# Efficient application of AI algorithms for large-scale verification environments based on NoC architecture

Anna Ravitzki,Vtool doo, Belgrade, (annar@thevtool.com)

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

Nemanja Mitrovic, Vtool doo, Belgrade, Serbia (nemanjam@thevtool.com)

***Abstract*—**The growing complexity of SoCs is challenging the efficiency of present-day verification approaches, making verification processes increasingly more involved. As waveform databases expand in size and simulation run-time becomes more time consuming, logs and code execution traces are harder to read and understand. This requires greater resourcefulness from verification teams, alongside more powerful verification tools. By addressing the various complex SoC verification bottlenecks, this paper proposes a viable, highly effective verification solution that uses Big-Data techniques for analyzing standard verification outputs. The new approach presented here transforms traditional debugging by applying proven AI techniques and algorithms on unified Big-Data datasets, derived from multiple sources. To highlight the applicability and potential of AI for complex SoCs verification, the new approach is tested, demonstrated, and ultimately proven to be successful using several NOC verification examples. It provides the nearest endpoints addresses algorithms, proving to be highly more effective in matching source and destination endpoints in failing transfers. The approach also suggests potential resolutions for common failures by finding the common data values in failed transfers, and its time-scale analysis extracts deviations in transfer execution time to suggest potential throughput issues, transfers timeouts, and more. Although standard debugging methodologies can resolve such common NoC verification issues, the new AI-driven approach described in this paper significantly accelerates SoC verification, transforming the entire verification process into a highly efficient one.

***Keywords—NoC (Network on Chip), SoC (System on Chip) verification, Cogita-PRO, AI (Artificial Intelligence), Machine Learning (ML), Big-Data***

## I. Introduction

The immense variety of SoC applications has introduced a wide range of techniques for their development and verification. In the realm of verification, each industry poses different challenges, such as the use of digital and analog verification environments and checks in the automotive field or the debugging of large-scale data in the data path processing field. This paper intends to focus on the latter, efficient debugging of high-complexity data path SoCs.

In general, around 70% of the effort invested for taping out a complex data processing SoC is spent on verification. Half of that effort, which translates to about 35% of the total invested SoC development time, is invested in debugging.[1] Furthermore, studies show that IC/ASIC verification engineers spend more time on debugging than on any other activity. Since debug time varies significantly between different projects, it introduces insurmountable uncertainties in project planning and scheduling.[2]

Contemporary SoC complexity impacts verification, requiring expertise and combined knowledge in many different domains such as hardware, software, analytical, and scripting. The main challenge in standard debug verification of complex data path SoC has to do with the massive data infrastructure, including software source code, disassembly, execution traces, UVM-SV source code, log files from verification methodology (e.g., UVM), RTL design source code, waveforms, and schematics. Therefore, data path verification is data-intensive by nature, slow, hard to understand and trace, and reliant on one focused part of simulation, which makes it a bottom-to-top process. This paper proposes a new top-to-bottom approach that transforms verification into a macro-level process, proving to be highly efficient and reliable since it is based on understanding the verification outcome as a one BigData dataset, which is also suitable for AI algorithms and techniques.

The AI algorithms run through the whole dataset, presenting information about issues beyond the thinking scope of the verification engineers, amplifying their capacity for capturing software irregularities and unused bandwidths across interconnect transmission. This is crucial for ensuring improved performance in a radical way.

One of the many complex topics in verification of data processing SoCs, is NoC (Network on Chip). NoC is typically distributed over the whole design, containing one main NoC and plenty of sideband NoCs as part of source endpoints and memories. Thus, one transaction from source to destination point passes over several different NoCs, transforming a global address many times until it reaches the destination. In addition, other transfer parameters are compressing and decompressing during inter-NoC communication, introducing another layer for potential bugs.

