2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Harnessing the Strength of Statistics and Visualization in Verification

Olivera Stojanovic, Vtool, Belgrade, Serbia (oliveras@thevtool.com)
Uri Feigin, Vtool, Belgrade, Serbia (urif@thevtool.com)

***Abstract*— This paper explores how statistics and visualization, which remain largely unexploited today, can improve verification processes. Verification debug is data-intensive, and the biggest challenge for engineers is to grasp enormous amounts of data from different simulation outputs. As this paper outlines, successful deployment of Big-Data dataset techniques relies on the quality of the data across the entire workflow, from the preparation of clean, relevant data to its distribution application, interface utilization, performance, and system bottlenecks. Creative use of statistics is also presented here, introducing new concepts of transaction paths and visualization methods that lead to an effective verification process. Examples demonstrate the power of these methods and their benefits in solving crucial data challenges for engineers.**

Keywords—SoC (System on Chip) verification, Cogita-PRO, Statistics, Big-Data

I. INTRODUCTION

Typically, about 70% of the work put into finalizing a complex System-on-Chip (SoC) is dedicated to verification. Within this scope, approximately half, equivalent to roughly 35% of the entire SoC development timeline, is devoted to debugging [1]. Verification debug is data-intensive, requiring engineers to shoulder the biggest challenge of verification, which is to grasp these enormous amounts of data derived from multiple simulation outputs such as UVM logs, waveform data base, source code, VIP transactions trace files, and CPU execution and disassembly files. Such heavy lifting of data calls on us to look at verification as one Big-Data dataset. In applying this approach, verification engineers are naturally required to adopt an entirely new mindset where things are done very differently to what they are used to.

In this paper we will focus on how the standard verification technique can be improved by approaching verification outcomes as one Big-Data dataset, which is much more effectively understood using statistics and visualization diagnostic equipment.

The benefits of our approach include:

1) *Ensuring the quality of randomization*
   i) *Understanding and improving quality of the test scenarios*
   ii) *Identifying constraint issues with distribution in the early stages of the project*
   iii) *Exploring transaction data field correlations*
   iv) *Grasping transition patterns*
   v) *Visualizing transaction path types and distribution over test*
2) *Understanding the utilization of outstanding transactions*
3) *Improving performance by applying distribution on duration and gaps between transactions*

Currently, data that is prepared to serve as database is primarily used for exploring AI applications in verification. It has not been sufficiently applied for investigating statistics and visualization, which can greatly benefit from these datasets and improve verification.

II. RELATED WORK

The industry has acknowledged that verification poses a bottleneck in the SoC development process, prompting substantial investments across various domains to address this key challenge.

EDA companies are actively investing in integrating ML and AI capabilities to enhance verification workloads. ML and AI boost the efficiency of multiple runs across various engines throughout the entire SoC design and

verification processes, bug prediction, correlation of test failures with RTL code facilitation, automatic test generation, resource optimization, and automated test-case failure classification [2]. Many companies are currently investing in generating AI-driven RTL code, verification environment, and tests.

A set of AI algorithms creates an efficient approach for handling [3]:

1) *Unexpected transactions, for*
   - *Matching source and destination endpoints in failing transfers*
   - *Resolving common failures*
   - *Handling security access and caching transactions*
   - *Interleaving burst translations*
2) *Error response transactions, for reserved and/or broken address ranges*
3) *Distribution of transactions, for qualifying test and verification environments*
4) *Utilization of outstanding transactions, for improved performance*
5) *Detection of repetitive transaction patterns irregularity, for measuring throughput and detection of transfer timeouts*

While the industry focuses the bulk of its exploration on how to apply AI in verification, little research has been dedicated to the application of statistics and visualization in this field, even though preparation of data as database is as the primary, most common ground for all of these. This paper will focus on the untapped power of statistics and visualization, highlighting their benefits and the issues they can address. By showcasing real examples and clarifying the advantages of integrating these techniques into verification workflows, we aim to inspire a paradigm shift in how verification debug can be better approached and improved.

