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

Automated Flow to Maintaining Consistency in Parallel Design Representations Using Cross-Level Verification

Javier Castillo, Mohammad Badawi, Jan-Hendrik Oetjens, Robert Bosch GmbH, Reutlingen, GERMANY

***Abstract*— Complex automotive ASICs and SoCs require rigorous verification and testing, which can be both time-consuming and costly. To accelerate this process, modern ASIC and SoC development flows utilize parallel design representations at different levels of abstraction, commonly virtual prototypes (VPs) alongside RTL and mixed-signal design. However, failing to maintain consistency among design representations can undermine the advantages of utilizing them in parallel, compromising quality and delaying time-to-market. This paper addresses the critical issue of ensuring consistency among system specifications, mixed-signal design representation, VPs, and software throughout ASIC and SoC development. Our approach focuses on applying the same use case scenarios across all design representations, which in turn enhances the verification of digital and mixed-signal designs using the real-world scenarios themselves and allows for the validation of overall system functionality when hardware and software designs operate together. We propose an automated method for use case mapping between abstraction levels (VP to RTL and vice versa) relying on signal traces recorded in VCD file besides small set of refinement information to facilitate essential timing adjustments and manage signal interdependencies. To demonstrate the effectiveness of our approach, we conducted a case study to verify a production automotive ASIC design, reporting quantitative measures of accuracy and effort reduction, as well as qualitative findings that enhanced verification strategy at early stage of the project.**

Keywords—SystemC; Virtual Prototyping; HW/SW Co-Verification; Testcase generation; model Consistency

I. INTRODUCTION

Modern automotive Application Specific Integrated Circuits (ASICs) and Systems on Chip (SoCs) are becoming increasingly complex, thus requiring intensive design verification as well as on-chip and off-chip software (SW) testing to satisfy quality standards and avoid the costly design defects discovered in the later stages of the project. Performing such verification and testing is fundamentally effort- and time-consuming, leading to increased project costs, extended project duration, and, more importantly, longer Time-to-Market (TTM). To mitigate the impact of prolonged verification and testing, sophisticated development flows leverage parallel design representations at different levels of abstraction, commonly using Virtual Prototypes (VP) alongside Register-Transfer Level (RTL) and mixed-signal design representations to shift left software development and testing, as well as block-level and top-level design verification. However, utilizing design representations at different levels of abstraction in parallel presents a key challenge: ensuring consistency between the different representations and the reference specification during development. Unless this challenge is effectively addressed, the anticipated gains in quality assurance and reductions in TTM are not achievable.

This paper focuses on consistency between system specification, RTL and mixed-signal, VP and SW during project development time, allowing to ensure: 1) equivalence between VP used during early SW development on the one hand and RTL and mixed-signal design representation on the other hand, 2) SW realization is based on correctly interpreted specification, 3) software operability, meaning that the SW that have been tested using the VP will perform faultlessly on the produced chip, 4) verifying mixed-signal design representation for real-field scenarios using production SW, filling possible gaps in verification strategy that is based on simulation and formal verification and zooming the focus further towards critical scenarios of system-level use cases. Without leveraging such SW-based test cases, specifying and implementing constrained-random and formal verification test cases to verify mixed-signal design and consequently validate overall system becomes intractable as it requires expertise across all software layers as well as system-application scenarios and is often hindered by barriers related to information sharing due to confidentiality concerns.

Our proposed approach, which also targets production SW use cases, performs automatic mapping and generation of use cases from a higher level of abstraction to a lower level or vice versa. In other words, enabling to map test cases executed by software using the SystemC-based VP to SystemVerilog (SV) test cases that can be

used to verify RTL and mixed-signal design in Unified Verification Methodology (UVM) environment or mapping RTL and mixed-signal verification tests cases to SystemC functional test cases. The generation of test cases mainly depends on activity traces stored in a Value Change Dump (VCD) file, which is a widely used format allowing flexibility in our approach. Due to varying abstraction levels, additional information is needed alongside the VCD file and can be provided in configuration files to define details like language of the target test, signal mapping and most importantly application of timing adjustments as well as deducing timing decisions for the signals that have interdependency. As a result, test cases can be rapidly generated to reduce testing and verification effort. More important, the use of real-field use-case enabled collecting valuable feedback to enhance mixed-signal design verification strategy at early stage of the project; besides, improving communication between designers, verifiers and software engineers during validation and troubleshooting.

