2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
10 YEAR ANNIVERSARY

A perfect blend of verification techniques, platforms, and AI-empowered debugging in Ethernet Subsystem Verification

Olivera Stojanovic, Vtool doo, Belgrade, Serbia (oliveras@thevtool.com)

Tijana Misic, Vtool doo, Belgrade, Serbia (tijanam@thevtool.com)

Abstract— **The verification of SoC hardware and software can present numerous challenges, with engineers dealing with the need to balance fast verification results and the inherent complexity of the system. This paper gives an overview of the flow that is followed in the SoC verification of the Ethernet Subsystem. There is a concept of a verification plan with a list of important segments of Ethernet SoC verification and the most suitable platform for the verification of each of them. Our success story tells about Ethernet features tested in formal verification, RTL and GLS simulation, emulation platform, and FPGA. Throughout the process, there is a strong emphasis on promoting the re-usability of testbench parts across various platforms. The paper also explains how an AI approach can improve the debugging process. The results achieved using this flow demonstrate that it can serve as a useful roadmap for other teams facing similar dilemmas in their verification strategies.**

Keywords— Ethernet, SoC verification, emulation, Cogita-PRO, AI

I. INTRODUCTION

This paper presents a true story about the flow we use in System-on-Chip (SoC) digital verification to accomplish efficiency and meet strict deadlines. There is always a question about the selection of tools and verification methodologies we need to ensure a bug-free tape-out. The main two bottlenecks during the SoC verification are the duration of simulation and debugging.

Making smart selections of the test suite and executing some of the tests in emulation may constitute a pivotal factor in obtaining both high-quality as well as fast results. When it comes to assessing performance, use cases, and firmware testing, emulation stands out as an immensely valuable step in the verification process.

Approximately 70% of the resources allocated to the fabrication of a sophisticated System-on-Chip (SoC) are dedicated to the validation process. Within this percentage, it is notable that roughly half, equivalent to 35% of the effort, is spent on debugging [1]. To address this bottleneck, the Artificial Intelligence (AI) debugging platform Cogita-Pro was utilized.

It is imperative for the verification process to be planned and carefully reviewed by the design team, architects, verification, and software team. The crucial part of planning is the selection of the tools and techniques to achieve the best results and meet deadlines. We divided the verification features into sections and decided on the best approach to the verification and debugging of each section. Debugging should be performed in a systematic approach, with maximum usage of the history of debugging by applying AI.

All Intellectual Properties (IPs) developed in-house are verified in IP verification environments which are then re-used at the SoC level following principles of Universal Verification Methodology (UVM). The software is verified in hardware/software (HW/SW) co-verification where specific software is loaded into memory and code is executed by the processor inside the design. Also, it is important to state that in the whole process, third-party Verification IPs (VIP), emulation-ready VIPs, and memory models are used inside the UVM verification environment. The final netlist is released only after all RTL and Gate-Level Simulation (GLS) regressions have passed. During debugging, we used Cogita-PRO, an AI debugging platform, that helped us to share the previous debug knowledge in the team, automatization, finding performance issues in the early stages, and assembling all relevant data that were coming from different sources. We listed the issues we faced during the process and stated

the way to mitigate them. A few directions for further improvement of the process are given in the closing point. Our experience can be a safe route for other teams when planning efficient verification of complex SoC.

II. Related Work

It is important to provide a verification environment that will enable hardware/software co-verification early in the process, producing quick results and preventing possible bugs in the later stages [2]. The robustness of the verification environment, early performance results, and power measurements are the key requirements in front of verification teams [3]. Hardware Emulation is the technique of prototyping real SoC design and accelerating the speed of design execution [4]. The main advantage is that emulation runs hundreds to thousands of times faster than RTL simulation. Limitations of emulation are the following: we are missing some checkers, removing physical models in the emulation netlist, and higher costs on the emulation side. There are good examples of the usage of emulators in power-aware verification [5] and firmware verification [6]. It is important to understand the difference between simulation and emulation. This difference can enable us to make a correct decision about which features are to be verified in simulation and which are to be verified in emulations. The emulation platform is an implementation of the design executed on special-purpose parallel processors. RTL simulators are software packages that simulate expressions written in one of the hardware description languages. This approach could improve the time for SoC bring-up to be measured in hours instead of weeks as was the case before. Also, there are a lot of opportunities to apply AI in the verification process. Different EDA providers are exploring how to apply machine learning (ML) and AI in different areas: ML for coverage closure, bug hunting, regression debugging, AI design analysis, and resource optimization.[7]

III. Methodology

The main motivation of this paper is to present how flexible and scalable solutions that can be used by multiple users can help us to achieve reliable and fast verification results. The most efficient way is to create a single flow that can be reused on different platforms. The use of AI tools has made the debugging process significantly more efficient and effective.