III. METHODOLOGY

The goal of this paper is to prove the successful application of on-project statistical techniques and visualization. It focuses on verification environments developed in SystemVerilog using UVM methodology, and suggests using the AI-driven debugging tool Cogita-PRO as the platform for delving into, retrieving, and linking the data, as well as for visualizing the outcomes. Data is extracted from UVM log and RISC-V execution files, alongside additional data fabricated from existing log files, which includes timestamp, duration of transactions, number of outstanding transaction, etc.

Statistics and visualization over simulation data and metadata offer:

1) *Clear test scenarios before implementing functional coverage*
2) *Exposed constrains issues*
3) *Accelerated cover test scenarios*
4) *Shorter regression time*
5) *Maximized interface utilization - number of outstanding transactions*
6) *Improved performance tests quality*
7) *Identified bottleneck in the SoC*

IV. RESULTS AND DISCUSSION

Results are grouped into four test cases. Each test case outlines the project, while explaining the type of visualization and statistic metrics used to resolve a certain issue based on their benefits.

A. Test Case 1: AXI Interconnect

The test case example used here involves an AXI interconnect, which features numerous masters and slaves. The output of the simulation is a UVM log file, containing prints for each request and response AXI transaction across each master and slave VIP instance. The database is constructed using the relevant data fields extracted from each AXI transaction found in the simulation UVM log file. Additionally, verification-specific metadata is extracted (e.g., timestamp, transaction path, transaction duration, gap between transactions, number of active transactions, etc.). The results presented for this test case use this database.

Functional and code coverage serve as indicators of the verification environment and stimulus quality. However, they are often inaccessible until later in the project timeline. To overcome this, data can be diagnosed from extracted messages of the log file. As Figure 1 shows, for an entire lifecycle of a master request transaction during simulation, correlation between fields can reveal constraint issues. In this instance, it is noticeable that *burst_type* and *burst_size* exhibit a single value throughout the simulation, indicating that these functionalities are not thoroughly tested. On the other hand, it is also clearly visible that additional fields are randomized with no strict correlation. As is readily apparent from the graph, all *mst_index* values are executed and fully random during the simulation, which stands out instantly owing to the distinct colors assigned to each value.

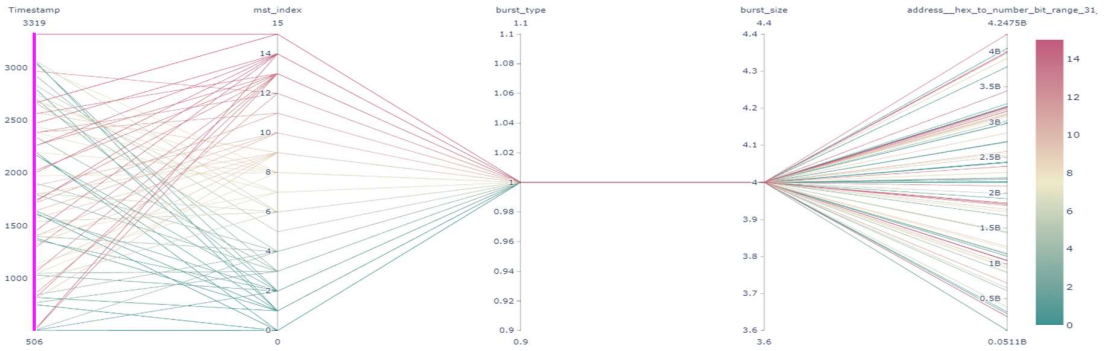

Figure 1. Transaction data field correlation.

Figure 2 presents a transaction flow from a master request, slave request, slave acknowledge, and master acknowledge connect by a specific id. The propagation of the entire flow is presented with the related fields of this transaction, and the direct test coverage between slave and master is easily spotted.

Solely based on statistics and visualization, this type of quick and efficient diagnostics takes place before any functional coverage code is even implemented, providing valuable conclusions to be drawn just from the simulation's extracted messages. Hence, fixing constraint issues and achieving even distribution can be done much earlier in the project, leading to immense savings from faster full coverage and shorter regression.

