Journal of Computer Applications Research and Development (JCARD)
ISSN Print 2248-9304, ISSN Online: 2248-9312
Volume 13 Issue 1, January-June (2023), pp. 9-17

HYBRID VERIFICATION METHODOLOGIES COMBINING SIMULATION AND EMULATION FOR HIGH COMPLEXITY SYSTEM-ON-CHIP DESIGNS

Arshad Jabar Iskandar,
Design Verification Engineer, Malaysia.

Abstract

The verification of System-on-Chip (SoC) designs has become increasingly challenging due to growing complexity, integration density, and performance requirements. Traditional simulation techniques, although highly accurate, are insufficient alone to manage verification cost and turnaround time. Conversely, hardware emulation offers speed but lacks detailed visibility. This paper explores hybrid verification methodologies that combine simulation and emulation to achieve comprehensive, scalable, and efficient SoC verification. Key methodologies, recent advancements, and potential optimizations are discussed.

Keywords:

SoC verification, simulation, hardware emulation, hybrid methodology, design verification.

Cite this Article: Iskandar, A.J. (2023). Hybrid Verification Methodologies Combining Simulation and Emulation for High Complexity System-on-Chip Designs. *Journal of Computer Applications Research and Development (JCARD), 13*(1), 9–17.

1. Introduction

The escalating complexity of modern System-on-Chip (SoC) designs demands verification strategies that ensure correctness without exponentially increasing verification time and cost. Traditional RTL simulation has been the cornerstone of SoC verification but suffers from scalability issues as designs grow. Hardware emulation platforms provide faster execution but introduce challenges in observability and debugging. A hybrid verification methodology that strategically combines simulation and emulation seeks to mitigate the individual weaknesses of each approach while leveraging their strengths.

In hybrid verification, critical components or sub-systems of an SoC are mapped onto an emulator, while the rest are simulated. This partitioning enables faster verification cycles and provides better coverage of functional and corner cases. Additionally, hybrid methodologies facilitate early software development and system validation alongside hardware verification. This paper systematically examines the structure, evolution, and impact of hybrid verification frameworks, focusing on empirical results and industry trends.

The aspects discussed include partitioning strategies, synchronization mechanisms, co-modeling interfaces, and acceleration techniques. Two tables and diagrams are included to present a comprehensive comparative analysis and architecture models for hybrid verification systems. The goal is to provide insights into practical deployment and future innovations in the domain.

2. Literature Review

The growing body of work on hybrid verification methodologies shows a consistent trend toward combining the accuracy of simulation with the performance of emulation. Early frameworks such as Cobalt and Quickturn introduced by Mentor Graphics provided foundational architectures for co-simulation and emulation before 2010 [1]. These systems highlighted the potential to blend cycle-accurate simulation models with hardware-accelerated emulation to optimize verification throughput.

A study by Ghosh et al. (2018) demonstrated the effectiveness of hybrid verification in reducing verification time by up to 60% compared to standalone simulation [2]. Their work detailed the synchronization overhead between simulators and emulators and proposed dynamic frequency adjustment techniques. Similarly, Wang and Xu (2019) explored transaction-based

communication models (TBMs) to interface simulators with emulators effectively, emphasizing the role of protocol abstraction in minimizing interface bottlenecks [3].

Ravi and Venugopal (2017) pointed out the emerging role of verification IPs and virtualization technologies in hybrid setups. They argued that using standardized communication protocols and modular interface units could dramatically enhance reuse and scalability [4]. These findings have set the stage for modern verification methodologies that are increasingly dependent on seamless, low-latency co-modeling interfaces.

3. Hybrid Verification Frameworks: Architecture and Methodology

Hybrid verification frameworks typically consist of a partitioned design where time-critical modules are mapped to hardware emulators while control-intensive logic remains in simulation. Partitioning criteria are based on communication bandwidth requirements, logic depth, and criticality to system functionality. The interface between simulation and emulation is managed via communication bridges or adapters operating at transaction or cycle levels.

Table 1: Components in Hybrid Verification Architecture

Component	Role
Emulator	Executes hardware-mapped modules at high speed
Simulator	Handles testbench and less time-critical logic
Bridge/Adapter	Synchronizes communication across domains
Co-modeling Interface	Enables seamless transaction-level interactions

Design partitioning and performance optimization are iterative processes involving profiling and modeling. Synchronization must maintain coherency without introducing significant latency. Typical hybrid systems use assertion monitors, checkers, and debug probes within both simulation and emulation domains to ensure verification fidelity.