During the verification process we used the following solutions: (1) for RTL and GLS simulations – we used third-party simulator and tool for running regressions, (2) formal verification is done in the formal tool, (3) emulation platforms, (4) FPGA, (5) AI debugging by Cogita-Pro support were also used during the process. In Figure 1, we give a timeline of the usage of different tools on the project during the verification process.

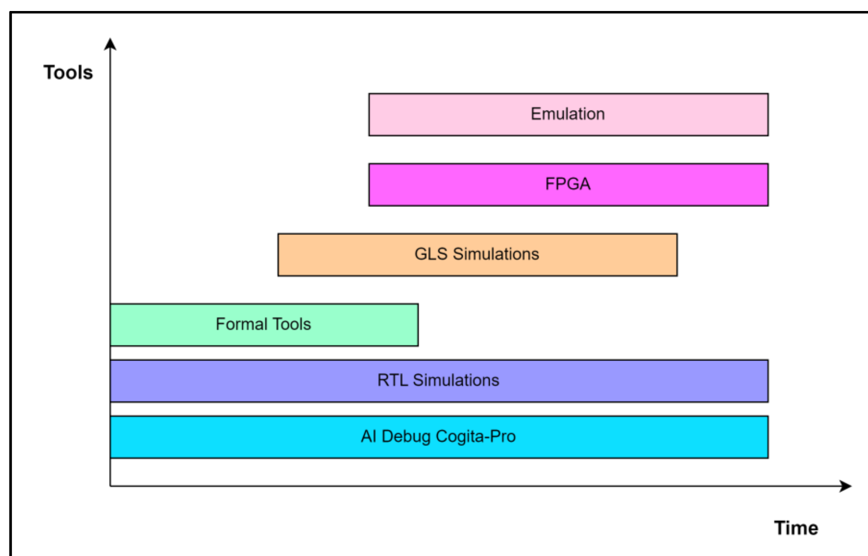

Figure 1 Different platforms and tools are used on the project depending on time

The RTL verification is implemented as a combination of UVM tests and complex tests for HW/SW co-verification. In HW/SW co-verification, we have verification drivers and tests written in C, which are compiled and stored inside memories and the code is executed by the Central Processor Unit (CPU) inside the design. Part of this software integrates production drivers which were done in the early phase of the process (UART, SPI, Flash). The complex tests that cover all data paths and different speeds are selected for running in GLS regression.

For describing the flow, we selected the Ethernet subsystem since it is connected to external ports and its software is complex. Performance results for traffic over the Ethernet interface were extremely important for the customer, due to the significant stress of interconnecting. The Gigabit Ethernet subsystem consists of an Ethernet controller, Physical Coding Sublayer (PCS), and Ethernet Physical Layer (PHY). The verification is implemented in multi-platform environments, which include formal verification, RTL and GLS simulation, emulation on Palladium, and FPGA prototyping. Cogita-PRO helped us to detect bottlenecks on interconnect, understand the quality of the tests, and the history of the debug gained during the project within the team.

Although the multi-platform verification was a challenging task, we managed to reuse things from one platform to another, such as Makefiles, software, and parts of tests. The schematic preview of the emulation testbench is given in Figure 2. Firstly, we developed Makefiles that can accommodate different requests and make them work on all supported platforms. The same code for the bare-metal boot of CPU is used in RTL simulation, FPGA, and on the emulation platform. The code for bare-metal drivers was re-used in the same way as well (e.g., Ethernet drivers and drivers for other peripherals). Consequently, Makefiles for the compilation of bare-metal software code are the same on both platforms.