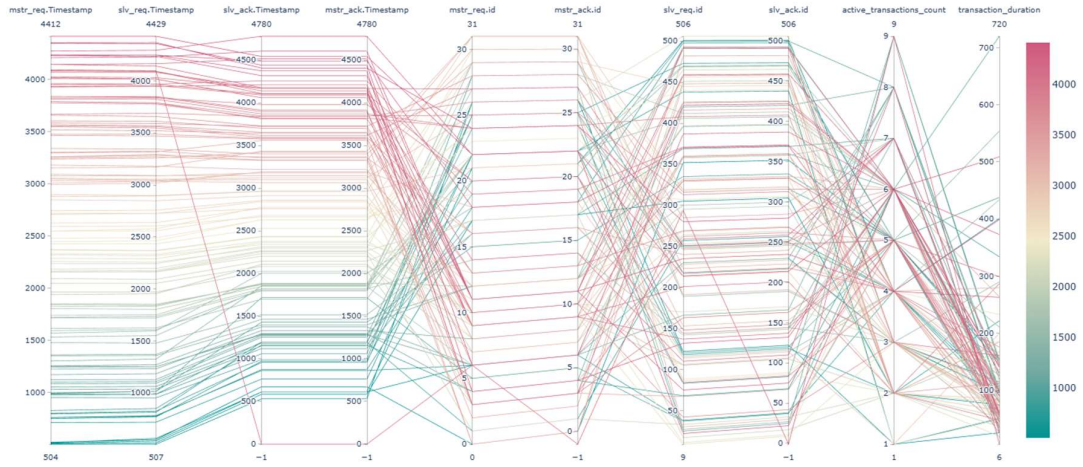

Figure 2. Transaction path data field correlation

Figure 3 visualizes a test case in three clear sections:

- Transaction flow – Each id is represented by a colored line connecting four dots signifying the master request, slave request, slave acknowledge, and master acknowledge (top, colored lines)

- Number of active transactions – Fabricated data considering start and end of transfer (middle, in orange)
- Error occurrence – Point 4100ns appears clearly (bottom, in blue)

This simple representation of the transaction flow exposes the density and trends of outstanding transactions during the simulation. This overall visualization exposes the direct correlation between error and maximum number of outstanding transactions reached.

Figure 3. Transaction path data field correlation

B. Test Case 2: RISC-V Verification

The results presented below, are related to RISC-V verification for a project that required extending the Veer EH1 RISC-V core with additional instructions [4]. Open-core verification solutions were explored, and a RISC-V DV was chosen based on SV/UVM technology that offers smooth integration [5]. The goal of this test case project, was to verify and determine the quality of our chosen solution and its extension capacity for new instruction. The full simulation was done using Universa SoC platform [6]. Besides the standard components required for SoC verification, an additional monitor was developed and used for monitoring signals on CPU trace interface and storing the trace in the log file.

Figure 4 presents part of this log file.

```
//Sim Time,      Cycle : #inst  hart   pc    opcode    reg=value   ; mnemonic
//-------------------------------------------------------------------
      3575,         30 :       #1 0 20000000 f14022f3  t0=00000000 ; csrrs   t0,csr_f14,zero
      3675,         31 :       #2 0 20000004 00004301  t1=00000000 ; c.li    t1,0
      3775,         32 :       #3 0 20000006 00628263              ; beq     t0,t1,0x2000000a
      6275,         57 :       #4 0 2000000a 00000e97  t4=2000000a ; auipc   t4,0x0
      7775,         72 :       #5 0 2000000e 00ce8e93  t4=20000016 ; addi    t4,t4,12
      7875,         73 :       #6 0 20000012 000e8067              ; jalr    zero,t4,0x20000014
     10475,         99 :       #7 0 20000016 400019b7  s3=40001000 ; lui     s3,0x40001000
     10575,        100 :       #8 0 2000001a 10498993  s3=40001104 ; addi    s3,s3,260
     11975,        114 :       #9 0 2000001e 30199073              ; csrrw   zero,csr_301,s3
     12075,        115 :      #10 0 20000022 00013c97  s9=20013022 ; auipc   s9,0x13000
     12175,        116 :      #11 0 20000026 192c8c93  s9=200131b4 ; addi    s9,s9,402
     12175,        116 :      #12 0 2000002a 0000a997  s3=2000a02a ; auipc   s3,0xa000
     13475,        129 :      #13 0 2000002e f2698993  s3=20009f50 ; addi    s3,s3,-218
     13575,        130 :      #14 0 20000032 0019e993  s3=20009f51 ; ori     s3,s3,1
     13675,        131 :      #15 0 20000036 30599073              ; csrrw   zero,csr_305,s3
```