Various projects have shown that, in large-scale SoCs, the following typical NoC verification challenges must be resolved:

1) *Unexpected transactions,* for
   - Matching source and destination endpoints in failing transfers
   - Resolving common failures
   - Security access transactions
   - 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 environment
4) *Utilization of outstanding transactions,* for improved performance
5) *Detection of repetitive transaction patterns irregularity,* for measuring throughput and detection of transfer timeouts

The number of transactions varies depending on the test goals, and often totals over a thousand, including different source to destination transactions per single test. If the faulty transaction numbers are high, as is usually the case in the beginning of a project, it is very hard to use standard debug techniques. The main issue here, is the time required for reverse tracing destination to source endpoint from logs, and once the transfer issue is localized the challenge is to trace it from waveforms and schematic tracers. The debug process is deductive, and certainly leads to the resolution of the existing issue, but it is single-transaction oriented. Missing information is the number one issue on which debug time is mostly spent, and compounded by the duration of repeated debug steps amount to inefficient processes. Running regression can help somewhat in providing valuable information, but this is typically unavailable in the beginning of a project.

To hurdle such pertinent verification issues, this paper proposes several AI algorithms that speed up the debug process by analyzing the entire verification outputs as one single dataset. For example, the **address correlation (AC)** algorithm is very suitable for efficiently matching source and destination endpoints in failing transfers. The **machine learning (ML)** algorithms are perfect for resolving common failures, security access transactions, caching transactions, interleaving bust translations, and for finding reserved and/or broken address ranges. **Statistical algorithms (SA)** provide a good overview of covered transactions and can measure the quality of tests and verification environments. Algorithms used for detection of repetitive patterns and irregularities over the whole database are a good match for measuring throughput and detection of transfer timeouts. In addition, a combination of algorithms will be proposed, showing outstanding transaction utilization.

The results presented in this paper have been successfully proven in several different NoC verification SoC projects.

## II. Related work

Increasing design complexities and time to market constraints have rendered verification to be a bottleneck in the development process. To address this and enable technology to be released faster and bug-free, verification needs to adopt new approaches and techniques. By its nature, verification is data-intensive and well-suited for the application of machine learning (ML) techniques. In the industry, numerous ML algorithms have been tested across diverse facets of functional verification, yielding varying levels of achievement mainly in requirement engineering, static code analysis, verification acceleration, bug detection, and localization.[4]

EDA companies are investing in adopting ML and AI features to optimize multiple runs of multiple engines across the entire SoC design and verification campaigns. These features are also applied in AI-driven bug prediction to link test failures to RTL code changes, and in automatic test generation, resource optimization, and automated test-case failure triage to improve verification workloads.[3]

This paper will focus on efficiently employing AI algorithms for debugging and resolving common performance issues on large-scale verification environments, based on NoC architecture.

## III. Methodology

The goal of this paper is to prove the successful application of specific AI algorithms and analysis on multiple different projects. The chosen verification objective is complex HW/SW co-verification, developed in SystemVerilog using UVM methodology and driven by C and UVM code test cases.

The platform proposed in this paper for exploring, extracting, and connecting the data over which AI algorithms are run and results are visualized, is the AI-debugging tool Cogita-PRO. Inputs used for AI analytics were UVM log file, execution file/tarmac, disassembly file, and waveform database. Each result presented in the paper is gained using a single test case.

### *A. Unexpected transactions*

One of the typical issues across SoC verification, is receiving transactions detected at the destination endpoints but not predicted by the dedicated scoreboard in the destination queue. In practice, the number of mismatches could be very high and their analysis very complex. Typically, such issues occur due to the presence of a system of NoCs that introduce destination endpoints address translations. Per each mismatch, debug processes require manual backtracking and calculation in order to understand source endpoint initiator. Additionally, outstanding transactions and overlapping address and data phase result in the interleaving of mismatch cases, making the debug process even harder and more time consuming. This paper proposes the use of a new algorithm, called Address Correlation (AC) algorithm, for resolving source and destination endpoints mismatch, and also demonstrates the use of ML algorithms for finding common failures.

The AI-debugging tool takes the address from the error transaction as a benchmark and runs an AC algorithm using address values from all source transactions. The result is the exact failed transaction origin. Graphical representation on the timeline of all messages related to this transaction indicates issues in prediction logic that are related to address translation.

