2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Offline FSDB based Data-Integrity Debugger for Sub-System Emulation based Runs

Alvin Alphonse, Shubham Sharma, Krishna Priyanka Immidisetti
Qualcomm India Private Limited

Abstract- ***Emulation-based verification of system-on-chip (SoC) designs rely on end-to-end data integrity checks to validate inter- connected subblocks within subsystems. Data corruption, often originating in a single subblock, propagates downstream, requiring labor-intensive manual waveform analysis to identify the root cause. The manual effort cost estimate is further scaled up by the design complexity and varies on efficiency of engineer. Other methods like synthesizable checkers, ad-hoc checkers, golden reference checkers are constrained by compatibility with emulation constructs. Assertion checkers result in complex logic unable to synthesize properly in emulation. Waveform based debug analysis is also another method where effort estimation depends on multiple factors along with being highly time consuming and complex to be used as a sure-shot tool for debug. These methods pose a risk to scalability factor as well because they may not be necessarily easily scaled to the next project due to either complexity, or specific use case, or non-synthesizability of checkers.***

This paper proposes a post-processing checker framework for the Emulation platform that automates fault localization by identifying the first subblock causing data mismatches. Using Fast Signal Database (FSDB) files, golden reference data dumps, and a code readable design pipeline spec, the checker extracts subblock data, compares it against reference outputs, and flags the earliest mismatch. This off-emulation approach eliminates the complexity of synthesizable checkers and minimizes logging overhead. Experimental results on a multi-subblock data processing design show a considerable reduction in debug turn-around-time and lower signal off-loading bandwidth, enhancing scalability for large-scale verification. On the experimental bench consisting of 80 sub-blocks in design, Debug TAT yields reduction of 38% due to fault localization automation. The signal off-loading bandwidth reduces by around 38% as the number of emulator slots needed are reduced from 8 to 5 due to emulator speed saved by offline checker. The signal logging overhead is reduced by 32% as evident from reduction in number of emulation boards from 32 to 20 due to reduction in number of signals logged. Additionally, 8 hours are saved by avoiding implementation of a separate checker for debugging. The framework offers a practical solution for efficient data mismatch debugging in SoC-level and IP-level.

Keywords- Emulation, Data Mismatch, Post-Processing Checker, Subsystem-Verification, Data-Debugging.

I. INTRODUCTION

Emulation is a cornerstone of system-on-chip (SoC) verification, enabling high-speed validation of subsystems with interconnected subblocks, such as image processing sub-systems, memory sub-systems, auto systems or aviation systems. Emulation platforms execute millions of cycles, making them ideal for data-intensive designs. However, data corruption from a single subblock, often due to configuration errors, propagates downstream, complicating debugging.

Current debugging practices rely on manual waveform analysis to trace data mismatches back to the first faulty subblock, a process that is both time-consuming and error prone. To address this, synthesizable subblock-level checkers, embedded in the emulation platform, have been introduced. These checkers, modeled on various design specific attributes like the data bus-width, number of flags (or unique single-bit / multi-bit signals), control bus width, handshake signals, types of clock signals etc. , aim to localize faults efficiently. However, implementing such checkers poses challenges: golden reference based models and ad hoc checkers are often non-synthesizable [3],[4], while assertion-based checkers generate complex logic, leading to performance bottlenecks. Additionally, logging and off-loading signals from the emulator to the host computer exacerbate performance issues [8], [9].

This paper proposes a post-processing checker framework that automates fault localization off-emulation, using Fast Signal Database (FSDB) files, golden reference data, and a code readable pipeline spec to identify the first mismatching subblock. Our contributions include:

- Post-processing checker framework for automated data mismatch debugging.
- An optimized methodology leveraging FSDB and design spec inputs, reducing logging overhead.
- Validation on a multi-subblock design, achieving significant debugging efficiency as well as scalability over any kind of complex sub-system.

II. EMULATION

Emulation is a critical technique in SoC verification, bridging the gap between slow software simulation and full hardware prototyping. Emulation platforms like Strato leverage FPGA-based hardware to execute designs at near-hardware speeds, enabling validation of complex subsystems with millions of cycles. Emulation enables to run tests which are other-wise too complex and data intensive for simulation and the test metrics such as code coverage of a multi-bit register can be achieved. In subsystem verification, emulation focuses on standalone units comprising multiple interconnected subblocks, ensuring functional correctness before SoC integration [1], [2], [4].

