2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Exploring Software-Defined Vehicles through Digital Twin Simulation with Extensible Prototyping FPGA: A Tool Perspective

Tasneem A. Awaad, Mohamed Ellethy, Mohamed AbdElSalam

Siemens EDA, Egypt

{ tasneem.awaad.ext, mohamed.ellethy.ext, mohamed.abd_el_salam_ahmed}@siemens.com

***Abstract*—In this paper, we present our compositional simulation interconnect framework for exploring Software-defined Vehicle implementations from cloud to edge. The framework enables heterogeneous clients' connections to create digital twins of Electronic Control Units (ECUs) of a Vehicle at different abstraction levels connected to real traffic scenario simulators, sensor models and mechatronic systems. The design methodology behind the framework is based on three enabling pillars: the gateway concept, Digital Twin Description Language (DTDL), and automation flow. An adaptive cruise control application is used as a demonstration of the interconnect framework.**

Keywords— software-defined vehicles; digital twins; co-simulation; pre-silicon verification

I. INTRODUCTION

In the rapidly evolving landscape of the automotive industry, the integration of advanced digital technologies has become paramount for innovation and competitiveness. One such transformative approach is the concept of Software-Defined Products (SDP) and the vehicle is just an example. Software-Defined Vehicle (SDV) is a new technology term that describes a vehicle whose features and functions are primarily enabled through software; the concept has grown rapidly because of the ongoing transformation from a vehicle that is mainly hardware-based to a vehicle that is a software-centric electronic device on wheels. SDV implementations are mainly driven by cloud-native development [1].

A key enabler of this transition to SDVs is the utilization of digital twin technology that describes a virtual replica of a physical asset capable of mimicking its behavior to better assess and understand its operation under both hypothetical and real scenarios [2]. Digital twin implementations are also driven by cloud-native development. Leveraging both technologies in automotive has a great benefit in software validation. For example, simulating *"on the cloud"*, the vehicle electronic control unit (ECU) software under real traffic conditions with ego vehicle dynamics, a noisy sensor or an adversarial attack and monitoring the safety and security of the vehicle software before deploying the software on the edge ECU [3].

"Pre-Silicon" ECUs include either Virtual System on a Chip (SoC) Platforms (running locally on a PC or a Virtual Private Cloud) or Register Transfer Language (RTL) representation of SoC Designs written in Hardware Description Languages (HDL) running on Hardware Assisted Verification (HAV) Platforms including HW emulation [4] and Field Programmable Gate Array (FPGA) Prototyping Systems [5]. High-computing ECUs include embedded microcontrollers with legacy Controller Area Network (CAN)/Local Interconnect Network (LIN) connectivity, and high-end processors with automotive ethernet switch (zonal) connectivity, running hypervisors, Real-Time OS, middleware, and applications [6]. Rapid advances in Advanced Driver-Assistance System (ADAS) have influenced the design of these servers as well to include artificial intelligence (AI) accelerators for perception.

Building an SDV-based methodology to deploy the SW developed on the cloud on a Pre-Silicon representation of an edge ECU has the benefit of evaluating different design architectures of the ECU design before fabrication. In this paper, we demonstrate the usage of our compositional simulation interconnect framework (Veloce System

Interconnect (VSI)) [7] that streamlines the verification process of the software implementation of SDV in the Software-in-the-loop (SIL) layer to deploying the SDV algorithm on a FPGA prototyping system (proFPGA Quad motherboard [5]) as an edge ECU as shown in Figure 1. The FPGA prototyping system is programmed with a high-performance RISC-V-based SoC design.