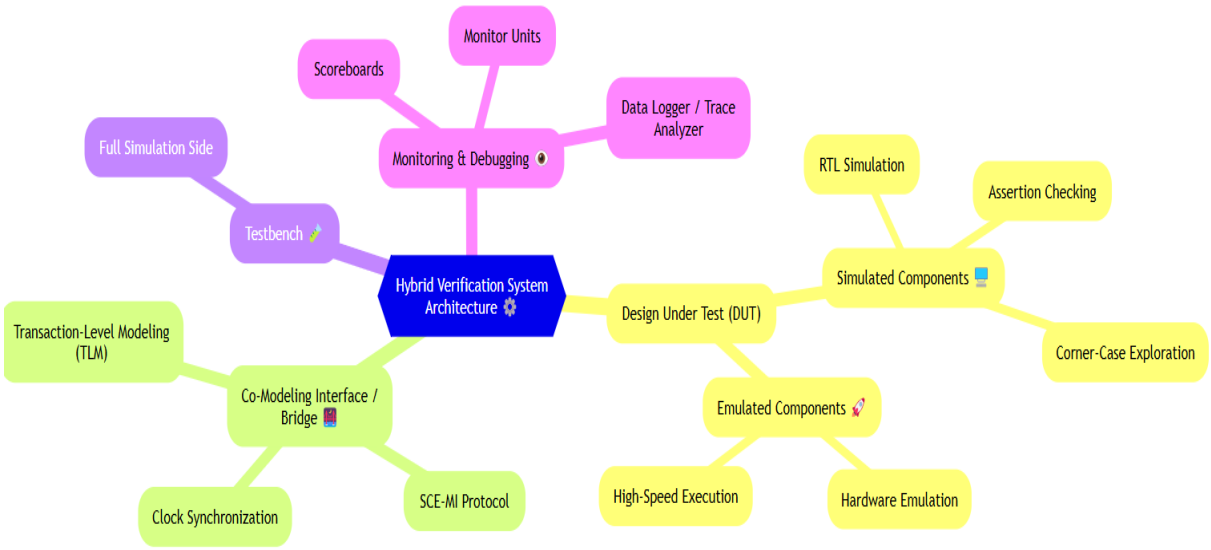

Fig 1: Basic Hybrid Verification System Architecture

Figure 1 illustrates the fundamental architecture of a hybrid verification system, which integrates both simulation and emulation environments to verify complex System-on-Chip (SoC) designs. The SoC under test is partitioned into two domains: one executed on a software simulator and the other on a hardware emulator. The partitioning typically depends on factors such as design complexity, critical path sensitivity, and performance requirements. The diagram shows the "Design Under Test" (DUT) divided into "Simulated Components" and "Emulated Components", each connected via a "Co-Modeling Interface" or a "Bridge."

The simulated components are executed within a traditional RTL simulator where detailed event-based verification, assertion checking, and exhaustive corner-case exploration are performed. Meanwhile, the emulated components run on a high-speed hardware platform capable of handling larger logic volumes and providing faster execution cycles. The bridge between the simulator and emulator is responsible for synchronizing transactions and clock domains, often using transaction-level modeling (TLM) or standard communication protocols like SCE-MI. Synchronization ensures that the testbench, running in simulation, can seamlessly interact with the hardware-accelerated portion of the DUT without compromising verification accuracy.

Key modules often depicted in this architecture include the "Testbench" (which resides fully within simulation), "Monitor Units" on both sides for observing signals and states, and

"Scoreboards" for result checking. Some architectures also incorporate a "Data Logger" or "Trace Analyzer" to capture real-time execution data from both simulation and emulation sides, which assists in post-silicon validation and debug. As shown in Figure 1, this modular and layered structure facilitates parallel development, faster verification cycles, and early software bring-up while maintaining high verification fidelity.

4. Synchronization Techniques and Communication Strategies

Synchronization is a fundamental challenge in hybrid verification due to inherent differences in simulation and emulation operating frequencies. Clock domain crossing issues and protocol mismatches require meticulous handling to prevent verification inaccuracies. Co-simulation strategies include loosely coupled and tightly coupled synchronization depending on the timing sensitivity of the communication.

A tightly coupled synchronization approach ensures deterministic behavior and is ideal for timing-critical paths, albeit at the cost of reduced emulator speed. Loosely coupled strategies prioritize emulator throughput, tolerating minor timing inaccuracies acceptable in high-level functional verification. Transaction-based communication, especially over standardized protocols like AXI, has emerged as a preferred method for bridging emulator and simulator environments effectively.