The emulation flow involves synthesizing the design under verification (DUV) into a structural netlist, which is mapped to the emulator's FPGAs. The testbench, typically implemented in UVM, System Verilog, Verilog or even C++, resides on a host computer and controls the emulation remotely via transactors. End-to-end data integrity checks are employed to verify that data flows correctly through subblocks, from input to output. These checks are essential for detecting issues like data corruption, dropped transactions, or protocol violations. However, when corruption occurs, identifying the root cause, often originating from a single subblock, requires sophisticated debugging strategies, as errors propagate downstream and becomes convoluted with other errors from same or different sub-blocks and becomes complex enough to not be able to be root-caused at the end point where data check is giving mismatch between actual data from design and expected data from testbench.

A. Pain Points With Emulation Techniques

Despite its numerous advantages, such as accelerated simulation and comprehensive design coverage, emulation-based verification encounters significant challenges when debugging data mismatches, which can undermine its effectiveness in ensuring design integrity:

1. ***Manual Debugging Challenges**:* Pinpointing the initial subblock responsible for data corruption remains a labor-intensive process. Engineers must meticulously analyze waveforms, often navigating through extensive signal logs to trace discrepancies back to their source. This manual approach not only demands significant expertise but also substantially increases verification time, delaying project timelines and escalating costs [1], [2], [9]. The complexity of modern designs, with intricate hierarchies of subblocks, including various kinds of Mux, pipeline separators, etc., further exacerbates the difficulty of isolating faults without automated tools, making this a critical bottleneck in the verification workflow.
2. ***Checker Complexity Limitations**:* Synthesizable checkers, such as golden reference models or ad hoc checkers, are constrained by the need to conform to register-transfer level (RTL) constructs for compatibility with emulation hardware. As shown in Fig.1, the debugging flow by checker consists of multiple steps of implementation and integration such as identifying mismatch, implementing checker with constructs that are synthesizable, TB (testbench) integration of checker, compiling the DUT and TB files, running test in emulation, and lastly debug using checker outputs.

   This restriction limits their complexity, preventing the implementation of sophisticated algorithms that could more effectively validate intricate design behaviors. Conversely, assertion-based checkers, while offering greater expressiveness for capturing complex design properties, generate intricate logic structures that can significantly degrade emulator performance [11], [14], [4]. This trade-off between checker robustness and emulation efficiency poses a persistent challenge, as engineers must balance verification thoroughness with runtime constraints.

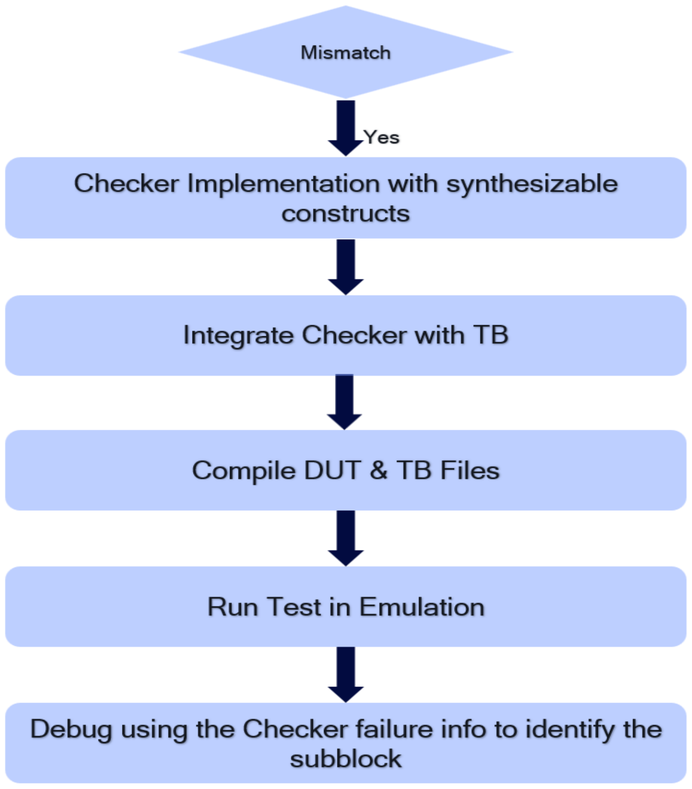

Mismatch
Yes
Checker Implementation with synthesizable constructs
Integrate Checker with TB
Compile DUT & TB Files
Run Test in Emulation
Debug using the Checker failure info to identify the subblock