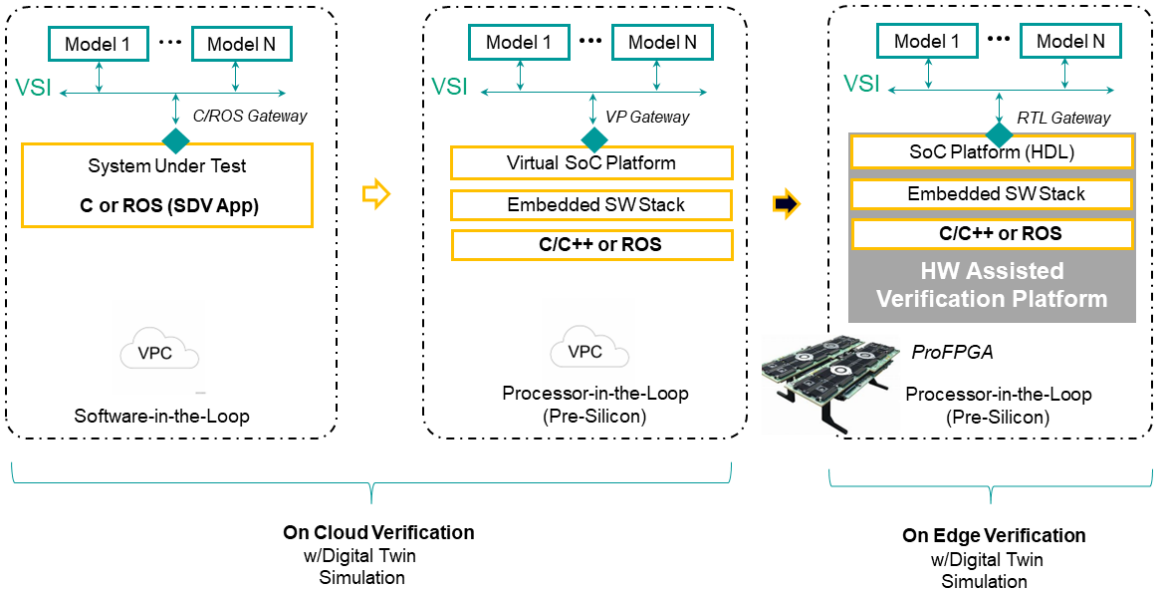

Figure 1. SDV-based Methodology w/FPGA Prototyping.

Extending SDV implementation to a complete digital twin requires the existence of a system interconnect framework enabling heterogeneous clients' connections and co-simulating all digital twin elements, in a synchronized and deterministic manner. There are generally three approaches for combining models for compositional simulation. 1) Model Transformation: The entire system model is built into one tool for the purpose of simulation. 2) Model Exchange: The exchange of models between tools to run a simulation in one of them. 3) Co-simulation: Loosely coupling two or more simulators [8]. Each approach has its pros and cons. Co-simulation maintains the operational semantics that is why we have adopted this approach when building our interconnect framework.

II. COMPOSITIONAL SIMULATION FRAMEWORK

We have developed an optimized interconnect framework [7] enabling heterogeneous clients' connections to create digital twins in the areas of automotive, robotics, avionics, and medical verticals. The interconnect features conformance with well-established electronic design automation standards: SystemC TLM 2.0 [9], JModelica FMI 2.0 [10], and Inter-Process Communication (IPC) connections.

The framework allows mixed-fidelity modelling. It allows connections of different client types e.g. sensor/scenario simulators, mechatronic systems simulators, dashboard software, cloud services, compute/think modules at different abstraction levels including C/C++ modules, and any platforms that have C/C++ interfaces, SystemC TLM models, Python modules, ROS modules, virtual platforms (VPs), RTL designs on digital simulators/ HW assisted verification platforms and external HW boards with physical connections (e.g. Ethernet, CAN). The design methodology behind the Digital Twin Interconnect Fabric is based on three enabling pillars: the gateway concept, Digital Twin Description Language (DTDL) and automation flow for building the interconnect framework, as shown in Figure 2.

A. The Gateway Concept

Each digital twin component port is connected to the digital twin interconnect fabric backplane server through a "gateway". A gateway executes data conversion and routing into digital twin interconnect fabric simulated & physical communication protocols. Gateways are categorized into Language2Protocol

(C++/C#/SystemC/Python/ROS), Pre-Silicon (VPs, RTL), Post-Silicon (Hardware-in-the-loop (HIL)) and Specialized (Sensors, Actuators, Cloud Services, 3rd party Tools). Communication Protocols of the backplane server include for example: Ethernet, CAN, Advanced Extensible Interface (AXI), etc.