ML algorithms are good candidates for simulations containing a large number of pass transactions with sporadic failures. ML algorithms deploy all transaction fields and duration, while the classifier uses passing/failing criteria. The result is visualized as a decision tree, presenting all the common values of every failing transaction. In cases where the precision of the algorithm is less than 100%, potential checkers of implementation issues are indicated.

### *B. Error response transactions*

During initial SoC development, frequent updates and re-assignments of address space (reserve function, or vice versa) generate hysteresis between design and verification update alignments, which requires additional debug time to repeat the same verification tasks over again. Moreover, traditional debugging with standard deductive techniques lacks the robustness needed for resolving complex issues quickly, slowing down the process even more. To attain an efficient debug process with immediate analysis of the entire memory address space, the ML algorithm proposed in section A is needed.

### *C. Transaction distribution*

The functional and code coverage indicate the quality level of the verification environment and stimulus. Yet, since they are typically unavailable before the project end phase, potential bottlenecks in the verification environment remain hidden. By applying the SA algorithm over the dataset, it is possible to see the results at the very beginning of the project, visualized on a heatmap. These results directly point out all the verification address constraint issues. The visual distribution of extracted data values from log file and/or waveform provides insight into the simulation quality, enabling engineers to promptly take action and improve the verification environment. By rewriting the constraints and optimizing the distribution in the early stage of a project, the duration of regression is slashed, enabling the last milestone to be reached much faster.

### *D. Outstanding transactions utilization*

The utilization of maximum outstanding transactions affects performance significantly. According to the number of temporary active transactions over a predefined time span, AI algorithms are capable of distinguishing different data paths and interface utilizations. Low-level utilization in the simulations reveals the issues in the design configuration of the number of outstanding transactions. The results are applicable for throughput and performance analysis and for HW/SW profiling.

### *E. Repetitive transaction patterns irregularity detection*

Some specific NoC back-to-back applications consist of the round-trip times (RTT) transaction set. RTT is extracted from a dataset, based on the calculated duration of the transaction of request and response messages. In simulation, the transaction set carries a typical repetitive pattern that AI algorithms easily detect from the RTT, while also isolating irregularities. Additionally, by analyzing the time difference in execution of extracted patterns, the AI algorithm also points out performance-related issues that are otherwise very hard to expose using standard techniques.

## IV. Results and discussion

Section III defines standard NoC verification issues, and proposes efficient AI-based debugging solutions. The NoC verification problems are unexpected transactions, error response transactions, transactions distribution, outstanding transactions utilization, and repetitive transaction patterns irregularities detection.

### *A. Unexpected transactions*

The NoCs data path transforms the global address transmitted from source several times before it finally reaches the destination. This makes the process vulnerable to many RTL bugs and verification environment errors in predicting destination address. Figure 1 represents the AI AC algorithm that links missing source/destination endpoint pairs.