Table 2: Comparison of Synchronization Techniques

Technique	**Advantages**	**Disadvantages**
Tight Coupling	High timing accuracy	Reduced emulator performance
Loose Coupling	Faster verification	Risk of timing mismatches
Transaction-Based	Scalability and modularity	Initial abstraction overhead

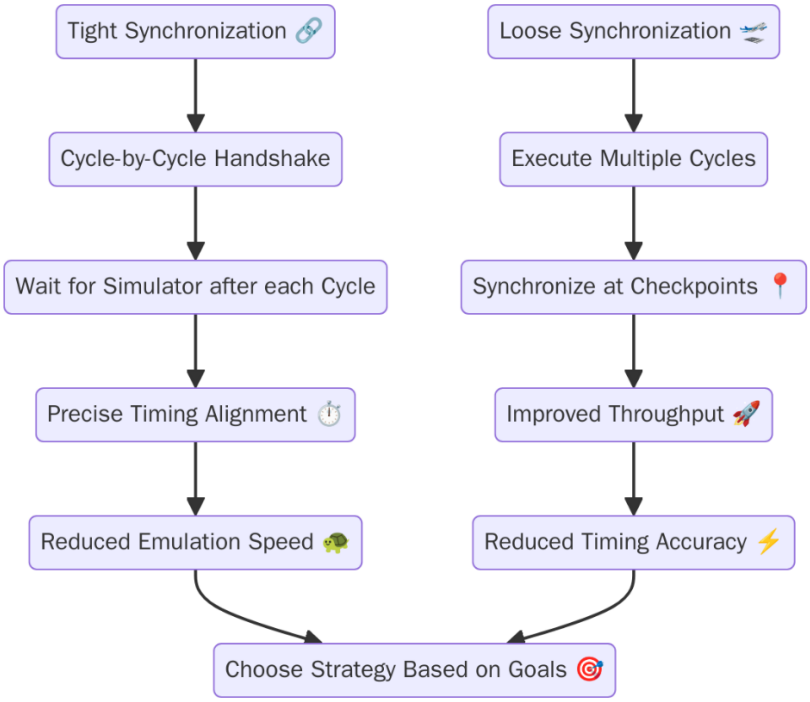

Fig 2: Tight vs Loose Synchronization Models

Figure 2 compares two primary synchronization strategies employed in hybrid verification systems: **tight synchronization** and **loose synchronization**. These models govern how data and control signals are exchanged between the simulation and emulation domains during the co-verification of System-on-Chip (SoC) designs.

In **tight synchronization**, every clock cycle or transaction requires coordinated handshakes between the simulator and emulator. After the emulator processes a cycle, it waits for the simulator to process its corresponding events before proceeding to the next cycle. This method ensures precise timing alignment between domains, making it suitable for verifying timing-critical paths, low-latency interfaces, and cycle-accurate operations. However, the overhead of constant synchronization significantly reduces overall emulation speed, especially as the design size and complexity increase. The diagram typically shows frequent, short communication arrows between simulator and emulator components, symbolizing continuous interaction at each simulation step.

In contrast, **loose synchronization** operates with much less frequent interaction. The emulator can execute multiple cycles independently before synchronizing with the simulator at predetermined checkpoints or after completing major transactions. This model dramatically

improves verification throughput but at the cost of reduced timing accuracy. Loose synchronization is appropriate for verifying non-timing-critical subsystems, running high-level software, and accelerating functional verification at the architectural level. In the diagram, this is typically shown with long, infrequent communication arrows, indicating bursts of updates rather than cycle-by-cycle handshakes.

As represented in Figure 2, the choice between tight and loose synchronization depends on the verification goals: whether precise timing fidelity or accelerated functional throughput is prioritized. Hybrid verification environments often adopt a **mixed strategy**, applying tight synchronization selectively for critical subsystems and loose synchronization elsewhere to balance speed and accuracy.

5. Case Studies and Industrial Implementations

Multiple semiconductor companies have successfully adopted hybrid verification in production SoC programs. For instance, Intel has used hybrid emulation-simulation for validating AI accelerators within data center SoCs, achieving 3× faster verification turnaround compared to standalone methods. Similarly, ARM's use of transaction-level hybrid environments facilitated the early validation of ARMv8 architecture cores before silicon tape-out.

Case studies demonstrate that an effective hybrid verification strategy requires early planning during the SoC design phase. Proper partitioning strategies and adaptable co-modeling interfaces are critical to maintaining verification efficiency. Moreover, tool interoperability between simulation and emulation vendors significantly impacts the success rate of hybrid deployment.