II. Related Work

The Portable Test and Stimulus Standard (PSS) provides a framework for creating unified test scenario representations usable across various integration levels and execution platforms [1]. Our approach maps real-production software use cases, developed by different organizations, to the appropriate abstraction level for design verification, ensuring consistency between design representations and overcoming inter-organizational confidentiality issues.

Choi et al. [2] proposed a HW/SW co-verification method using SystemC-RTL co-simulation with Inter-Process Communication (IPC) for early software development. Their approach enabled SW simulation to interact with a HW emulator or Field Programmable Gate Array (FPGA). However, it required RTL for SW testing and had limited HW virtual components. Our approach avoids multiple processes and directly maps test cases to the target abstraction level, hence allowing cross-level verification using tests developed at different abstraction levels.

Ehrlich et al. proposed in [3] using hardware-in-the-loop (HiL) simulations for accelerated co-verification, leveraging a UVM SystemC environment to test various DUT implementations including a complete SystemC DUT, a mixed-abstraction DUT, an FPGA DUT, and an ASIC DUT. The proposed setup includes a driver and a monitor for interfacing with the DUTs, allowing the reuse of test cases across all implementations. If all tests pass, the DUTs are deemed consistent. However, the HiL tester must have a compatible HW interface (e.g., GPIOs, SPIs, ADCs), high-performance processing to run real-time and predictable timing for accurate DUT stimulus generation. Our approach also addresses consistency between different DUTs, besides it targets early phases of the project, where RTL is still under development.

Teo Vallone et al. presented an interesting flow [4] for the automated generation of SystemC models for AMS designs. Their approach automatically generated a top-level using a netlist as input; however, it required users to manually specify test cases, either directly in code or using a provided GUI that could generate code for the specified tests, which were then used for co-simulation. In contrast, our flow targets early verification, even when RTL is unavailable for netlist generation. More importantly, our flow automatically generates the testbench and maps complex, real-world test cases that are used for SW development and concept validation.

Matteis et al. presented in [5] a scalable Verification Intellectual Property (VIP) to enhance co-verification in ASICs by translating firmware events into UVM transactions. They utilized scoreboards and checkers to bridge the gap between verification and firmware development. Our approach aims at bridging the gap between SW (or firmware), design and verification without restricting the level of abstraction to be used.

III. Automated Cross-Level Verification Flow

We propose a methodology, illustrated in Figure 1, that ensures consistency between the intended features of the ASIC implemented in the VP and those represented in the RTL and mixed-signal designs. Additionally, this approach guarantees that software developed and tested using the VP can faultlessly operate on the produced chip. Performing cross-level verification through this methodology, leads to identify potential issues early in the project, including: 1) ambiguities in specifications and possible misinterpretations, 2) defects in the VP itself; although it serves as a reference model, it is not inherently immune to errors, and 3) defects in the RTL and mixed-signal

design representations. As shown in Figure 1, we achieve such consistency and operability assurance by reusing test cases across different levels of abstraction through automatic generation of test cases for a given target language. In the scenario depicted in Figure 1, use cases executed by the software using the VP are mapped to SystemVerilog test cases that can be utilized to verify the RTL and mixed-signal design representations. To facilitate this process, the VP must be instrumented with trace collectors that capture and store traces of all ASIC signal activity in a VCD file. It is important to note that trace collectors are commonly used in VPs, and their implementation requires minimal effort. While a VCD file is generated for all signals, some details may be obscured since VP operates at a higher level of abstraction; therefore, additional refinement of information is necessary as input besides the VCD trace file. Next, we will outline the test-case generation flow and provide further details on signal and timing alignments.

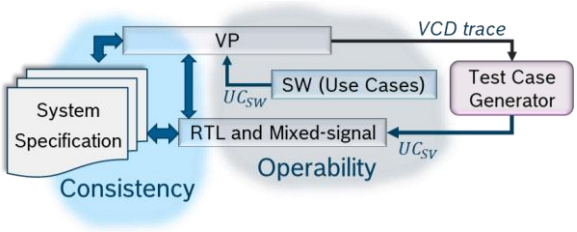

Figure 1: Overview of the Proposed Automated Cross-Level Verification

A. Test-Generation

The test case generator is a Python-based tool that maps test cases across different levels of abstraction. The test case generation flow, illustrated in Figure 2, consists of four stages. While the generation primarily relies on a VCD trace file, additional input files containing refinement information facilitate signal mapping and timing alignment, thereby enhancing the generator's flexibility and usability across multiple projects. These input files are provided in simple JSON format to ensure uniformity and reuse across test cases, moreover they can be fully or partially reused across project variations. The four stages are detailed as follows:

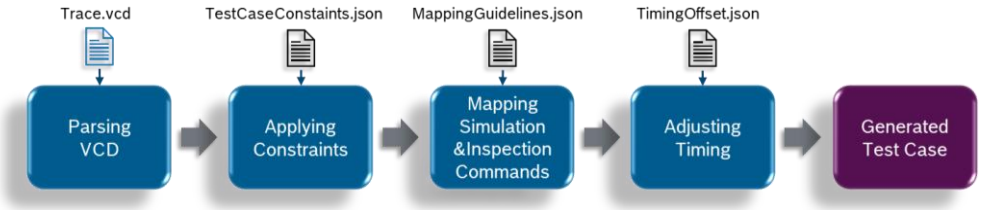

Figure 2: Structure of Test Case Generation Flow

1) VCD Parsing

The first stage will read the VCD trace file generated by the VP. This contains the traces of all the inputs and outputs of the ASIC, these can be Boolean signals, double values, as well as busses. The VCD is standard format and is widely supported by various tools [6], which provides flexibility for mapping across levels of abstraction.

2) Applying Constraints

This stage requires the information about the list of all the supported stimulation and inspection commands that can be generated for the testcase, as well as their timing constraints. Such information is provided in an input file as denoted by “Testcase_constraints.json” in Figure 2. While there are no timing limitations when modelling communication protocols within a VP, the case for mixed-signal design demands conforming to timing constraints. For example, SPI transactions modeled with Transaction-Level Modeling (TLM) are instantaneous and in some cases do not consume simulation time whereas in mixed-signal design, they do. These timing constraints must be provided to our generator to validate the VCD. In case any timing violations are detected, the test case generation is cancelled, and an error is raised; consequently, ensuring the validity of the generated test case.

3) Mapping Stimulation & Inspection Commands

This stage utilizes the list of constrained stimulation and inspection commands obtained in the previous stage and maps each signal of the VCD file to a specific command. The mapping is then further refined with the help of

the information about relevant ASIC signals, which can be provided in a file as denoted by "Mapping_guidelines.json" in Figure 2.

4) Time Adjustment

This stage is crucial for achieving accuracy, besides facilitating test generation flexibility. It performs necessary timing modification to the behavior of each stimulation and inspection command associated with a signal and resolves timing interdependency between signals. Timing adjustments in this stage depends on timing offset information, which can be supplied as an input file, as denoted by "Timing_offsets.json" in Figure 2. Further elaboration for timing adjustment and resolving interdependencies between signals is provided in III.B.

5) Generated Test Case

The output of the previous stages is a test-case code generated according to user directives to stimulate the inputs of ASIC design representation and verify/check its outputs. The consistency between the VP and the mixed-signal design representation regarding the feature under study is verified using this test case. If the generated test case successfully passes, we can state that mixed signal design representation and VP are functionally equivalent; otherwise, we take advantage of this rapid generation flow to generate adapted test cases for troubleshooting.

B. Timing Adjustments Between Levels of Abstraction

Our approach is applicable to mixed-signal ASICs; in other words, it can handle both digital and analog signals; this includes analog voltages, digital inputs/outputs as well as bus interfaces, e.g. SPI, PSI, UART, etc. However, due to the varying levels of abstraction between the VP and mixed-signal design representations, slight temporal offsets may be observed for signals at each level of abstraction within the design. Since the VP represents the design at a higher level of abstraction, the detailed behavior of analog signals may be abstracted into discrete values. In many cases, the ramp-up time of analog signals is not modeled, leading to slight differences in timing observed between the same signal in the VP and in the RTL and mixed-signal design representations. Additionally, our code generation flow effectively manages signal interdependencies, including circular dependencies where one signal relies on another in a sequence, with the first signal relying on the last. If these circular dependencies are not detected and resolved, generating valid test cases becomes impossible. Next, we will explain how to align signal temporal offsets and how to detect and resolve circular dependencies.