Figure 4. CPU trace file

Based on data extracted from the trace file, the results and conclusions presented here include the simulation time, pc value, opcode as well as instruction – mnemonic.

Figure 5 shows the values of opcode, pc, and mnemonic over time. As the red line shows, SW executed sequentially with minimum utilization of jump instructions. The anomaly on this red line (circled in green) indicates the code branching with minimum occurrence in the code.

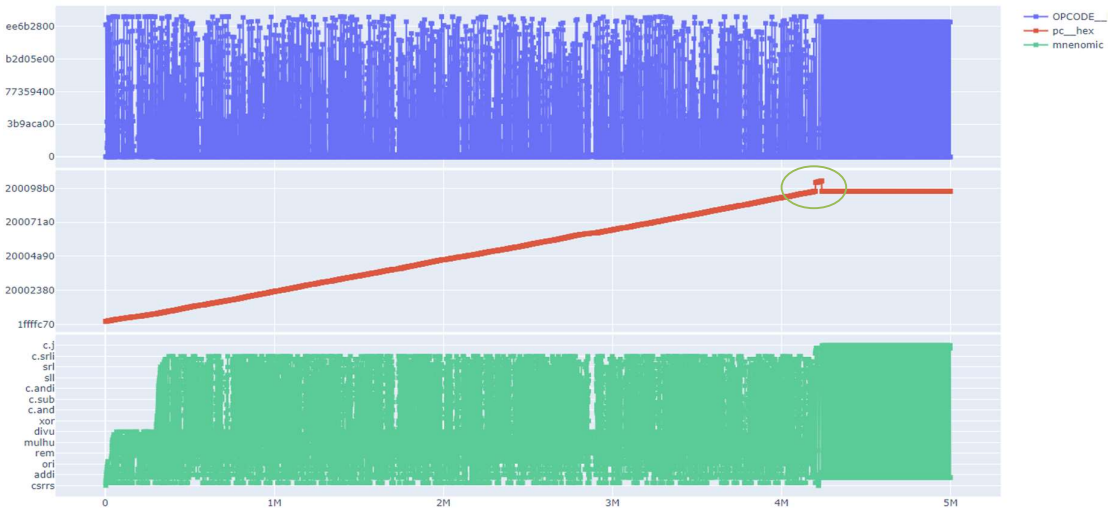

Figure 5. Values of opcode, pc, and mnemonic over time

Figure 6 shows the instruction distribution, confirming the minimal utilization of jump instruction while also indicating that a uniform distribution of other instructions exists inside the simulation SW.