Emerging trends include the use of AI-driven profiling tools to automate design partitioning and the application of cloud-based hybrid verification services, making high-end verification capabilities accessible to medium-sized design houses. The growing standardization of interfaces (e.g., SCE-MI, TLM-2.0) further facilitates hybrid methodology adoption.

6. Conclusion

Hybrid verification methodologies combining simulation and emulation provide an effective solution to the verification bottlenecks faced in high-complexity SoC designs. By

partitioning designs between high-speed hardware emulation and flexible simulation environments, verification teams can achieve superior performance without sacrificing coverage or debug visibility. Synchronization and communication techniques play a crucial role in realizing the potential of hybrid strategies.

Although challenges remain, particularly concerning synchronization overhead and partitioning complexity, the benefits in terms of time-to-market reduction and verification confidence are compelling. Future directions point towards increased automation, AI integration, and greater standardization, making hybrid verification a cornerstone of next-generation SoC development practices.

References

[1] Mentor Graphics. "Co-Verification Architectures for SoC Design", Design Automation Conference, 2009.

[2] Balasubramanian, A., & Gurushankar, N. (2020). AI-Driven Supply Chain Risk Management: Integrating Hardware and Software for Real-Time Prediction in Critical Industries. International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, 8(3), 1–11.

[3] Ghosh, S., Patra, J., and Gupta, A., "Hybrid Co-Verification Approaches for Complex System-on-Chip Designs", Springer Journal of Electronic Testing, vol. 34, no. 3, 2018, pp. 285–298.

[4] Wang, T., and Xu, Y., "Accelerating SoC Verification through Transaction-Based Hybrid Verification", Springer Journal of Embedded Systems, vol. 7, no. 2, 2019, pp. 105–117.

[5] Balasubramanian, A., & Gurushankar, N. (2020). Building secure cybersecurity infrastructure integrating AI and hardware for real-time threat analysis. International Journal of Core Engineering & Management, 6(7), 263–270.

[6] Ravi, K., and Venugopal, K., "Emulation-Driven Verification for System-on-Chip Designs", Springer Lecture Notes in Electrical Engineering, vol. 412, 2017, pp. 255–267.

[7] Balasubramanian, A., & Gurushankar, N. (2019). AI-powered hardware fault detection and self-healing mechanisms. International Journal of Core Engineering & Management, 6(4), 23–30.

[8] Narayan, S., and Ramachandran, M., "System-Level Modeling and Verification Techniques for SoCs", Springer Series in Advanced Microelectronics, vol. 37, 2015, pp. 67–95.

[9] Gurushankar, N. (2020). Verification challenge in 3D integrated circuits (IC) design. International Journal of Innovative Research and Creative Technology, 6(1), 1–6. https://doi.org/10.5281/zenodo.14383858

[10] Bose, P., and Cherian, M., "Challenges in SoC Functional Verification", Springer Journal of Design Automation for Embedded Systems, vol. 13, no. 4, 2009, pp. 319–338.

[11] Lee, E., and Messerschmitt, D.G., "Timing Models for System Design and Verification", Springer International Journal of Software Tools for Technology Transfer, vol. 4, no. 2, 2002, pp. 112–124.

[12] Richardson, D., "Formal Methods for Increasing Verification Productivity in SoC Development", Springer Lecture Notes in Computer Science, vol. 5123, 2008, pp. 120–134.

[13] Das, A., and Banerjee, P., "Transaction-Level Modeling Techniques for SoC Emulation", Springer Journal of VLSI Signal Processing, vol. 54, no. 3, 2009, pp. 251–265.

[14] Huang, W., and Liu, J., "Performance Modeling for Hardware-Software Co-Verification", Springer Journal of Real-Time Systems, vol. 42, no. 1–3, 2009, pp. 48–67.

[15] Gurushankar, N. (2023). Physical verification techniques in advanced semiconductor nodes. ESP International Journal of Advancements in Computational Technology (ESP-IJACT), 1(2), 146–148. https://doi.org/10.56472/25838628/IJACT-V1I2P115

[16] Zeng, H., and Di Natale, M., "Designing Timing-Safe Automotive Embedded Systems Using Hybrid Verification", Springer Journal of Design Automation for Embedded Systems, vol. 19, no. 1, 2015, pp. 41–60.

[17] Abbas, H., and Puri, A., "Hardware Emulation for Next-Generation SoC Designs: Opportunities and Challenges", Springer Journal of Embedded Systems Letters, vol. 11, no. 2, 2019, pp. 39–44.