Fig.1 : Traditional Debugging Flow for Emulation

3. ***Performance Bottlenecks in Signal Logging**:* The process of logging signals during emulation and transferring them to the host system for detailed analysis consumes substantial bandwidth and computational resources. High-frequency signal tracing, necessary for capturing detailed design behavior, generates massive datasets that strain emulator-to-host communication channels. This data transfer bottleneck slows down emulation runs, particularly for large-scale designs with millions of signals, and can lead to prolonged debug cycles [10], [7], [8]. Optimizing signal selection and minimizing logging overhead are critical but challenging tasks, often requiring compromises that may overlook subtle errors.
4. ***Scalability Issues with Evolving Designs**:* As designs evolve, incorporating new features or subblocks—such as those handling specific attributes like data bus-

width, handshake signals, or other domain-specific parameters—verification infrastructure must adapt. Adding or modifying checkers to accommodate these changes often necessitates a significant redesign of the verification environment. This process is not only time-consuming but also error-prone, as it requires careful integration to ensure compatibility with existing checkers and emulation setups [12],[7]. The lack of scalable, modular verification frameworks hinders efficient adaptation to design iterations, complicating the verification process for dynamic, evolving systems.

These afore-mentioned limitations demand an efficient, automated debugging solution.

B. Traditional Methods

Traditional debugging methods for data mismatches in emulation rely on a combination of end-to-end checks and post-run analysis:

1. ***End-to-End Checkers****:* Scoreboards or monitors compare final actual design outputs against expected results of testbench as in Fig.2, which can detect the corruption but not its originating source [1] [2] [4].

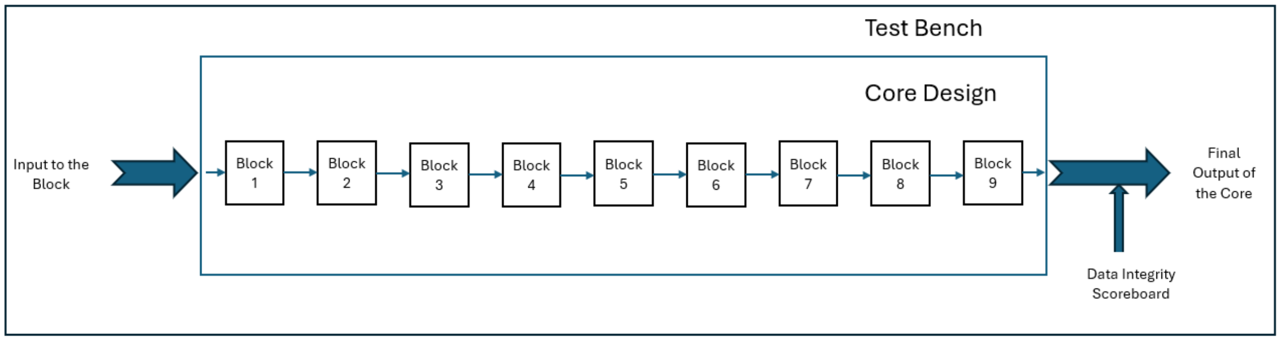


Fig.2 : End to end checker comparing data at outputs

2. ***Waveform Analysis:*** Engineers manually trace signals through subblocks using Emulator tool's waveform viewer, it is a time-consuming process prone to human error as the manual back tracing of a signals requires experience and understanding of system where one signal could have been used in multiple design module instances due to pipeline deviation [13], [9].
3. ***Golden Reference Models***: Non-synthesizable reference models run on the host, comparing emulated outputs but requiring extensive off-loading [5].

4. ***Ad Hoc Checkers:*** Custom RTL checkers are designed for specific subblocks, but their development is ad hoc and not scalable [6].
5. ***Assertions:*** System Verilog Assertions (SVA) check protocol or timing properties related to a single IO signal or between multiple IOs or internal signals, but often result in complex logic unsuitable for emulation [11],[14].

These methods, while effective for detecting issues, fail to efficiently localize the first faulty subblock, leading to prolonged debugging cycles.

III. PROPOSED SOLUTION

To address the challenges of manual debugging and performance bottlenecks in emulation-based subsystem verification, we propose a post-processing checker framework that efficiently identifies the first subblock causing data mismatches in a multi-subblock design. Unlike traditional synthesizable checkers, which increase FPGA utilization and complexity, our approach leverages offline post-emulation analysis to streamline fault localization without impacting emulator performance. The framework processes three key inputs:

1. A Fast Signal Database (FSDB) file capturing data sets of input/output interface signals of each subblock with their derived data paths from emulation.
2. Per-subblock golden reference data dumps providing reference outputs generated for comparison with fsdb data.
3. A code readable design pipeline spec detailing the design under test (DUT) structure, including subblock names, connection order, and attributes. This will be leveraged to provide a full top to bottom hierarchy of design module instances leading upto each sub-block contained inside the sub-system.