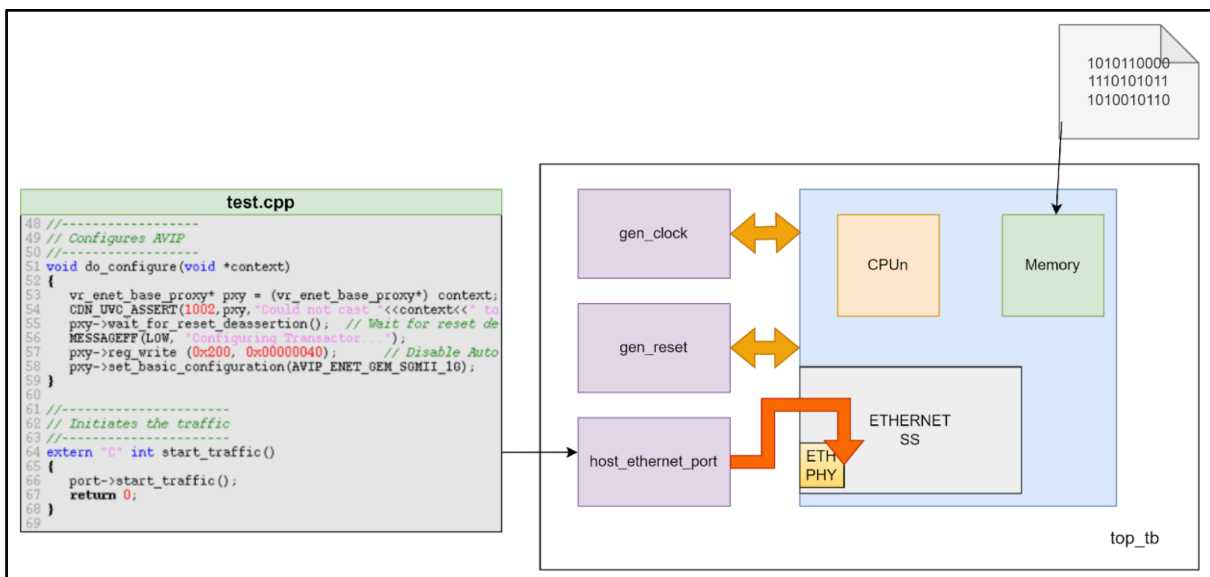

Figure 2 Emulation Testbench

Scripts for running tests and memory preloads from RTL simulation are slightly updated and re-used on the emulation platform. The UVM part of tests used for driving Ethernet VIP in RTL simulation is not re-used on the emulation platform. We developed a new code for Ethernet emulation-ready VIP in C++ to score faster execution on the emulation platform as recommended by provider support [8].

Some changes have been requested to create a design netlist that is run on the emulation platform. All external reset and clock sources necessary for the test are identified and generated in the emulation testbench.

Memories used in RTL simulation are replaced by synthesizable models. Off-chip memories are modeled by flash memory models optimized for emulation. Ethernet PHY is replaced by an empty wrapper since PHY model is not synthesizable. This impacts the emulation-ready VIP connection to be moved to PCS interface instead of SerDes boundary of Ethernet PHY. This is an important difference since in RTL simulation, Ethernet PHY is

present in the design and the full data path is tested from memory over the Ethernet controller, PCS, and PHY to Ethernet VIP. In the emulation platform, we are testing part of this path bypassing Ethernet PHY.

