mapped to platform-specific constructs, including APIs, scheduling primitives, and execution contexts. LLM and VLM tools support this mapping by locating suitable platform elements.

Command and Syntax Validation (Step 10-11): Command sequences derived from the mapped behaviour are checked for syntactic and logical correctness. Relevant signals and commands are extracted, ensuring that the resulting code will activate and interact with the correct platform modules and hardware interfaces.

Code Generation (Steps 12a–12d): With validated event logic and resolved mappings, platform-ready source code is generated considering the following aspects:

A) Generation of implementation skeletons from code templates.

B) List of relevant commands, topics, and signals.

C) Linking to existing reusable modules stored in the platform library.

D) Final assembly and configuration to form a deployable platform code.

The details of this work are provided by a publication submitted to the GACLM2026 conference [6].

2.3.4 Current Release

We plan to open-source all the related software assets. Currently, HAL4SDV partners can ask for access to the source code at the following address: https://gitlab.lrz.de/hal4sdv/vss-test-generation

2.3.5 Interactions with other tools

This activity provides input for LLM-based test generation to the module 2.6 described in D3.2.

2.4 Hypervisor support

2.4.1 Contributing Partners

POLIMI, STMicroelectronics, Resiltech

2.4.2 Category

Hypervisors

2.4.3 Description and Achievements

This contribution addresses the critical challenges inherent in the automotive industry's transition from distributed Electronic/Electrical (E/E) architectures to centralized Zonal Architectures. As the demand for consolidating heterogeneous functionalities—ranging from real-time control to high-bandwidth connectivity—onto single hardware platforms grows, the role of the hypervisor becomes pivotal. Our research provides a comprehensive evaluation of virtualization technologies across the full spectrum of embedded processing, investigating both safety-centric microcontrollers (ARM Cortex-R52+) and high-performance application processors (ARM Cortex-A76). By leveraging the Bao and Xen hypervisors, this work establishes architectural blueprints for ensuring performance, isolation, and efficient resource sharing in Mixed-Criticality Systems (MCS).

2.4.3.1 Efficient I/O Virtualization on Safety Controllers

As part of a collaboration between POLIMI and STMicroelectronics, a specific focus was on the validation of Ethernet virtualization on resource-constrained, safety-oriented processors. While virtualization is well-established on powerful chips, its application on MCUs like the Cortex-R52+ presents unique challenges regarding overhead and resource contention. We developed and evaluated a novel I/O virtualization architecture using the Bao hypervisor, designed to efficiently share Automotive Ethernet Network Interface Controllers (NICs) among multiple guests.

Co-funded by the European Union

Chips JU

The proposed architecture introduces a split-driver model where a backend "broker" VM retains hardware access and exposes virtual interfaces to frontend VMs via the hypervisor's Inter-Process Communication (IPC) facilities. Crucially, we demonstrated that mapping guest virtual interfaces directly to distinct hardware DMA channels enables a viable sharing mechanism that respects the hardware's constraints. Through synthetic workload testing on an industrial automotive platform, we identified that while the architecture is fundamentally sound, the hypervisor's IPC mechanism constitutes the primary performance bottleneck. This investigation provides a clear roadmap for future optimizations, confirming that with targeted improvements to the IPC path, safety-centric MCUs can effectively host virtualized, high-bandwidth networking stacks.

2.4.3.2 Strict Isolation and Performance Determinism in High-Performance Nodes

Complementing the work on microcontroller I/O, we investigated the limits of temporal and spatial isolation on high-performance nodes using the Xen hypervisor as a collaboration between POLIMI and Resiltech. As these processors (e.g., Cortex-A76) increasingly host mixed workloads, ensuring that non-critical, "noisy" guests do not degrade the performance of safety-critical functions is paramount. We conducted a rigorous analysis of interference channels, specifically focusing on shared memory hierarchy and interrupt latency.

Our experiments confirmed that Xen, when configured as a separation kernel, incurs negligible overhead on raw computational throughput for bare-metal workloads, maintaining near-native performance. However, we identified a consistent interrupt propagation delay (~2 µs), highlighting the cost of virtualization in strict real-time contexts. To mitigate memory subsystem interference—where a critical guest suffers due to cache eviction by a non-critical neighbour—we implemented and evaluated Cache Colouring techniques. The results were significant, demonstrating a reduction in interference regarding memory writes by approximately 24%. Furthermore, we characterized the "saturation point" of this technique, revealing that aggressive colouring can paradoxically lead to bus bandwidth saturation if guests are overly constrained to L1/L2 caches.