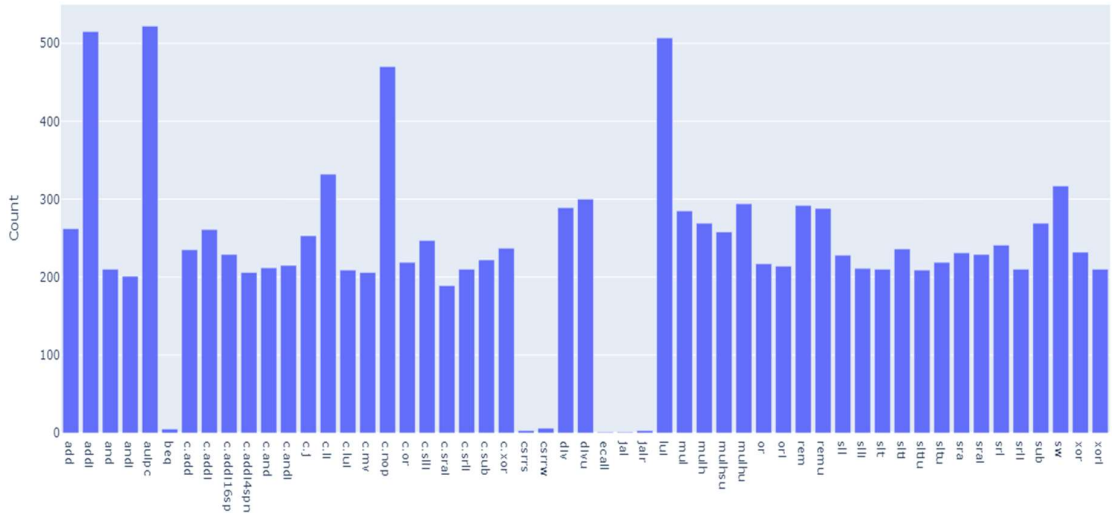

Figure 6. Instruction distribution graph

To check in the SoC time needed to execute a single instruction, a new field is fabricated, called timestamp_datafield_difference_self.

Figure 7 presents the correlation between instructions and their duration, spread across three vertical axes:

- Timestamp – The simulation time
- Instruction mnemonic – The RISC-V instruction
- Timetamp_datafield_difference_self – Single instruction execution time, which is a field fabricated from the log file by calculating the time difference between two instructions

Time differences are shown using a color scale that differentiates the instructions by their duration time (lowest in green and highest in red). Instructions with the longest execution time are easily spotted, and in this case there are four such cases (plotted as red lines).

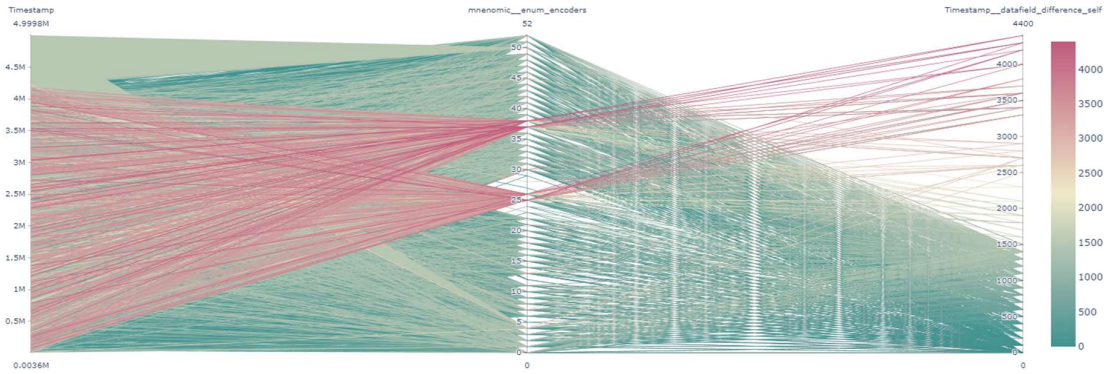

Figure 7. Correlation between instructions and their duration

The correlation between the timestamp and mnemonic is also shown, determining the randomization level. Clearly, there is no correlation between simulation time and instruction, concluding that randomization has been achieved throughout the simulation.

Figure 5, Figure 6, and Figure 7 display statistics and visualization in a way that makes it easy to determine the quality of our chosen solution, particularly:

- Randomization level of SW for CPU verification
- Distribution of instruction
- Instructions with highest execution time

Future work will delve into the CPU and SoC verification and benchmark, based on RISC-V ISA that contains custom extended instructions.

C. Test case 3: Descriptors

In this test case, a change request added a new data buffer that affects calculations for transfers in sequences, requiring sequences update. This change influenced the correlation of the generated fields and sequences, including address offset, transfer length, and predefined possible orders of descriptor execution. The main challenge was that the RTL was not yet ready, but to minimize debug time, we still had to determine if the stimulus would work well with this new RTL design before running simulation.