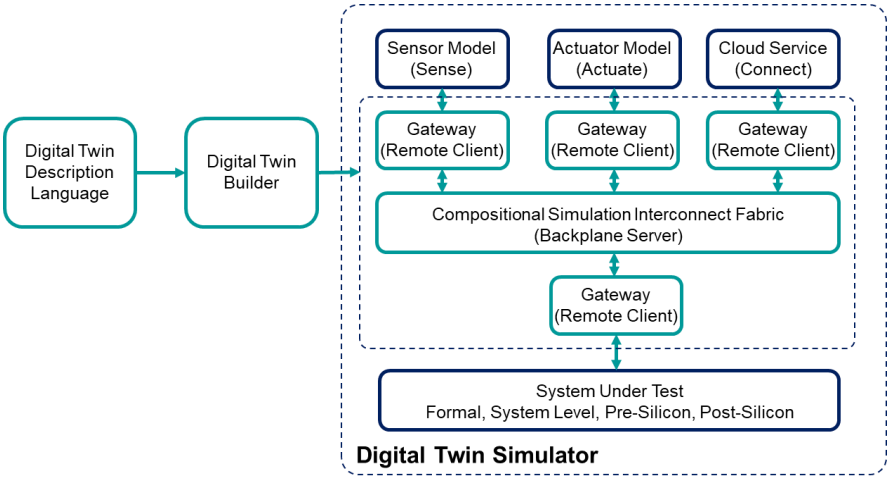

Figure 2. Digital Twin Interconnect Fabric Framework.

B. Digital Twin Description Language

This is an intermediate & proprietary language format upon which system modelling languages can be built on top. It defines the connectivity between different digital twin components. The syntax and semantics of this language are beyond the scope of this paper due to space limitations. However, the concept can be illustrated with the example in Figure 3, which will be discussed in Section III.

C. The Automation Flow

This flow mainly eases the generation of the infrastructure of the backplane server and remote clients' gateways needed to connect the heterogeneous components of a Cyber-Physical System (CPS) including its compute or think, sense, actuate and connect elements. The user design entry is the DTDL to describe the architecture of the digital twin and a builder that parses the language and generates the infrastructure needed to connect the different models and tools, then again comes the role of the user to insert his/her System Under Test (SUT) or Design Under Test (DUT) and kick-off the operation of entire compositional simulation using the digital twin simulator managing the simulation time with a unified timing engine while allowing the visualization of the Gateways' interconnect signals.

- *Digital Twin Builder:* Developing a user's own digital twin configuration requires a relatively high level of knowledge and expertise in many areas and a considerable amount of development time. Digital Twin Builder provides an automated flow where users can create the skeleton of their desired digital twin configuration easily.
- *Digital Twin Simulator:* The simulator provides an interface for the user to be able to control the simulation time and display information about the digital twin during the simulation. An X-Term window pops up for each of the DT components and a command prompt interface for "Digital twin Simulation Control" provides full control over the time advancement and can display some useful information about the system.

III. CASE STUDY

In this section, we demonstrate a comprehensive three-client automotive digital twin to showcase the SDV methodology for validating an ADAS algorithm, enabled through our compositional simulation framework. This example exhibits the framework's adaptability to deploy an ADAS algorithm from high-level SW in (A) to Pre-silicon in (B), abstracting all complexities pertaining to connecting and synchronizing several heterogeneous components in a cycle-locked deterministic simulation, requiring substantial expertise across various domains.

To create the SDV-based digital twin, we used the DTDL depicted in Figure 3 to define three ports with appropriate gateway types. This included configuring the ports by specifying their signals and directions and setting up data exchange based on the protocol used, which in our case is the CAN protocol.