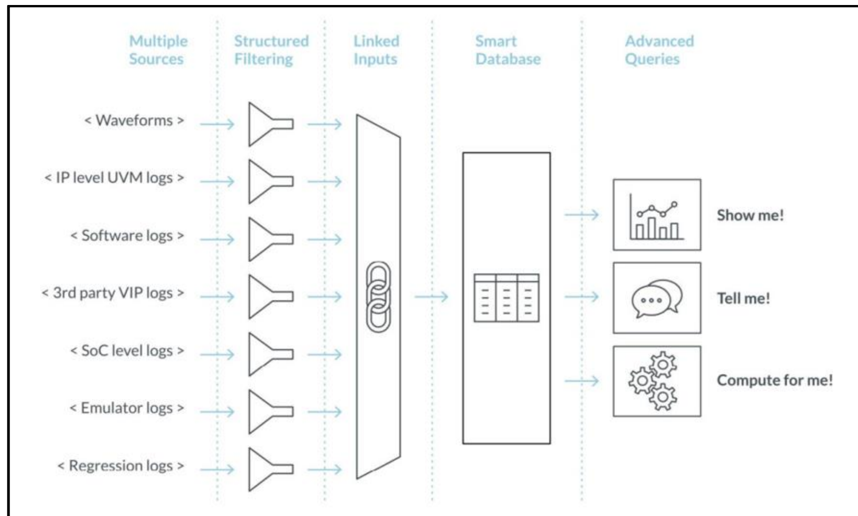

Figure 3 A process of root causing the bug by AI tool

The main challenge during verification is debugging. Debugging on the SoC level requires analyzing big data that is coming from different sources. We used UVM log files, transaction trace tables, logs from third-party VIPs, waveform, and tarmac as presented in Figure 3. With an AI-debugging tool (Cogita-Pro) we were able to use a unique platform for exploring all this data, connecting them, running algorithms, and visualizing any type of information that was relevant for debugging. The infrastructure and database built for AI-tool were completely re-usable between different versions of the chip and on all levels of verification: IP, sub-system, SoC.

IV. AI Tools Improvement of the Debugging Process

A few issues which were detected by using AI algorithms in the early stage of the verification process are presented in the following chapter. During HW/SW co-verification, input that we used for AI analytics were UVM log file, tarmac, disassembly file and waveform database.

A. Issue with HW/SW mechanism

In this case, the debugging process was based on a built-in algorithm inside an AI tool for analyzing events associated for each transaction transmitted over interconnect. The inquiry was based on UVC logs, which contained messages with relevant data for test traffic. After configuration, the AI tool collected a relevant set of data that was based on the collection of start and end times correlated to each transaction. By analyzing relevant data, it was concluded that simulation ended before the last transaction reached the destination. This was leading us to the root cause of the issue inside the HW-SW mechanism that is used for controlling the test execution. After updating the implementation of the HW-SW mechanism the issue was solved.

B. Issue with Throughput

For analyzing throughput, the same input files and set of data were used as in an example with HW/SW mechanism issue. The performance test showed poor throughput results that were below expectations of SoC architects. An AI tool was used for characterization of transfers in a performance test by making a visual preview of the captured burst sizes, lengths, addresses and other relevant data. After this, the root case of low throughput results was spotted fast. The configuration of maximum burst length inside DMAC (Direct Memory Access Controller) was inadequately configured. By reconfiguring maximum burst size the issue was solved and expected throughput was achieved.

C. Anomalies in data

The biggest challenge in HW-SW co-verification is to analyze all sources that we have in this environment. Verification engineer needs to use: source code of SW, disassembly, tarmac, UVM-SV source code, log files from UVM, RTL design source code, waveforms and schematics. Instead of focusing on one particular point in simulation, our approach is to look at simulation as one big data set. AI algorithms that are run over this data can give us information about issues that were out of team thinking scope like capturing software anomalies and unused bandwidths inside interconnect transmission that can be used for performance improvement.

In the example below the analysis was based on UVC logs and tarmac files which were processed in the way that relevant data set for research was extracted from the overall data provided by these sources.

The significant information used as a start point of debugging is based on:

- Transaction address from tarmac file
- Transaction start time extracted from UVC log
- Transaction end time also extracted from UVC log

Cogita-PRO extracted the memory address of all IT (instruction taken) from the tarmac file and visualized it over a timeline as presented in Figure 4. By only looking at visual presentations of addresses no significant conclusion can be made. However, by running AI algorithms over relevant data, the tool recognized patterns and detected anomalies as presented in Figure 5. Inspecting memory address values (instruction taken) over time, AI tools can give us a nice overview of specific, possibly unwanted, events in simulation like unexpected branches, loops, drops in efficiency of algorithm or unexpected interrupt.

In this given case, upon memory address analysis, we noticed that the CPU enters branch code that is distinguished from the regular pattern as shown in Figure 6. In addition to that, the AI algorithm was run over transaction duration times and it indicates an anomaly in duration drops during the transfer as presented in Figure 7. The correlation between anomalies detected between these two SW and HW data processing highlighted bandwidth utilization drop that happens occasionally during the test execution. In this example, AI revealed an anomaly hidden inside the data transfer. After checking protocol specification, we confirmed that after each block of data packets, one control packet should be sent. It is confirmed that this behavior is in compliance with Ethernet protocol. However, this kind of insight can bring new ideas for further improvement of bandwidth, utilization of interconnect resources and traffic balancing.