To check if randomization of sequence order is performed correctly, we used the transition probability matrix. The expected behavior of one command life cycle is *command_desc -> data_desc -> data_pointers_fetch ->*, then multiple *single_transfers,* and lastly, depending on command size, the cycle is either completed or a new *data_desc* is generated.

Figure 8 shows the probabilities of transitions, both before the bug constraint is fixed (left) and after (right). On the left, it is easy to spot that in the generated test scenario there are 0.26 probabilities for transition *data_desc->data_desc,* meaning it is an illegal transition that is incorrect according to the specification. This helped us find issues in constraints related to sequences execution. On the right, the correct transitions can be seen after the constrains bugs were fixed.

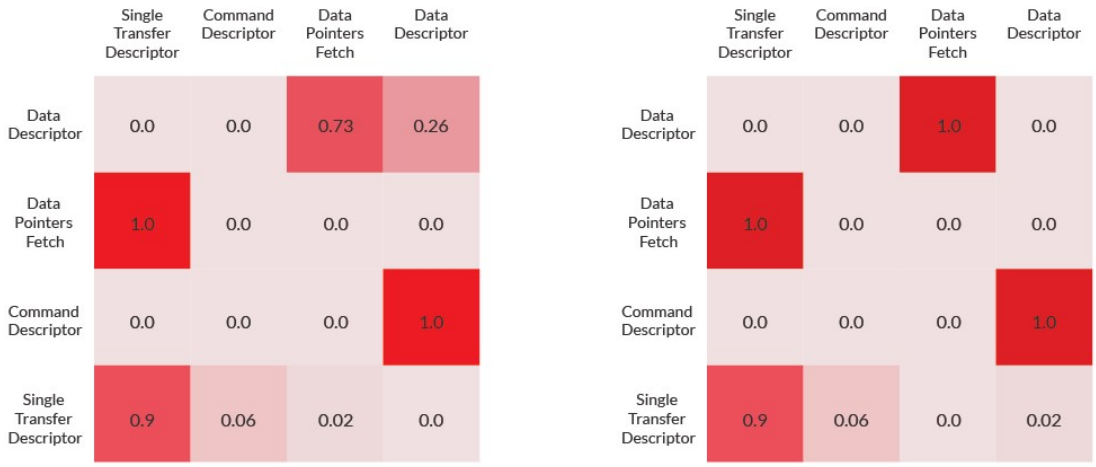

Figure 8. Transition probability matrix, before (left) and after (right)

D. Test case 4: Stress Test Quality Assessment

The following testcase is taken from a bridge translating AXI read/write requests into proprietary protocol B. The verification engineer was tasked with evaluating the stress test exercising the module to its expected limits.

A new dataset is fabricated, capturing the correlation of all requests (ext_prot_rd_cmd) to their respective responses (ext_prot_rd_rsp). Metadata calculation was used to extract the number of outstanding transactions of protocol B at any given time.

Figure 9 focuses on the first 5000 ns of the test. The first eight transactions start every 10 ns, reaching a maximum outstanding transaction count of eight (in purple). The next transaction is requested 1500 ns later, after three transactions have completed. From here onwards, new transactions are requested with a delay between the transaction distribution, as seen in Figure 10.