2.4.3.3 Summary

These activities successfully validate the technical feasibility of hypervisors in next-generation mixed-criticality environments. We have achieved the following:

1. **Architecture Validation:** Proven the viability of splitting Ethernet drivers via DMA channel mapping on limited-resource MCUs.
2. **Interference Mitigation:** Quantified the efficacy of Cache Colouring in restoring up to 24% of lost performance in critical workloads under stress.
3. **Overhead Characterization:** Precisely measured the virtualization "tax," pinpointing IPC on microcontrollers and interrupt injection on application processors as key latency drivers.

Co-funded by the European Union
Chips JU

2.4.4 Current Release

We do not plan to upstream changes to Bao as these are related to a proprietary STMicroelectronics platform.

We also do not plan to open source the benchmarking suite for Xen, but HAL4SDV partners can request access to our repo: https://github.com/polimi-aos-lab/xen-perf-measurements

2.4.5 Interactions with other tools

We are in the process of understanding whether these artifacts can be introduced in UC3.

2.5 Middleware for Performance (DDS, Rust)

2.5.1 Contributing Partners

POLIMI, STMicroelectronics (for the DDS part)

2.5.2 Category

Middleware

2.5.3 Description and Achievements

This contribution was dedicated to advancing the foundational software infrastructure of SDVs, specifically focusing on the critical intersection of safety-verified virtualization and deterministic communication middleware. To support the shift toward modular, mixed-criticality automotive architectures, our work focused on two complementary layers of the embedded stack: the development of a memory-safe execution environment and the rigorous characterization of real-time data distribution protocols.

At the virtualization layer, we addressed the safety challenges inherent in C-based system programming by developing Armor SK, a lightweight separation kernel and static partitioning hypervisor written entirely in Rust. Designed effectively for MPU-based architectures such as ARMv8-R and RISC-V (extended with SPMP), Armor SK leverages Rust's compile-time guarantees and runtime bound-checking to eliminate common memory vulnerabilities without compromising performance. Through a comparative analysis against the Bao research-prototype hypervisor, we demonstrated that Rust abstractions—specifically generics and traits—significantly enhance code portability and maintainability compared to traditional C preprocessor directives. Crucially, the achievements of this work include a reduction in the text section size by approximately 45% and execution metrics that match or exceed those of C-based counterparts; notably, Armor SK achieved a 40% speedup in Inter-Processor Interrupt (IPI) handling while maintaining negligible overhead from runtime safety checks.

We simultaneously tackled the predictability of the communication layer by optimizing the eProsima Micro XRCE-DDS middleware on STMicroelectronics Stellar MCUs. Moving beyond black-box analysis, we conducted an in-depth characterization of the protocol over Ethernet to identify system dimensions and bottlenecks. The primary achievement in this domain was the formulation of a mathematical estimation methodology that allows for the static

prediction of system performance (Latency, Throughput, Memory Usage, and CPU Load) without requiring full application deployment. By mapping the dependencies between middleware parameters and performance KPIs, we created a methodology that not only generates optimized DDS/XRCE applications but also enables developers to anticipate resource usage and prioritize specific metrics during the design phase.

2.5.4 Current Release

We do not plan to open-source Armor SK but HAL4SDV partners can request access to our repo: https://github.com/polimi-aos-lab/aSK

2.5.5 Interactions with other tools

We are trying to understand how the work on a Rust hypervisor can fit into BB-E.

2.6 VPSim Support

2.6.1 Contributing Partners

CEA

2.6.2 Category

Virtual Platform

2.6.3 Description and Achievements

As automotive systems evolve toward extraordinarily complex, software-defined vehicle (SDV) architectures, virtual prototyping solutions must address new requirements in terms of scalability, performance, automation, and rapid design-space exploration. VPSim responds to these challenges by providing a flexible and efficient virtual prototyping framework that enables early and continuous system-level validation.

VPSim [7] [8] is a powerful and versatile platform for modelling and simulating complex systems, with strong relevance for automotive, HPC, and heterogeneous multicore architectures. It enables the composition and evaluation of complete automotive computing platforms, from multi-SoC architectures to integrated software stacks. Designed to support early-stage HW/SW co-design, VPSim is a modular and highly configurable framework that facilitates architectural exploration—allowing performance evaluation of multiple platform configurations—and advanced software development, including execution, profiling, and debugging of full software stacks (e.g., boot firmware, hypervisors, and automotive workloads) on virtual platforms.