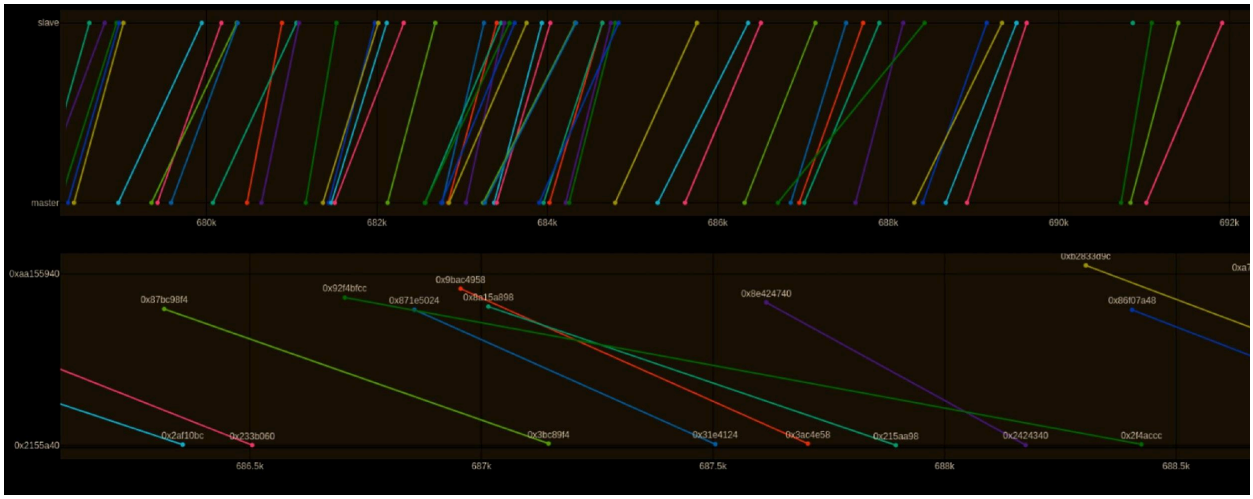


Figure 1. The AI AC algorithm linking missing source/destination pairs

The missing source recognition is done by exploring the log file for correlated addresses. For example, (source address, destination address) = (0xaa155940, 0x155940). This analysis enables the origin of the missing source endpoints to be understood much faster, which is immensely beneficial for the entire process. Standard debug techniques require the investigation of single transactions, one by one, but AI AC algorithms capture all the mismatch results from a single transaction debug, all at once.

In addition to the significant acceleration of debug time, applying ML algorithms on the analysis results also provides statistical information regarding the NoC issue root causes. Figure 2 represents what is common to all mismatched source/destination address.

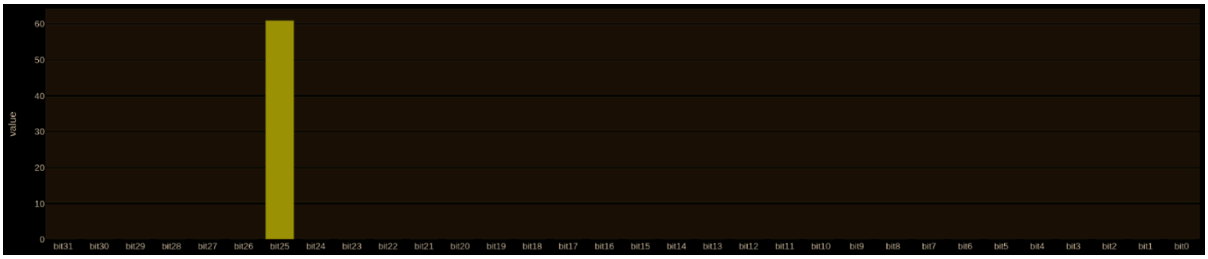


Figure 2. ML algorithm search of linked source/destination pair for finding common failures

Figure 2 presents the ML algorithm process outcome, based only on the classification and clusterization of address values, taken from the previous analysis. The result clearly points at the one singular common cause of all source/destination mismatch addresses, which Figure 2 exposes to be 0 at 25th bit of the transformed address. This result points at either an RTL design issue, indicating the exact incorrect NoC address translation, or to a verification environment prediction being incorrect.

### *B. Error response transactions*

The most typical error, regardless of NoC presence and/or SoC complexity, is failures with accessing reserved or unimplemented address space. If the error response transaction addresses fall within a similar range, it can point to

the access reserve address space. Figure 3 shows the ML algorithm classification and clusterization of address space of the error response transactions.