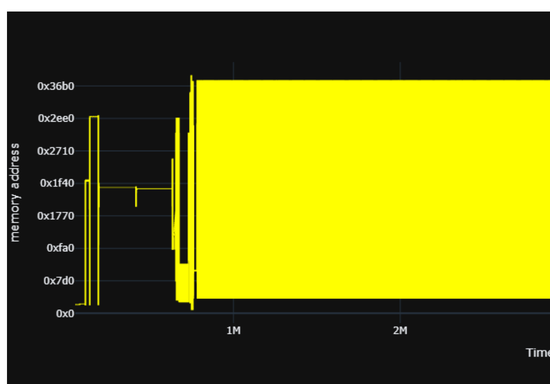

Figure 4 Visualizing memory address of instruction from tarmac over time by AI tool

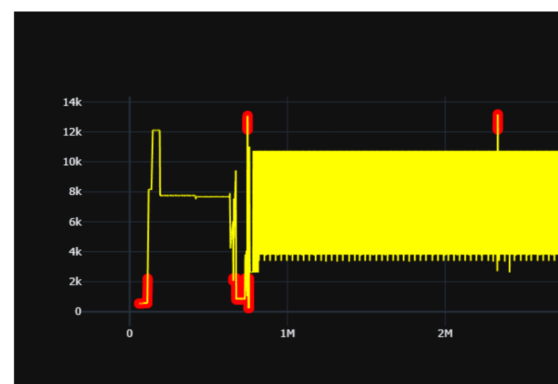

Figure 5 Visualizing anomalies of memory address values of instruction from tarmac over time by AI tool