Co-funded by the European Union
Chips JU

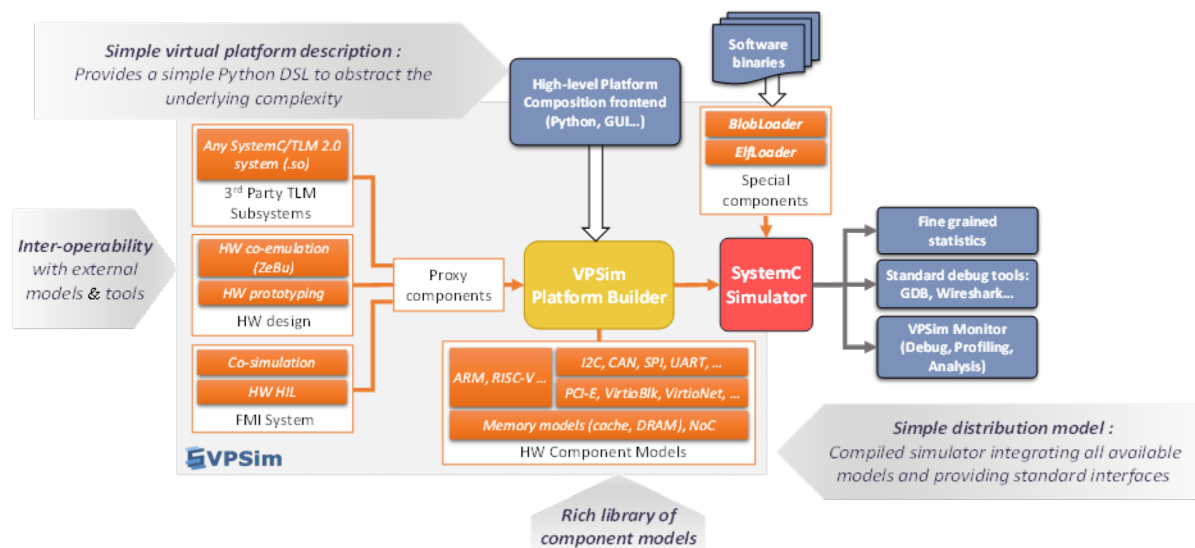

Figure 9: VPSim platform overview.

Figure 9 gives a global overview of the VPSim tool, which features a diverse collection of CPU models, such as ARM and RISC-V, as well as models of several peripherals and buses. These components are fully standard-compliant SystemC/TLM 2.0 following the Loosely Timed paradigm with a blocking communication interface, which offers a good trade-off between execution speed and accuracy. The main supported model provider in VPSim is the open-source system emulator QEMU [9], which enables high simulation speed. Other model providers can also be imported from the ARM Fast Models, Open Virtual Platforms, and equivalent tools. Users can develop models directly in SystemC or Python to extend their virtual prototypes.

Furthermore, VPSim stands out for its capacity to accommodate external subsystems through standard and non-standard interfaces like SystemC, Python, QEMU devices and HW designs. Moreover, it can connect with other simulation tools and modelling software through a Functional Mockup Interface (FMI) based co-simulation, as demonstrated in [10] [11]. VPSim offers a large set of performance counters and statistics for profiling and benchmarking purposes. These counters can be classified into two categories: functional and time-related counters. Functional counters focus on instruction counts, including the number of instructions and memory accesses. In comparison, Time-related counters involve simulated time and all associated factors, such as memory bandwidth and latencies.

2.6.4 Current Release

Within the HAL4SDV project, we contributed to enhancing the benchmarking capabilities of the VPSim platform to better address the real-time constraints of automotive systems and Software-Defined Vehicle (SDV) architectures.

Beyond the traditional Unix-based execution mode, we introduced a bare-metal execution mode that enables more accurate and deterministic performance estimation. This approach allows the concurrent evaluation of multiple tasks executing in parallel without operating

Co-funded by the European Union
Chips JU

system interference, which is particularly relevant for real-time and safety-critical automotive workloads.