1) Aligning Signal Temporal-Offset

To ease illustration, consider an example of a mixed-signal design that includes an analog signal, V_a and a digital signal S_a, where a loosely-timed VP modeled in SystemC, along with SystemVerilog for RTL and mixed-signal simulation are utilized. Figure 3 (left) shows the behavior of the analog signal V_a in both the VP and the mixed-signal design representation. In the VP, we observe that the analog signal reaches its new stable voltage immediately, while in the mixed-signal design representation, the signal exhibits a mandatory ramp-up time before V_a reaches its target voltage. Figure 3: Comparison of Behavior of Analog (left) and Digital (Right) Signals in VP vs RTL and Mixed-signal Design. (right) demonstrates the behavior of a digital signal S_a in the VP and the mixed-signal design representation. In VP, where clock behavior is abstracted to gain simulation performance, a register is updated with the value of S_a immediately while in mixed-signal design representation, the register is updated on the next active edge of the clock; quantitively, with a clock of $1MHz$, this delay would be $1\mu s$.

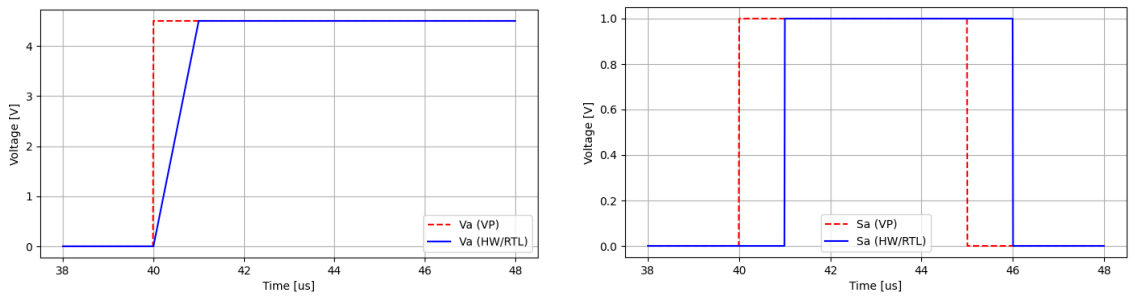

Figure 3: Comparison of Behavior of Analog (left) and Digital (Right) Signals in VP vs RTL and Mixed-signal Design.

The timing differences between the VP and the mixed-signal design representation must be considered when generating test cases. Both examples mentioned earlier require the injection of timing delays during test case generation, to appropriately postpone checkers to right points in time. To address these scenarios, the generator utilizes information from the "Mapping_guidelines.json" file, as shown in Figure 2, and automatically injects offsets for the various test-case functions as needed. We refer to this as temporal-offset alignment of signals, and it adjusts the timing of stimulation and inspection commands within the generated test cases, ensuring their applicability at the target level of abstraction.

Temporal-offset alignment offers greater flexibility for more complex scenarios. For example, as illustrated in Figure 4, consider a mixed-signal design that includes three analog signals, V_a, V_b and V_c, which represent voltage levels that must ramp up and stabilize by the rising edge of the digital S_a. In such case, aligning the temporal offsets of the voltage signals individually requires the user to know the expected delays for all voltages and provide this information to the generator, leading to increased manual effort and a higher risk of human error. To address this issue and leverage test generation efficiency through automation, we use an event-based temporal-offset alignment method to position stimulation and inspection commands at the appropriate time.

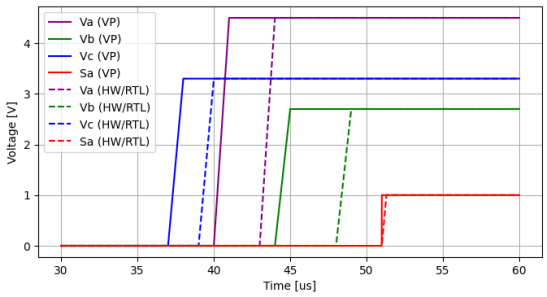

Figure 4: Comparison Between Analog Voltage Ramp-up in VP and Mixed-signal Design.