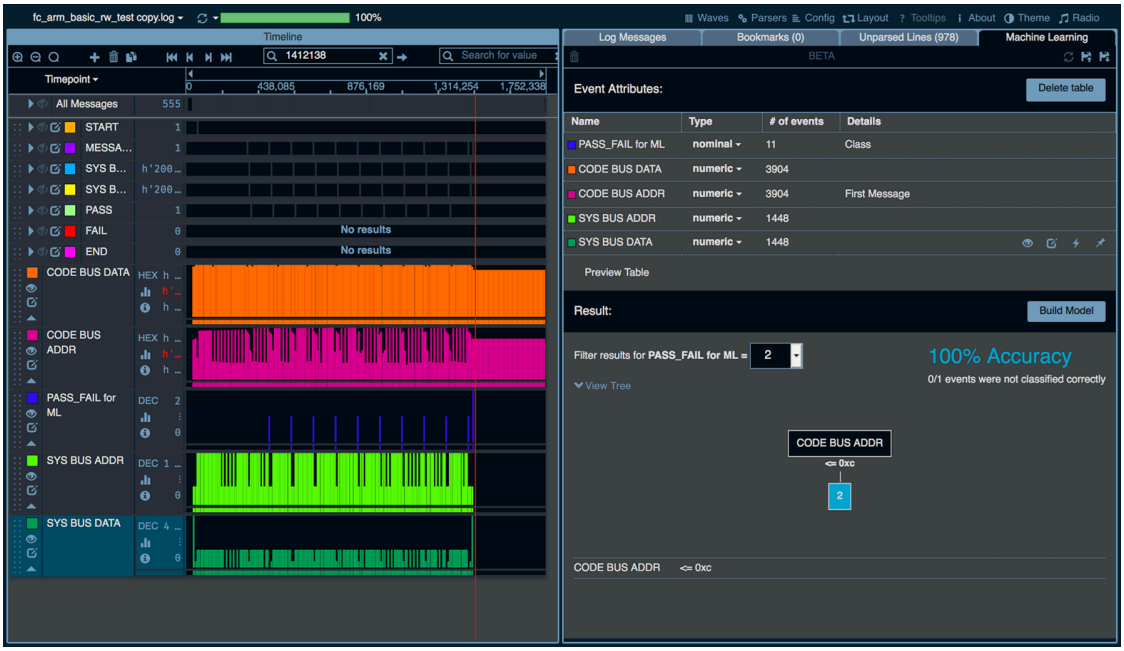


Figure 3. ML algorithm classification and clusterization of error response address space

The ML algorithm is applied to all the transaction fields, while HRESP is used as a classifier. Based on the results presented in Figure 3, all failing transactions accessing the address space are lower than the address value 0xC, indicating reserved address space access. This points out issues in the usage of instruction memory addresses.

*C. Transactions distribution*

The functional coverage measures the quality level of the entire verification environment, yet during the verification environment development phase the stimuli quality is not evaluated. Complex NoC-based SoCs stimulus consists of transactions from multiple sources to multiple destination endpoints. Depending on the testing subject, particular sources and destination endpoints are more or less triggered, which is highly valuable information that is very hard to extract. It can point to either unreachable destinations, RTL design holes, or potential verification dead loops of the same destination access. Figure 4 presents the distribution of source/destination endpoint transfers on a heatmap generated by AI SA algorithms.

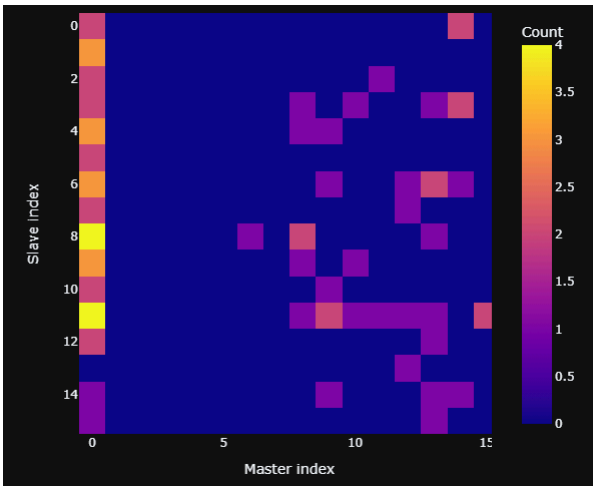


Figure 4. AI SA algorithm-generation heatmap of source/destination endpoint transfers distribution

The results point at the stimulus utilizing source 0 endpoint the most, initiating transactions to all destinations except 13 and a few other sources/destination transactions. This behavior is expected, since destination 13 is not accessible to source 0. If destination 13 was successfully accessed, it would be an RTL bug that is hard to catch at such an early stage of a project.