To support applications modelled as task graphs, we developed a snapshot-based profiling mechanism that is better suited for analysing temporal behaviour, task interactions, and execution overlaps under real-time constraints. This profiling approach facilitates a more precise assessment of timing behaviour and resource contention.

To enable these capabilities, dedicated libraries were developed and integrated into VPSim, extending its benchmarking, profiling, and performance analysis features and strengthening its support for early-stage HW/SW co-design of real-time automotive systems. More precisely, two APIs are now available in C/C++ and Python, for a profiled use-case to trigger instrumentation and generate task-based timing and memory access reports.

VPSim is available online at: https://github.com/CEA-LIST/VPSim

2.6.5 Interactions with other tools

As part of the WP 3.3, these new capabilities are integrated with tools such as Polygraph. Polygraph is a task-based schedule to provide accurate timings, necessary for any relevant timing analysis or scheduling.

2.7 Build- and Deployment-Pipeline for Xen-based Systems

2.7.1 Contributing Partners

VIF

2.7.2 Category

Virtual Platform

2.7.3 Description and Achievements

Virtualization offers many advantages in software development. Existing hardware resources can be utilized more effectively, solutions are more scalable, more flexible, and faster to deploy. Due to all these benefits, virtualization is widely used for cloud applications.

Such virtualization solutions would bring significant benefits to the automotive domain in the context of "software-defined vehicles" (SDV). We are therefore investigating to apply these open-source virtualization solutions (i.e., the Xen hypervisor) from the cloud domain to SDVs. In this context, it is essential to assess the suitability of these open-source solutions for the automotive industry and whether they are suitable for ensuring compliance with required standards (e.g., ISO 26262) as well as performance requirements (i.e., real-time behaviour).

To allow for a more efficient development in accordance with the DevOps scheme, we are setting up a build- and deployment-pipeline for Xen-based systems in the context of SDVs. It

Co-funded by the European Union
ChipsJU

allows for automatic build of the Xen host, as well as Xen guests (consisting of the applications and an operating system).

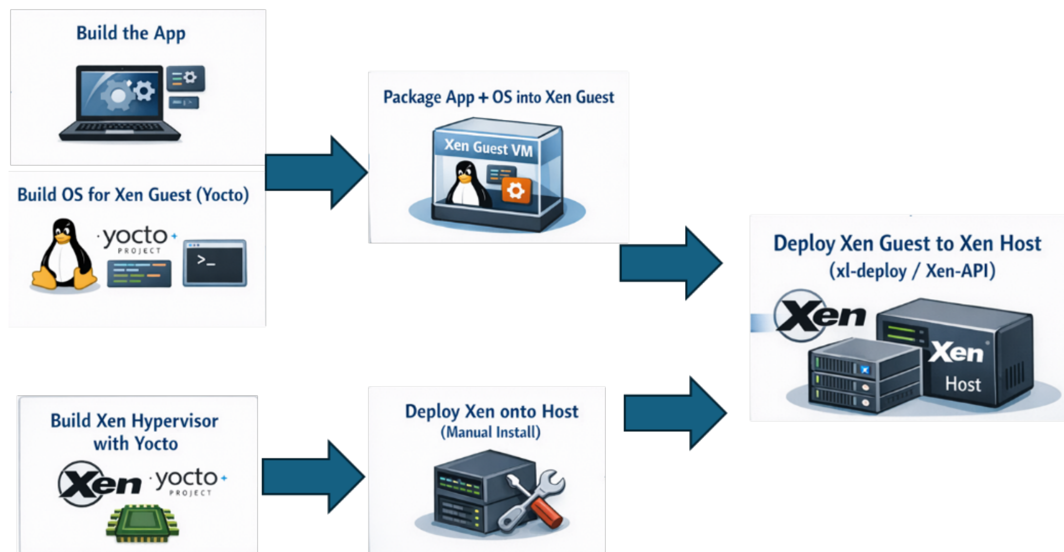

Figure 10: Build- and deployment-pipeline for Xen-based systems.

2.7.4 Current Release

These Xen guests, which contain virtualized automotive applications, can be deployed on various Xen hosts (lab setup in office, cloud, prototyping vehicle). On these Xen hosts, various verification activities of the virtualized applications will be performed. Once the verification succeeded, the same Xen guests can be deployed onto the final Xen host (inside the SOP-ready vehicle) without the need of further modifications.