In event-based temporal-offset alignment, the user only needs to specify the generator the event to wait for when inserting stimulation and inspection commands. Such information needs to be provided in the "Mapping_guidelines.json" file that is shown in Figure 2. The generator then inserts the commands at the appropriate time when the event occurs. To illustrate the result of event-based temporal-offset alignment on the generated test case code, Figure 5 (left) shows the SystemVerilog code produced without temporal-offset alignment, where the inspection command "CHECK" for voltage signal V_a is placed before the rising edge of digital signal S_a. As a result, the generated test case will fail to detect changes in V_a just before the rising edge of S_a, resulting in a false pass. In contrast, the code generated with event-based temporal-offset alignment, shown in Figure 5 (right), ensures that voltage signal V_a maintains its stable value until the rising edge of S_a, thereby enhancing test quality and preventing false positives.

```
#10us // time=152us
CHECK(Va === 3.3);
#10us // time=162us
// ... // various events&checks
#10us
CHECK(Sa === 1); // rising edge Sa, time=346us
```

```
44  #20us // time=162us
45  // ... // various events&checks
46  #10us
47  CHECK(Sa === 1); // rising edge Sa, time=346us
48  CHECK(Va === 3.3); // Va checked on rising edge of Sa, time=346us
49
```

Figure 5: Generated Test Case in SystemVerilog without Temporal-offset Alignment (left) and with Temporal-offset Alignment (right)

2) Detecting and Resolving Circular Dependency

In our approach, interdependency between signals is modeled as a directed graph that consists of vertices and directed edges connecting them. We represent signals requiring temporal-offset alignment by vertices and interdependencies between these signals by the edges, such that the direction defined by the preceding and subsequent signals. For instance, in our previous example, the analog signal V_a and the digital signal S_a illustrate this concept: all stimulation and inspection commands for V_a are aligned with the next rising edge of S_a, indicating that V_a depends on S_a. This relationship can be represented as two vertices, V_a and S_a, with a directed edge from V_a to S_a. The same representation applies to V_b and V_c, resulting in the directed graph shown in Figure 6 (left).

However, in the more complex scenario shown in Figure 6 (right), all stimulation and inspection commands for the analog signal V_a are aligned with the next rising edge of the digital signal S_a. In this case, S_a depends on S_b, S_b depends on S_c, and S_c, in turn, depends on S_a. This circular dependency makes determining a stable value for S_a a moving target, preventing temporal-offset alignment and successful test case generation. To address this issue, we enhanced the test code generator with a mechanism based on the directed graph model to detect and resolve circular dependencies, such that it employs Depth First Search (DFS) to traverse the directed graph while tracking visited vertices. While traversing the directed graph, an identifier for each vertex that has been visited is added to the "visited" array, and if the identifier already exists in the array, then it means the vertex has been already visited indicating a circular loop within the directed graph. When the generator detects circular dependency in the use case, given the input VCD and other JSON files, it iterates a finite number of times to find a stable solution for all signals involved in the offset alignment. The maximum number of iterations is specified by the user in the "Mapping_guidelines.json" file that is input to the generator. If this limit is reached, the generator raises an error, indicating that test case generation has failed due to the circular dependency. This allows users to review and ensure that all input files (the VCD file and all other JSON files) are compatible and complementary, leading to eliminate the circular dependency. Note that users can also instruct the generator not to iterate by setting the iteration limit to 0 in the "Mapping_guidelines.json," which forces the generator to report an error immediately upon detecting circular dependency.

Figure 6: Directed Graph Examples without Circular Dependency (left) and with Circular Dependency (right).

IV. CASE STUDY

To demonstrate the applicability and effectiveness of our approach, we employed our cross-level verification flow to verify a production automotive ASIC design. The ASIC is designed to operate within an Electronic Control Unit (ECU), where it reads sensor and voltage data and communicates with an on-board microcontroller. As illustrated in Figure 7, we performed the cross-level verification in our case study in three steps:

1. The VP was integrated into the ECU model, and a production software use case was executed on the microcontroller. The microcontroller sent requests and received responses from the VP that in turn processed the data communicated with external sensor and voltage models. Consequently, the SW use case was validated, and the traces of all the inputs and outputs of the ASIC were recorded in a VCD file.
2. The VCD trace file recorded in step 1, along with configuration and refinement files, were input into the test case generator, which produced a corresponding SystemVerilog test case.
3. The generated SystemVerilog test case was imported into the UVM testbench to verify the mixed-signal design representation that will be manufactured as ASIC chip. Therefore, we could conduct the verification using the exact real-life scenario used in the final product.