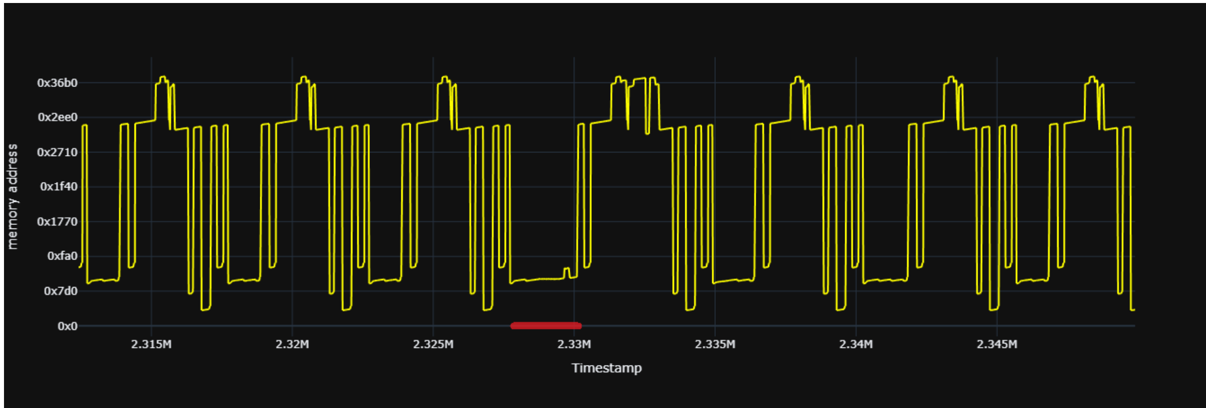

Figure 6 Detecting anomaly in visual report by AI tool

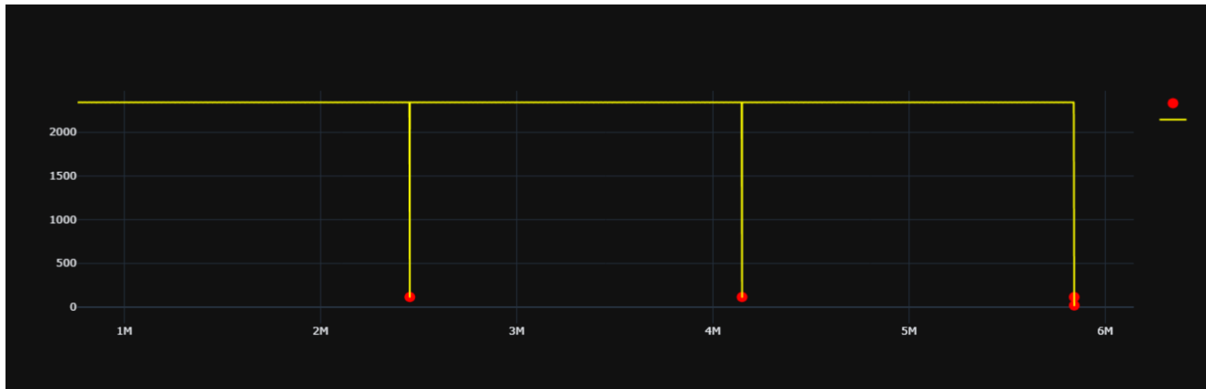

Figure 7 An anomaly in duration of transactions

D. Latency Issue with an Impact on Overall Performance Results

A good example of an application AI-based tool in SoC verification debugging is the case when the issue with transaction latency was found inside the interconnect. Again, the inquiry was based on UVC logs, which contained information that was extracted to a data set relevant for the research. This data set contained information about the round-trip times (RTT) that applied to each transfer captured in the test. All transactions were sent in an outstanding manner. AI analysis revealed that one transaction had a reasonably larger RTT compared to all other transactions. This anomaly was immediately detected by the AI debugging tool and the root cause is pointed out precisely. Since this is a performance-related issue, it would be hardly caught by the traditional debugging approach.

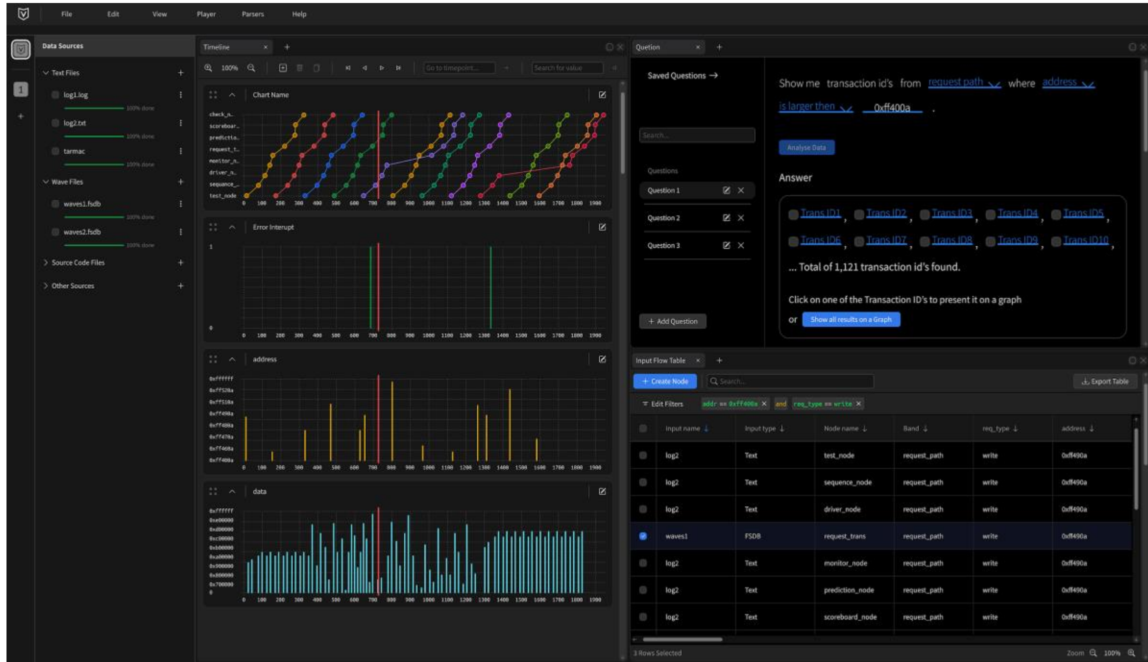

Figure 8 Cogita-PRO debug GUI

V. Overview of Used Verification Objectives and Platforms

The process of Ethernet verification is split into different technologies depending on the features to achieve results of the highest quality as soon as possible. There are 4 different platforms for testing purposes on the project: formal verification, RTL simulation, emulation, and FPGA. The debugging is empowered by usage of an AI debug tool that provides us with numeric and visual reports like in Figure 8. Our test plan is prepared in such a way that different functionalities are verified on the platform that is the most suitable for it. The tests are prioritized according to the importance of the features they cover. The test plan is given in Table I. It is reviewed in several review cycles by architects, designers, the verification team, and software developers.