To support the second phase of the DevOps schema (i.e., operations), we are currently investigating the possibilities of monitoring the Xen guests once they are deployed onto the SOP-ready vehicle. We are therefore investigating the capabilities of the Xen API (XAPI). It can provide various information about the Xen host and all its Xen guests. We are also investigating how the XAPI can be mapped to the VSS protocol.

2.7.5 Interactions with other tools

- T3.2: It was planned to use RISC-V hardware with the same Xen software baseline. However, the Xen hypervisor is not yet fully supported on RISC-V. Therefore, this activity is currently not active.
- T3.3: Profiling and tracing tools to perform mixed-criticality analysis (freedom-from-interference should be demonstrated).

Co-funded by the European Union

3 Conclusion

This deliverable outlines the modelling and simulation activities conducted in Task 3.4 of the HAL4SDV project. The described project outcomes address critical challenges by providing tools and methodologies that facilitate early exploration of design spaces, performance and safety analysis, quality assurance, and system integration across diverse automotive computing architectures.

The activities described in this deliverable cover multiple abstraction levels, including virtual prototyping, timing-aware platform simulation, model-based performance prediction, AI-assisted test generation, and virtualization and deployment workflows. By integrating low-level virtual platforms with higher-level architectural modelling and analysis tools, Task 3.4 enables a systematic evaluation of design alternatives even before hardware is available and large-scale software integration occurs. This approach helps to reduce development risks and costs.

Overall, the deliverable presents a cohesive set of modelling and simulation capabilities that complement the RISC-V-focused activities of Task 3.2 and contribute to the broader goals of WP3. The tools and methodologies highlighted in this deliverable lay the groundwork for integrated HAL4SDV toolchains and support the project's aim of facilitating robust, scalable, and reusable development workflows for the next generation of software-defined vehicles.

Co-funded by the European Union

Chips JU

4 References

[1] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini and D. Rossi, “GVSoC: a highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021.

[2] N. Petrovic, Y. Zhang, M. Maaroufi, K.-Y. Chao, L. Mazur, F. Pan, V. Zolfaghari and A. Knoll, “Multi-modal Summarization in Model-Based Engineering: Automotive Software Development Case Study. In: Arai, K. (eds) Intelligent Systems and Applications,” in *Intelligent Systems Conference*, 2025.

[3] D. Zyberaj, L. Mazur, N. Petrovic, P. Verma, P. Hirmer, D. Slama, X. Cheng and A. Knoll, “GenAI-based test case generation and execution in SDV platform,” *arXiv preprint arXiv:2509.05112,* 2025.

[4] N. Petrovic, F. Pan, V. Zolfaghari, K. Lebioda, A. Schamschurko and A. Knoll, “GenAI for Automotive Software Development: From Requirements to Wheels,” *arXiv preprint arXiv:2507.18223,* 2025.

[5] K. Lebioda, N. Petrovic, F. Pan, V. Zolfaghari, A. Schamschurko and A. Knoll, “Are requirements really all you need? using llms to generate configuration code: A case study in automotive simulations,” *IEEE Access,* 2025.

[6] N. Petrovic, V. Zolfaghari, F. Pan and A. Knoll, “LLM-Empowered Functional Safety and Security by Design in Automotive System,” *arXiv preprint arXiv:2601.02215,* 2026.

[7] A. Charif, G. Busnot, R. Mameesh, T. Sassolas and N. Ventroux, “Fast virtual prototyping for embedded computing systems design and exploration,” in *Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools*, 2019.

[8] F. Jebali, O. Matoussi, A. Wicaksana, A. Charif and L. Zaourar, “Decoupling processor and memory hierarchy simulators for efficient design space exploration,” in *System Engineering for constrained embedded systems*, 2022.

[9] F. Bellard, “QEMU, a fast and portable dynamic translator,” in *USENIX annual technical conference, FREENIX Track*, 2005.

[10] S. E. Saidi, A. Charif, T. Sassolas, P.-G. Le Guay, H. V. Souza and N. Ventroux, “Fast virtual prototyping of cyber-physical systems using systemc and fmi: ADAS use case,” in *Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP'19)*, 2019.

[11] C. Bernardeschi, P. Dini, A. Domenici, A. Mouhagir, M. Palmieri, S. Saponara, T. Sassolas and L. Zaourar, “Co-simulation of a model predictive control system for automotive applications,” in *International Conference on Software Engineering and Formal Methods*, 2021.

Co-funded by the European Union
ChipsJU