*D. Outstanding transactions utilization*

The AI algorithm is fed with data per transaction, start time, end time, initiator number, and destination number. Figure 5 shows the number of multiple outstanding transactions over time, revealing one bug in the design of the number of outstanding transactions.

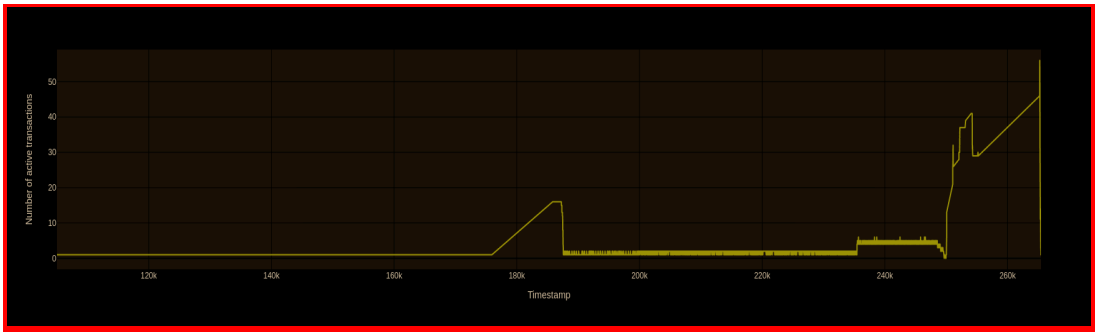


Figure 5. Number of multiple outstanding transactions over time

This bug is evident from the results, which show that the maximum outstanding transactions were never reached during the simulation. The same algorithms are applicable for throughput and performance analysis and for HW/SW profiling.

*E. Repetitive transaction patterns irregularity detection*

Instruction memory addresses of instruction are extracted from tarmac files, and visualized over the timeline. The AI algorithm applied on transaction duration times indicates an irregularity in duration drops that occur during the transfer. Figure 6 and Figure 7 show that isolated irregularities were detected, and that the correlation between them represents HW/SW and HW data processing. The drop in bandwidth utilization happens occasionally during the test execution.