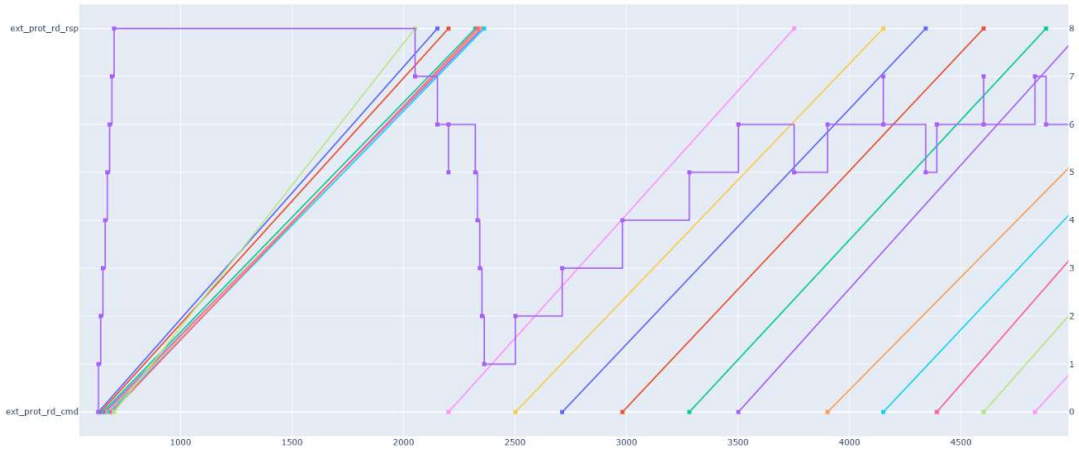

Figure 9. Transaction chart overlaid with outstanding transactions count

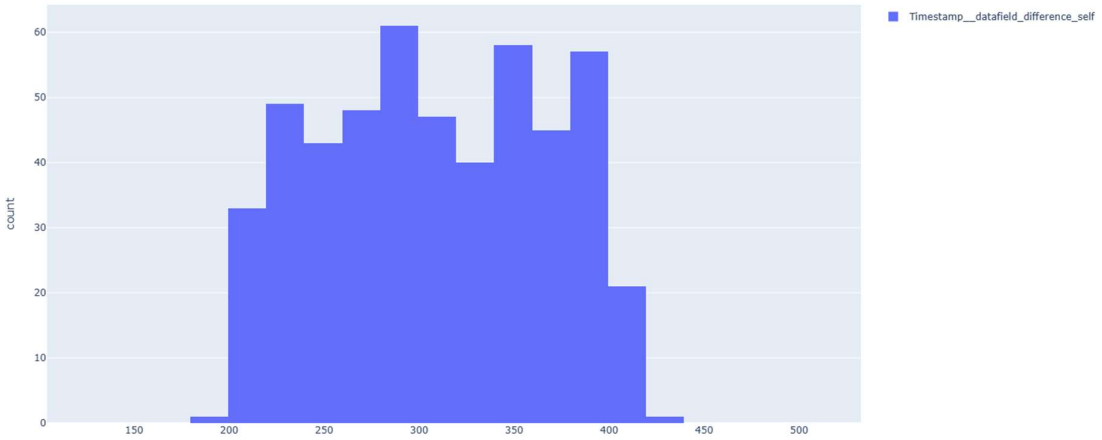

Figure 10. Delay between further transactions

Reviewing the entire test in Figure 11 reveals that the maximum outstanding transaction count of eight was achieved only once, in the beginning of the test. After a hysteresis of three, the stress test never managed to push more than seven outstanding transactions at any given point of time.

Even though classical coverage metrics approve that the maximum capacity of the pipe was reached, observing the waveform only indicates that repeated activity happened across the entire test. It is this statistical analysis that allowed us to zoom in on the crucial performance bug, which prevented the module from working at its full capacity after reaching the maximum threshold.

Figure 11. Active transaction count, full test view

In Figure 12, a second statistical analysis was performed to calculate the overlapping transactions. It is clear that the new transaction finished 12 out 513 times before the previous one ended, with a maximum overlap of only one. Analysis revealed that the stress test did not exercise the full capacity of all the out-of-order transaction functionality.