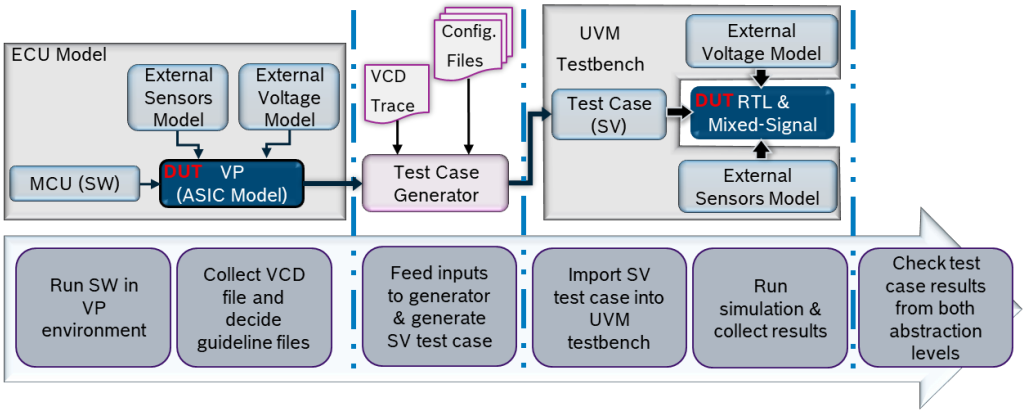

Figure 7: Employing Automated Cross-level Verification Flow to Verify the Production ASIC.

By employing our cross-level verification approach in this case study, we identified two different discrepancies in the mixed-signal design at a very early stage and reported them to the verification team for further investigation. We also detected and resolved one discrepancy in the VP. Furthermore, we fulfilled the software team's request for validating SW use cases and providing feedback on its behavior.

A. Accuracy: Waveform Comparison for VP vs Mixed-signal Design

Figure 8 demonstrates the accuracy of signal timing achieved in the generated SystemVerilog. As shown, the digital signal S_a (note that signal and test names have been modified for confidentiality) reaches its stable falling edge at time $t = 24.430ms$ when the SW test case is executed using the VP, while it reaches this stable value at time $t = 24.438ms$ in the UVM SystemVerilog test case. This difference in falling edge time arises from the varying levels of abstraction between the VP and the mixed-signal design representation. As illustrated in subsection III.B and Figure 3, rising and falling times are not considered in the VP; thus, if the UVM checker is inserted earlier than necessary, the check falsely fail due to an invalid test case. The temporal-offset alignment addressed this issue and postponed the inspection command "CHECK" to a later time ($t = 24.440ms$ instead of $t = 24.430ms$), hence ensured a stable signal value and led to generate a valid test case.

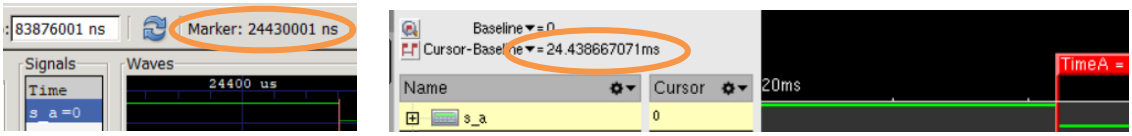

Figure 8: Difference in Timing Observed in VP (left) and Mixed-signal design Representation (right).

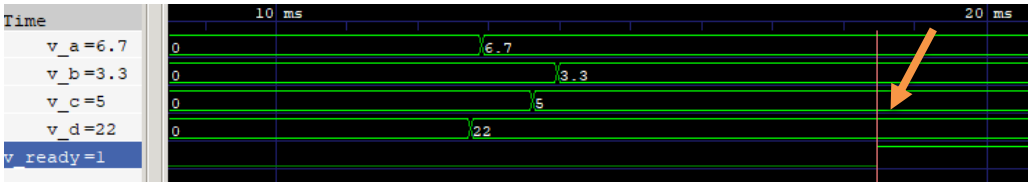

Figure 9 Waveform of Ramp-up within the VP.

Figure 10 Waveform of Ramp-up within the Mixed-signal Design Representation.

```
#100us // time=12.50ms
CHECK(Vd === 22);
#240us // time=12.74ms
CHECK(Va === 6.7);
#730us // time=13.47ms
CHECK(Vc === 5);
#200us // time=13.67ms
CHECK(Vb === 3.3);
#5000us // time=18.67ms
CHECK(Vready === 1);
```

```
71  #5000us // time=18.67ms
72  CHECK(Vready === 1);
73  CHECK(Va === 6.7);
74  CHECK(Vb === 3.3);
75  CHECK(Vc === 5);
76  CHECK(Vd === 22);
77
78
79
```

Figure 11 Generated SystemVerilog Test Case with Temporal-offset Alignment (left) and an Event-based offset (right).