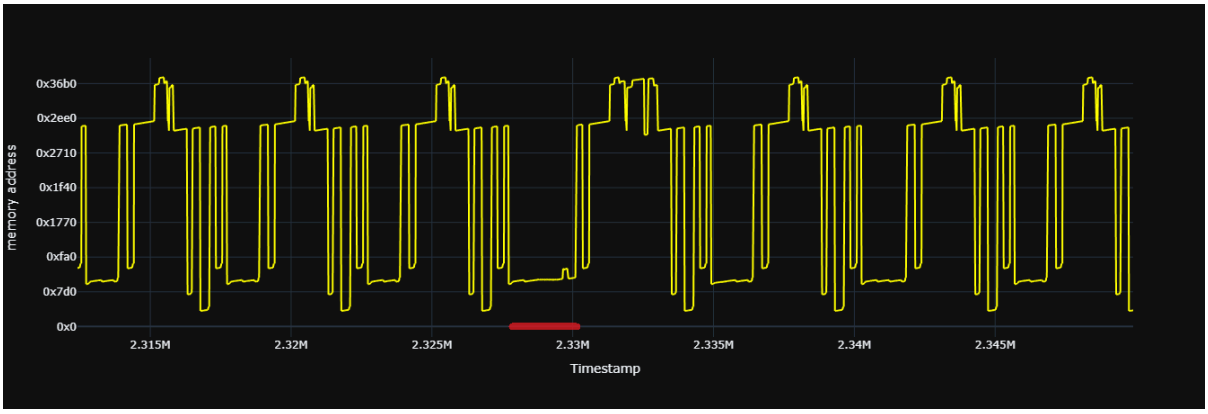


Figure 6 Detected irregularity of memory address values

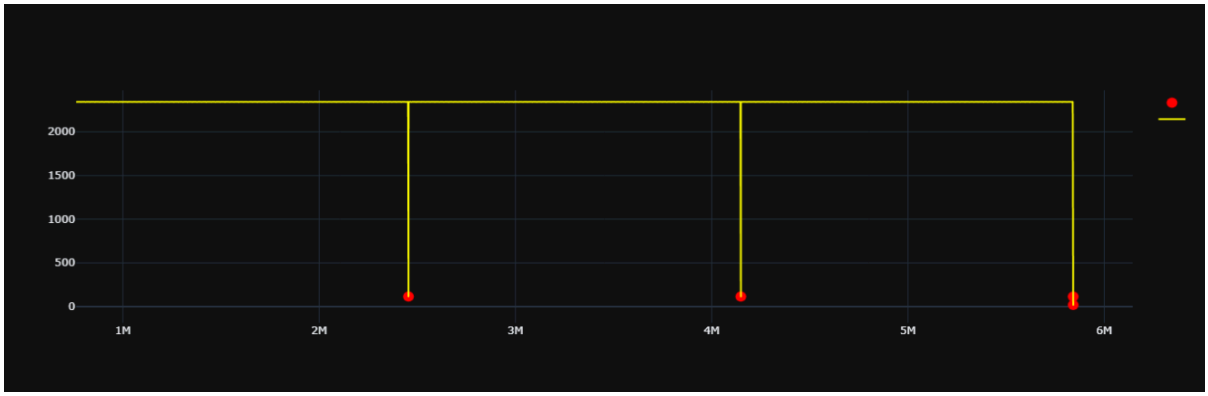

Figure 7 Detected irregularity of transaction duration

## V. Conclusion

This paper presents a new and highly efficient debug approach based on a set of AI algorithms, which is applied to big datasets extracted from simulation output results. It demonstrates the viability of this approach, proven for NoC-based SoC verification through several examples. The major benefits of this approach, applied over the AI-driven platform Cogita-PRO, is the optimized time reduction achieved for the debug process and the improved HW/SW utilization, performance, and throughput issues. The debug techniques are suitable for large-scale complex SoC verification environments and HW/SW co-verification.

In terms of time reduction, AI AC algorithms link all mismatched sources/destinations pairs in the same time span it would have taken standard debugging to link only a single pair. In addition, ML algorithms reveal the exact NoC address translation issues in RTL. When applied to error response transactions, these same ML algorithms classify and cluster the address space ranges, revealing the access to either reserve or unimplemented address spaces. The AI SA algorithms applied on source/destination transfers bring to play valuable transactions distribution. Thus, the new approach provides the utilization of a particular source and/or destination, while disclosing the unused or forbidden accesses in RTL. Analyzing the outstanding transactions utilization, AI algorithms provide insight regarding the RTL throughput and performance. Finding repetitive transaction patterns by AI algorithms points to irregularities detected in the data processing.

Finally, this paper demonstrates the power of AI algorithms for accelerating the debug process, and the applicability of AI algorithms for analyzing bottlenecks, HW/SW correlation, root cause analysis, throughput, and detection of irregularities. Based on these conclusions, it can be asserted that the potential of AI algorithms is limitless for optimizing verification. Further industry research and exploration will ultimately reveal their usage for automatic test generation, bug analysis, requirements versus design implementation crosscheck, and all other areas where verification workloads can be alleviated, as this paper has shown.

## References

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

[2] Harry Foster, "2022 Functional Verification Study," Wilson Research Group and Siemens EDA, https://blogs.sw.siemens.com/verificationhorizons/2022/10/10/prologue-the-2022-wilson-research-group-functional-verification-study/, retrieved on Nov. 18, 2022

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

[4] Dan Yu, Harry Foster, Tom Fitzpatrick, A Survey of Machine Learning Applications in Functional Verification” in Proceedings of Design and Verification Conference (DVCON), San Jose, US, 2023.