Figure 4 shows the digital twin components and their connections. The used digital twin components are i) Simcenter Prescan: A scenario simulator used for high-fidelity sensor modelling in realistic traffic scenarios to

assess the behaviour of 'smart cars' under different hypothetical scenarios, conditions, and unexpected disruptions [11]. ii) Simcenter Amesim SimRod FMU: A model for representing the mechanical dynamics of the vehicle operating in the scenario simulator [12]. iii) SDV client: This client deploys an Adaptive Cruise Control algorithm (ACC).

```
add port -type C++ -gateway c++2DtCan -name controlAlgorithm
add port -type Fmu -gateway fmu2DtCan -fmuPath <fmu path> -name egoVehicle
add port -type C++ -gateway c++2DtCan -name scenario
add port -type Rtl -gateway rtlCan2DtCan -name scenario

define portSignals -portName controlAlgorithm -signals
"[distanceToTarget:double:input,egoVelocity:double:input,throttle:double:output,b
rake:double:output]"
.
. (defining Port Signals of the remaining ports)
.
define frame -portName controlAlgorithm -canId 10 -signals
[distanceToTarget:0:64]
define frame -portName scenario -canId 10 -signals [distanceToTarget:0:64]
.
. (defining frames of the remaining signals)
.
set digitalTwinName sdvDigitalTwin
set simulationStep 0.1 s
set totalSimTime 100 s
```

Figure 3. Digital Twin Description Language for SDV-based digital twin.

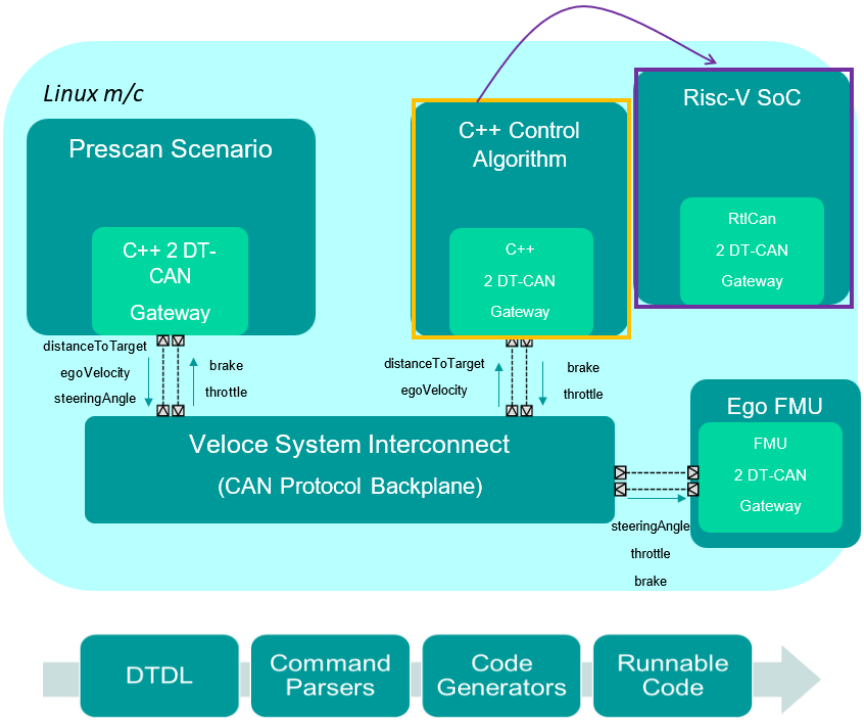

Figure 4. The comprehensive SDV-based digital twin and its automation flow.

To validate the control algorithm, the vehicle operating in the scenario simulator is equipped with a speed sensor and a proximity sensor to provide the necessary data to the ACC client. The ACC client performs processing to determine the appropriate brake and throttle values needed to maintain a safe distance when the vehicle approaches its target.

Table 1: The average time taken to execute packet processing per each client in the SIL case.