Figure 9 and Figure 10 illustrate the differences in initialization times of voltage regulators between the mixed-signal design and the abstract VP model. According to design used in this case study, initialization time varies due

to runtime factors such as temperature and load when the ASIC chip in operation, meaning that the exact initialization time may not be critical for the test case; however, the correct sequence of stimulation and inspection commands is essential. As shown in Figure 9 and Figure 10, all voltages must be stable at the rising edge of the V_{ready} signal in the VP model and its equivalent S_a in the mixed-signal design (signal names modified for confidentiality). Therefore, temporal-offset alignment was performed, hence postponing voltage checkers in the generated System-Verilog code to ensure stable rising edge of V_{ready}. A snapshot from generated code of this test case is shown in Figure 11, illustrating the difference in the order and time of "CHECK" temporal-offset alignment.

B. Test Case Generation Speed and Reduction in Effort

In this case study, we examined five different software use cases executed on the VP and utilized our test case generation flow to create the corresponding SystemVerilog test cases, each containing 200 to 500 lines of code. We evaluated code generation time on a personal laptop with an Intel i5 Core processor, finding that the average time to generate a single test case is $50ms$. In contrast, manually developing these SystemVerilog test cases would have taken 3 to 7 days due to significant challenges. These challenges might even make the manual development of test cases impossible as they include the need for expertise across software layers—not just the application layer but also lower layers like drivers and boot loading—and obstacles related to information sharing due to the confidentiality of the software layers. It is important to note that the effort required to develop the lightweight Python-based test case generator (consisting of fewer than 1,200 lines of code) is a one-time investment, equivalent to the time that would be spent on manual development a very few simple test cases. This generator can be reused across projects and the input files containing refinement information is a one-time development task per project; nevertheless, these files can be fully or partially reused for different project variations.

V. SUMMARY

This paper introduces a method to ensure consistency across parallel design representations at various levels of abstraction through cross-level verification. Moreover, the rapid mapping of production SW use cases enabled verifying mixed-signal design for real-field scenarios, overcoming the intractable task of specifying and implementing equivalent test cases to verify mixed-signal design with using simulation and formal verification. Utilizing our automatic generator, which features a compact codebase and a minimal memory footprint, rapid mapping of use cases can be achieved, eliminating unnecessary engineering efforts. The conducted case study demonstrated the applicability and accuracy of our proposed approach and its effectiveness in capturing discrepancies at a very early stage. Looking ahead, we aim to extend this approach to facilitate the generation of test cases for the hardware in the lab, thereby accelerating chip bring-up.

VI. ACKNOWLEDGEMENT

The TRISTAN project, nr. 101095947 is supported by Chips Joint Undertaking (CHIPS-JU) and its members Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey and including top-up funding by Federal Ministry of Education and Research, BMBF (Germany)

VII. REFERENCES

[1] https://accellera.org/downloads/standards/portable-stimulus

[2] Choi, J., Kang, K., Lee, B., Park, S., & Im, J. (2021). Early HW/SW Co-Verification Using Virtual Platforms. *18th International SoC Design Conference (ISOCC)*, 1-2.

[3] Ehrlich, P., Nguyen, T., & Voertler, T. (2014). UVM-SystemC based hardware in the loop simulations for accelerated Co-Verification. *Design and Verification Conference and Exhibition.* Munich.

[4] T. Vallone, H. V. Hasou, E. Colizzi, S. Vinco and D. Zoni, "A novel virtual prototyping methodology for timing-accurate simulation of AMS circuits," *2024 25th International Symposium on Quality Electronic Design (ISQED)*, San Francisco, CA, USA, 2024, pp. 1-8.

[5] de Matteis, M., & Barbati, M. (2023). A scalableVIP component to increase robustness of co-verification within an ASIC. *Design and Verification Conference and Exhibition.* Munich.

[6] "IEEE Standard Verilog Hardware Description Language," in *IEEE Std 1364-2001*, vol., no., pp.1-792, 28 Sept. 2001.