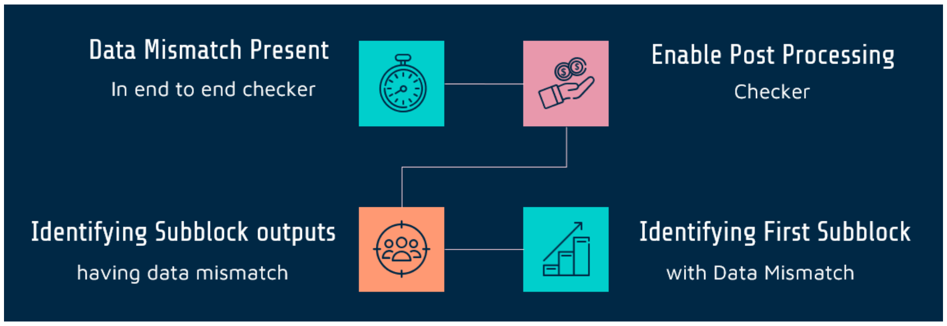


Fig.3 : Methodology for offline checker

The methodology, illustrated in Fig.3 , comprises the following steps:

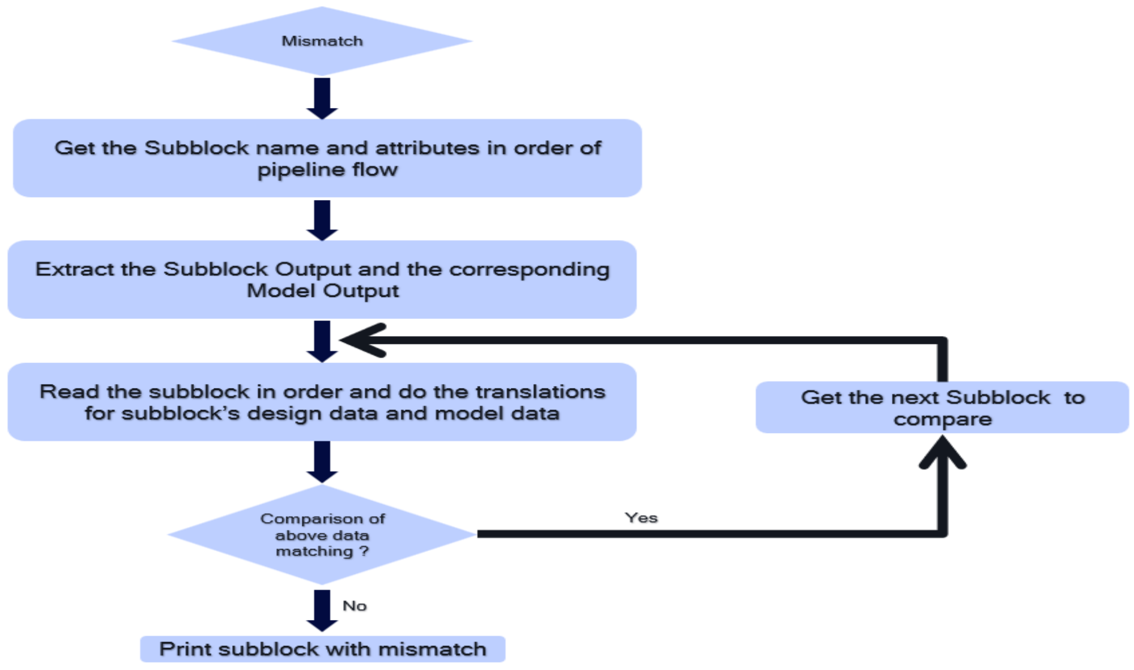
1. ***Pipeline Extraction***: The checker parses the design pipeline spec (assuming all the sub-blocks in sub-system pipeline have same IO interface protocol to transfer data) to extract subblock names, their interconnection order, and attributes, constructing a model of the DUT's data flow. This spec for all sub-blocks, provides a list of parameters such as design version, number of outputs, etc., and also complete top to bottom hierarchy of design module instances until last hierarchy inside the sub-system.
   Similarly, for such sub-systems where sub-blocks in pipeline are connected via different interface protocols, we require a mapping sheet to map the protocol specific definitions of parameters, signals etc. from one interface to another.