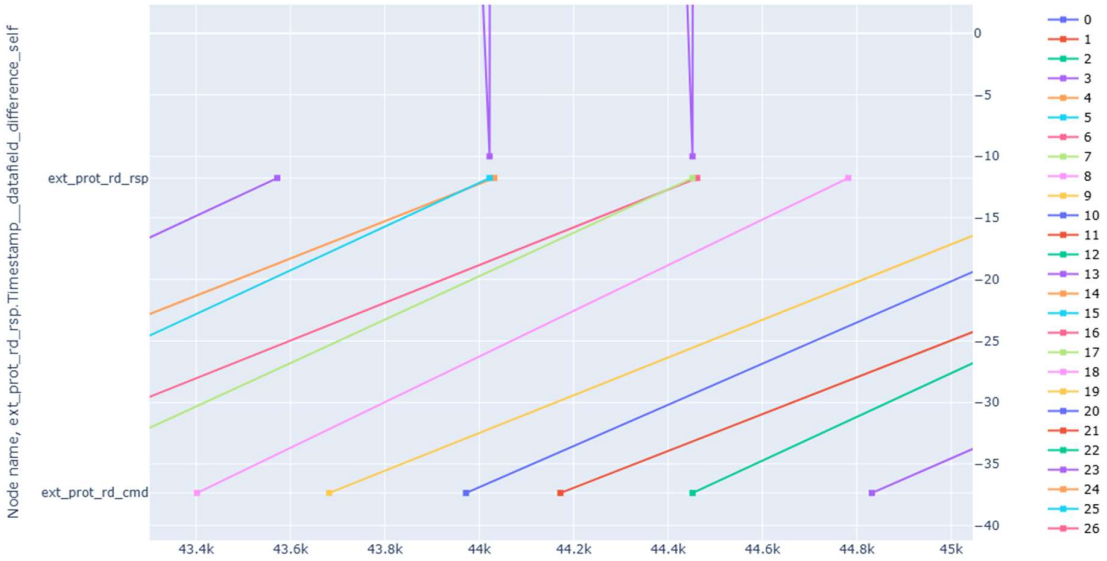

Figure 12. Transaction overlapping

V. CONCLUSION

The power of statistics and visualization in functional verification can be harnessed to easily analyze, interpret, and present SoC behavior, thereby ensuring functionality and reliability. This approach is applied on Big-Data datasets extracted from multiple simulation output results using the AI-driven platform, Cogita-PRO. The key advantages of this approach include minimizing debugging time and maximizing verification quality.

In the test cases presented in the paper, we have proven that statistics can be a powerful tool for understanding random generated tests scenarios, constrains issues, and uneven distribution. Visualization of extracted and fabricated verification-specific data can point to correlations between errors and simulation data. Moreover, new concepts of transaction flow enable engineers to clearly understand the density of outstanding transactions while revealing issues in the verification environment or bugs in the design.

Even though the benefit is clear, and results can be observed instantly, statistics and visualization are not widely used in the industry. Yet, as this paper shows, by viewing simulation as one Big-Data dataset, verification engineers can adopt a new mindset that gears them with macro-level insights of the simulation, from which they can then dive into the specifics with ease and simplicity.

With the industry heading towards open-source solutions and AI-generated code, we forecast that statistics and visualization will become the key diagnostic tools in verification. These enable engineers to first gain a clear understanding about the quality of the generated code and simulation outputs, as this paper has demonstrated.

VI. REFERENCES

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

[2] M. Graham, "Verification 2.0 – Multi-Engine, Multi-Run AI-Driven Verification," in *Proceedings of Design and Verification Conference (DVCON)*, Munich, 2022.

[3] A. Ravitzki, O. Stojanovic and N. Mitrovi, "Efficient application of AI algorithms for large-scale verification environments based on NoC architecture," in *Design and Verification Conference (DVCON)*, San Jose, 2024.

[4] "Cores-VeeR-EH1, VeeR EH1 design RTL," [Online]. Available: https://github.com/chipsalliance/Cores-VeeR-EH1.

[5] "RISCV-DV Documentation," [Online]. Available: https://htmlpreview.github.io/?https://github.com/google/riscv-dv/blob/master/docs/build/singlehtml/index.html#document-index.

[6] "Universa, Integrated HW/SW ready-to-use RISC-V SoC," [Online]. Available: https://www.thevtool.com/universa/.

[7] M. Sanie, "Debugging the debug challenge," 2013. [Online].