Time of C++ Client (ms)	Time of Vehicle Interface (ms)	Time of Prescan Client (ms)
8.6	1.3	49.5

A. SDV software implementation (SIL): The ACC Control *a*lgorithm *was* written in a C++ high-level language. This client is connected through an appropriate gateway featuring a C++ interface, highlighted by the yellow rectangle in Figure 4. This interface was created using its corresponding DTDL language, also marked by a yellow rectangle in Figure 3. By going through the creation steps, the digital twin is successfully generated and ready to run the simulation through the framework's digital twin simulator, as depicted in Figure 5.

Table 1 summarizes the average time taken to receive a packet, process the logic, and send back the result for each client connected to the framework over a total simulation time of 40 seconds. The results indicate that the bottleneck exists in the Prescan simulator, as it requires more processing time than the other clients.

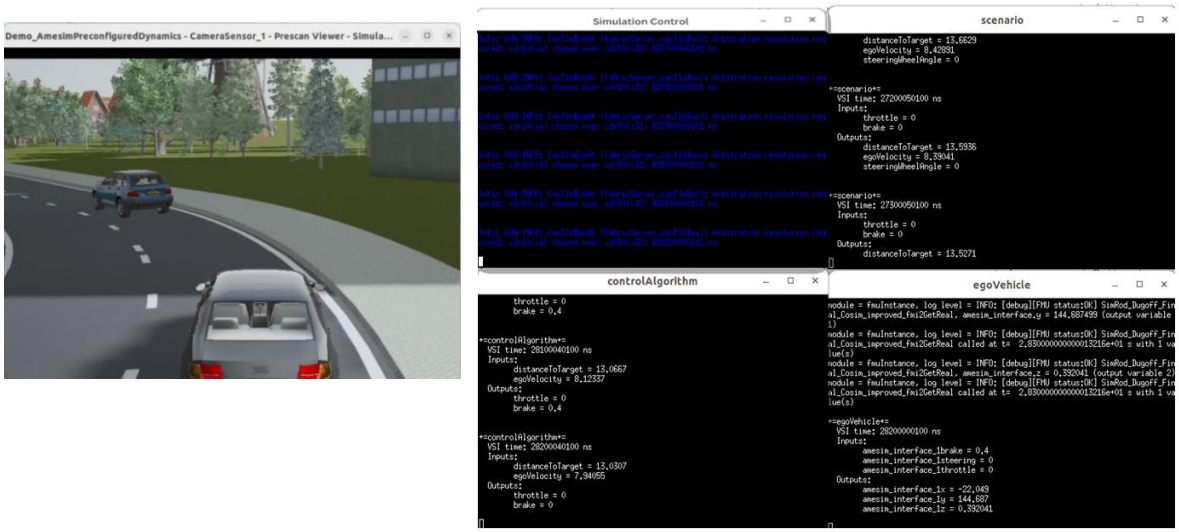

Figure 5. X-Term Windows show Digital Twin Simulation Control and Digital Twin clients exchanging signals through the CAN backplane, along with Prescan Viewer.

B. Deploying to Pre-silicon Platforms: The ACC algorithm is then seamlessly deployed to a proFPGA Quad motherboard through a simple modification in the Digital Twin Build Commands written in DTDL, highlighted by a purple rectangle in Figure 3 and reflected in its corresponding gateway in Figure 4. Notably, the connections of the other clients remain unchanged, with modifications made only to the part related to the ACC client, as shown in Figures 3 and 4. This approach significantly reduces the time required to transfer the SDV algorithm from high-level design to deployment on the pre-silicon platform.

Table 2 provides a summary of the average duration needed to receive a packet, execute the logic, and return the result for each client connected to the framework over a 95-second simulation period. In the case of running SDV on the edge, the simulation duration is extended due to network latency, as each client resides on a different host. The simulation step for both experiments (A) and (B) is 0.00085 seconds, as defined by the user in DTDL. This means the simulation progresses by 0.00085 seconds in each step, ensuring that all components complete their processing. This simulation step is acceptable for the SimRod FMU component, as it does not perform calculations in our case. However, the simulation step could be increased to accommodate the slowest component in the system.