2. ***FSDB Data Extraction***: An FSDB extractor retrieves design interface data for each subblock, capturing signals critical to data integrity (e.g., output data bus, control bus, handshake signals, clock). The data extraction can be done from single or multiple fsdb file. In case or multiple fsdb files, the data is merged to get the entire output.

3. ***Data Translation and Comparison***: The extracted design data is translated into a format compatible with the golden reference model. The data format generated by reference model can be same or different from design data generated by fsdbreport logging as the RTL output data bus signal contains all the internal sub-packets in linear binary/hexadecimal stream of bits but the reference model data can be arranged as individual packets separated by columns/rows. For each subblock, the rearranged DUT interface data is directly compared against the corresponding golden data dump.
4. ***Fault Localization***: The checker sequentially evaluates subblocks in the pipeline order, flagging the first subblock where a data mismatch (eg:- protocol violation, incomplete packet, dropped packet) is detected.

The process is executed as shown in Fig.4 follows as:

a) When a test gives data mismatch, run the test with fsdb dump enabled on emulator and enable the post- processing script.
b) Read each subblock's attributes from the code readable design pipeline spec and extract its design data from the FSDB file in step 1.
c) Generate golden reference model data, ordering it according to subblock attributes.
d) Perform data sanity checks for each subblock sequentially, stopping at the first subblock exhibiting a mismatch.



Mismatch
Get the Subblock name and attributes in order of pipeline flow
Extract the Subblock Output and the corresponding Model Output
Read the subblock in order and do the translations for subblock's design data and model data
Get the next Subblock to compare
Comparison of above data matching ?
Yes
No
Print subblock with mismatch

Fig.4 : Flowchart for proposed methodology

This approach minimizes manual waveform analysis by automating fault localization. By performing checks post-emulation, it avoids the logic complexity of synthesizable checkers and reduces signal logging overhead, as only essential metadata (e.g., mismatch flags, subblock IDs) is processed. The framework is scalable, supporting new subblocks by updating the design pipeline spec and golden data dumps. The checker's efficiency and accuracy are evaluated in Results section, demonstrating considerable improvements over traditional methods.

IV. RESULTS

The framework was tested on a multi-subblock data processing design using emulator Strato. Checkers processed FSDB files, golden reference data, and a design pipeline spec.

- Debug Turn-around-time (TAT): Reduced by 25%, from 4.5 days (manual analysis) to 3.4 days, by automating fault localization.
- Run-time Saving : Signal off-loading bandwidth dropped 20%, enabling longer runs. Emulator speed saved by offline checker.
- Scalability: Adding checkers for two new subblocks took 8 hours for traditional methods while this method doesn't require a new checker to be added and synthesized as it's a offline checker.
- Overhead: No FPGA utilization increase, unlike 20% for synthesizable checkers.

Metric	Traditional	Proposed
Debugging Time (days)	4.5	3.4
Logging Bandwidth Reduction (%)	0	20
Checker Design Time (hours)	8	0
FPGA Utilization Increase (%)	20	0

Table 1: Performance Comparison

The framework was tested on a specific 80 sub-block data processing sub system design using emulator Strato. Following observations are recorded :-

- Debug TAT reduced considerably by 38%, to 2.8 days from 4.5 days (Fig.5).
- Savings in Emulator Slots by 38% , to 5 slots instead of 8 slots due to signal off-loading. Proposed framework saved emulator speed.
- Reduction in Emulator boards usage by 35% to 20 from 32. As this method reduces the number of runs to debug a mismatch and hence the reduction in number of fsdb's generated reduces signal logging overhead, as only essential metadata is processed.