Table I Verification plan for Ethernet multi-platform verification

Feature	Test	Priority	Platform
Connectivity and integration of interrupts and GPIOs		1	Formal
Register access over the configuration port	UVM	1	Simulation, AI-debugging tool
Master port access to all targets	UVM	1	Simulation, AI-debugging tool
Functional Safety features	UVM	2	Simulation, AI-debugging tool
Basic transfer over Ethernet interfaces	C-UVM	2	Simulation, emulation, AI-debugging tool
Low Power scenarios	C-UVM	3	Simulation, AI-debugging tool,
Gate Level scenarios	C-UVM	3	Simulation
Performance testing	C (with AVIP)	3	Emulation, Simulation
Bring-Up preparation	C	3	Emulation /FPGA
Software API verification	C (with AVIP)	4	Emulation /FPGA

It is important to have well-defined, measured, and analyzed verification metrics. Constrained random stimulus enabled by the usage of VIPs gives an efficient way of verifying different device states. Automatically collected functional coverage provides us with a view of the final quality that verification accomplished and a clear overview

of what is covered in overall verification. Special attention should be paid to tracking configuration settings (which blocks were included in the platform at the time when coverage is collected) and what kind of firmware is run.

VI. Conclusion

In this paper, there is a specific flow for SoC verification presented. It is demonstrated that verification should be done at different levels - block, subsystem, and top-level and on different platforms. The horizontal and vertical portability of the verification environment and test sequences should be supported in the sense that the parts of the environment can be used at the block, subsystem, and SoC level, but also for verification on different platforms (RTL simulation, emulation, FPGA). Smart usage of AI tools can empower the SoC verification team by applying a systematic approach, with maximum usage of the history of debugging. When using AI, it is very important to have an open mindset and explore simulation as big data. Benefit can be huge in the area of analyzing bottlenecks, HW-SW correlation, root cause analysis, throughput and anomaly detection. Our success story can be used as a safe path for the verification of complex subsystems inside modern SoCs. Finally, our aspiration is to eliminate the boundary among tools and enable users to run the same test on different platforms: RTL simulation, emulation, the FPGA prototype, and post-silicon validation making verification and debugging fast and easy. Also, the flow would be further improved by implementing AI features in boosting bug analysis, automatic test generation, and other areas in which big data analysis and AI can improve verification workloads.

References

[1] M. Sanie, “Debugging the debug challenge”, Synopsys, 2013. [Online]. Available: https://www.techdesignforums.com/practice/technique/debugging-debug/

[2] S. Morgansgate, J. Grinschgl, D. Lettnin, "Development of a Core-Monitor for HW/SW Co-Debugging targeting Emulation Platform," in Proceedings of Design and Verification Conference (DVCON), Munich, Germany, 2022.

[3] Riad Ben Mouhoub, “Palladium Z1 and Protium S1: The Fastest Way to Make Your System-on-Chip Prototyping a Success Story,” CDNLive, Silicon Valley, USA, 2018.

[4] R Frank Schirrmeister, Robert Kaye, "Reducing Time to Point of Interest with Accelerated OS Boot," ARM TechCon, Santa Clara, California, USA, 2014. [Online]. Available: https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2112-00-00-00-00-58-30/Reducing_5F00_Time_5F00_to_5F00_Point_5F00_Schirrmeister.pdf

[5] Ruchi Misra et al., ”An Accelerated System Level CPU Verification through Simulation-Emulation Co-Existence,” in Proceedings of Design and Verification Conference (DVCON), Munich, Germany, 2022.

[6] D. Ciaglia, T. Winkler, J. Kundrata, " Unified firmware debug throughout SoC development lifecycle," in Proceedings of Design and Verification Conference (DVCON), Munich, Germany, 2022.

[7] M. Graham, “Verification 2.0 – Multi-Engine, Multi-Run AI-Driven Verification”, in Proceedings of Design and Verification Conference (DVCON), Munich, Germany, 2022.

[8] “Ethernet Accelerated VIP User Guide”. 2022. [Online]. Available: https://support.cadence.com/