Table 2: The average time taken to execute packet processing per each client, when verification of the SDV algorithm is moved to run on edge.

Time of SDV on Edge (ms)	Time of Vehicle Interface (ms)	Time of Prescan Client (ms)
0.23	4.4	75.4

Table 3 shows the statistical results of the ACC client on proFPGA. The DUT (RISC-V) is moderately complex with a significant amount of logic and storage elements utilized and reasonable hardware time taken with respect to the total time simulation. The hardware time is calculated by multiplying the number of clock cycles taken to run the simulation by the duration of one clock cycle. It is important to note that the number of clock cycles is relatively high in the simulation, even though no packet processing occurs during this time. Therefore,

hardware time is not an accurate indicator of the program's performance on the edge. Instead, we had to analyze the time taken for processing a single packet for each client as illustrated in Table 2. The software time is the total time the software clients take to execute all needed transactions during the whole running scenario of the SDV case study. The communication time refers to the duration required for interactions between the software and hardware components. In our case, the frequency is limited to 20 MHz and cannot exceed this frequency due to the constraints of the Co-model channel, which is used for communication between the SW and HW DUT.

Table 3. Statistical results of running SDV algorithm on edge ECU (proFPGA).

Capacity	No. of FPGAs	Hardware time (sec)	Software Time (sec)	Communication Time (sec)	Frequency (MHz)
LUTs: 69684 Registers: 50565 RPMs: 21 RPDs: 8	1	62.7	188.2	9.6	20

Through the demonstrated use case, we have presented the compositional simulation framework's versatility and robustness for testing SDVs by seamlessly integrating various components and deploying high-level software to Pre-Silicon platforms (proFPGA).

IV. CONCLUSION

Our compositional simulation interconnect acts as a versatile and scalable platform for SDV validation, by its agility in integrating various models and simulators seamlessly through an automated flow. It provides a comprehensive ecosystem for testing the safety and efficacy of automotive applications. The provided use case and results provide a practical application of the framework, while the performance metrics could be assessed on different platforms to ensure optimal configuration prior to ECU fabrication. In our future work, we will extend our study to include the implementation of SDV algorithms on Virtualized SoC Platforms on the cloud with extended digital twin simulation capabilities.

REFERENCES

[1] Liu, Z., Zhang, W. & Zhao, F. Impact, Challenges and Prospect of Software-Defined Vehicles. Automot. Innov. 5, 180–194 (2022).

[2] Michael Grieves. Origins of the digital twin concept. Florida Institute of Technology, 8, 2016.

[3] David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. Characterising the digital twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology, 29:36–52, 2020.

[4] Siemens Veloce CS System. Available at: https://eda.sw.siemens.com/en-US/ic/veloce/

[5] ProFPGA Available at: FPGA Prototyping | FPGA based Prototyping | FPGA Board Multi-FPGA (profpga.com)

[6] Hiroyuki Matsumoto, and Philip Neubauer.Vehicle Servers/High Performance ECUs. IEEE Intelligent Vehicles Symposium,2017.

[7] Siemens EDA Pave360 backplane. Available at: https://resources.sw.siemens.com/en-US/fact-sheet-pave360-backplan.

[8] Yuanfu Li, Jinwei Chen, Zhenchao Hu, Huisheng Zhang, Jinzhi Lu, and Dimitris Kiritsis. Co-simulation of complex engineered systems enabled by a cognitive twin architecture. IJPR, 2021.

[9] G. Frank, Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems, Springer (2005).

[10] T. Blockwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Jung-hanns, J. Mauss, D. Neumerkel, et al., Functional mockup interface 2.0: The standard for tool independent exchange of simulation models, in: Proceedings of the 9th International Modelica Conference, The Modelica Association, 2012.

[11] Siemens Simcenter PreScan. Available at: https://www.plm.automation.siemens.com/global/en/products/simulation-test/active-safety-system-simulation.html.

[12] Siemens Simcenter Amesim. Available at: https://www.plm.automation.siemens.com/en/products/lms/imagine-lab/amesim/.