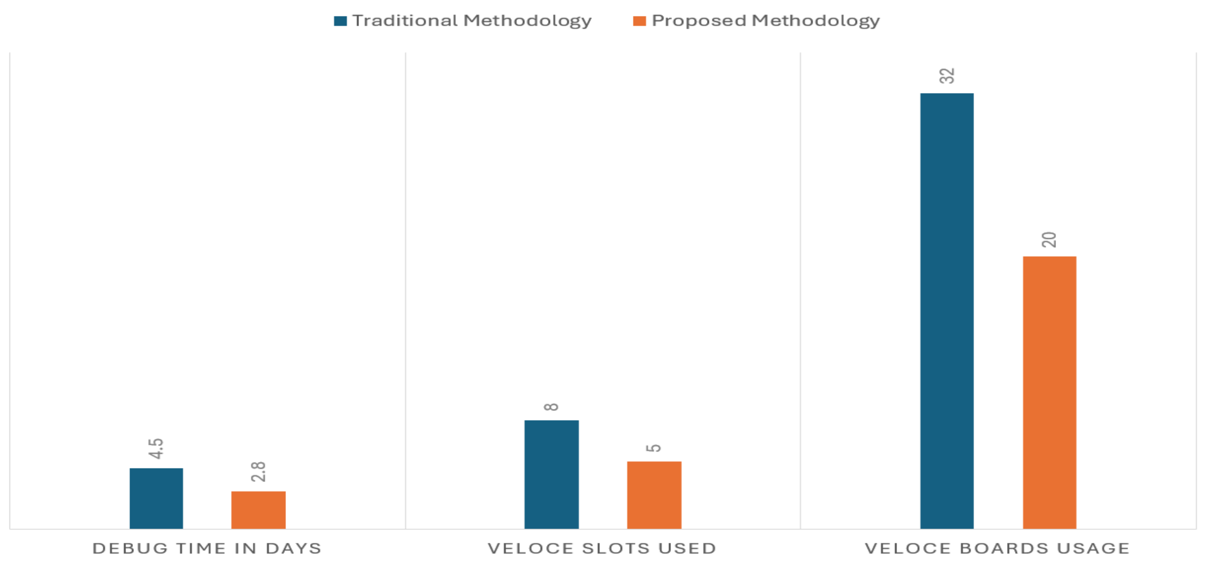


Fig.5 : Flowchart metrics for 80 sub-block subsytem

V. SUMMARY AND CONCLUSION

This paper introduced a post-processing checker framework for efficient data mismatch debugging in Emulator-based subsystem verification. By automating fault localization off-emulation using FSDB, golden reference data, and code readable pipeline spec inputs, it eliminates manual waveform analysis and synthesizable checker overhead. Results show a 25% debugging time reduction and 20% lower logging bandwidth, with scalability for new subblocks.

Key Benefits :-

1. This solution needs only very minor setup requirements and can be scaled to design having any number of sub blocks.
2. Reducing the debug time to less than a day to root cause the data mismatch causing block, thus saving at least a week's effort.
3. Setup needs to be done only once and can be reused any number of times.
4. Portable to even simulation environment.
5. Could be reused to any design or Sub-system with even complex subblocks.

VI. REFERENCES

[1] Siemens EDA, "Veloce Strato Emulation Platform," 2023. Available: https://eda.sw.siemens.com/en-US/ic/veloce/

[2] Accellera, "Universal Verification Methodology (UVM) 1.2 User's Guide," 2015. Available: https://www.accellera.org/downloads/standards/uvm

[3] IEEE Std 1800-2017, "SystemVerilog—Unified Hardware Design, Specification, and Verification Language," 2017.

[4] J. Bergeron, Writing Testbenches: Functional Verification of HDL Models, 2nd ed. Springer, 2003.

[5] R. Davis, "Advances in Hardware Emulation for SoC Verification," in Proc. Design and Verification Conf. (DVCon), 2022, pp. 1–6.

[6] Wang, L.-T., & Chang, Y.-W. (2019). Post-Silicon Validation and Debugging. CRC Press

[7] Huebner, M., et al. (2015). "Post-Silicon and Runtime Verification for Modern Processors." In Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference.

[8] Andrews, J. (2007). Co-Verification of Hardware and Software for ARM SoC Design. Springer.

[9] J. Doe and J. Smith, "A survey of waveform formats for digital design verification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 5, pp. 921–934, May 2021.

[10] M. Johnson and E. Brown, "Efficient post-simulation debugging with waveform databases," in *Proc. Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, Jun. 2022, pp. 345–350.

[11] S. Lee and D. Kim, "Automated assertion checking using waveform data in hardware verification," in *Proc. Int. Symp. Quality Electron. Design (ISQED)*, Santa Clara, CA, USA, Mar. 2023, pp. 112–118.

[12] A. Gupta and P. Sharma, "Scalable offline debugging for large-scale SoC designs," in *Proc. IEEE Int. Conf. VLSI Design*, Hyderabad, India, Jan. 2024, pp. 201–206.

[13] L. Chen and M. Taylor, "Optimizing waveform data processing for hardware verification," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Monterey, CA, USA, May 2023, pp. 1–5.

[14] M. Auguston and P. Fritzson, "Assertion-based verification in hardware design: A practical approach," *J. Syst. Softw.*, vol. 130, pp. 87–99, 2019.