MANTECH Publications

Journal of Research in VLSI Design Tools and Technology
Volume 10, Issue 3, September-December 2025

Fpga/Asic Co-Design and Hardware–Software Co-Verification: An Integrated Framework for Performance-Driven Embedded System Development

Dr. Priya Menon[1], Mr. Arvind K. Sharma[2]

Associate Professor[1], Assistant Professor[2]

Department of Electronics and Communication Engineering[1], Department of Electrical and Electronics Engineering[2]

National Institute of Technology, Tiruchirappalli, Tamil Nadu, India[1], Indian Institute of Information Technology (IIIT), Allahabad, Uttar Pradesh, India[2]

Email ID: *priyamenon27@gmail.com[1], arvindsharma123@rediffmail.com[2]*

Abstract

Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs) represent two key paradigms in modern hardware design—offering flexibility and performance optimization respectively. As embedded systems grow increasingly complex, integrating FPGA and ASIC workflows through co-design methodologies has become crucial for efficient prototyping and product development. Moreover, hardware–software co-verification ensures system correctness and functional reliability before final silicon implementation. This paper explores the fundamental concepts, methodologies, and challenges associated with FPGA/ASIC co-design and co-verification. It presents a detailed discussion of design partitioning, verification environments, system modeling, and AI-assisted optimization in hardware–software integration. The paper also highlights the scope for future research in automated design frameworks, hybrid verification platforms, and reconfigurable computing systems.

Keywords: *FPGA, ASIC, Co-Design, Co-Verification, Embedded Systems, Hardware Acceleration, System Prototyping, Verification Frameworks, Hardware–Software Integration, Reconfigurable Architectures.*

INTRODUCTION

The increasing demand for high-performance, energy-efficient embedded systems has driven a paradigm shift in hardware design methodologies. Traditional hardware development cycles relied on distinct FPGA-based prototyping followed by ASIC implementation, but the growing complexity of systems-on-chip (SoCs) has blurred the boundaries between the two. FPGA/ASIC *co-design* introduces a synergistic approach where both reconfigurable and custom hardware elements are optimized concurrently to balance flexibility, cost, and performance.

Equally important is *hardware–software co-verification*, which addresses the growing verification bottleneck in SoC design. As modern chips integrate processors, memory, accelerators, and software stacks, early verification becomes critical to detect design bugs and reduce time-to-market. Co-verification provides a unified testing environment that validates interactions between software drivers, operating systems, and hardware components—enabling system-level accuracy before fabrication.

LITERATURE REVIEW

Early FPGA and ASIC Development Paradigms

In the early 2000s, FPGA devices were primarily used for hardware emulation and algorithm validation before ASIC tape-out. ASICs offered higher performance and lower power consumption but suffered from high non-recurring engineering (NRE) costs. Researchers began to combine both technologies, using FPGA prototypes for rapid system validation before ASIC implementation.

Evolution of Co-Design Techniques

The rise of heterogeneous computing and SoC integration led to the evolution of *co-design frameworks*. These frameworks unified hardware and software development, where tasks are partitioned between FPGA-based accelerators and software components. Tools such as Xilinx Vivado HLS, Intel Quartus, and Cadence Stratus HLS emerged, enabling hardware generation from high-level programming languages like C/C++.

Advances in Co-Verification Methodologies

Hardware–software co-verification techniques advanced with the emergence of SystemC and Transaction-Level Modeling (TLM). These models allowed simulation of hardware–software interactions before RTL synthesis. Emulation platforms, such as Synopsys ZeBu and Cadence Palladium, further enhanced verification throughput. More recently, AI-driven testbench generation and formal verification have significantly improved verification coverage and reduced debug time.

FPGA/ASIC CO-DESIGN FRAMEWORK

Table 1: Comparison Between FPGA and ASIC Characteristics

Parameter	FPGA	ASIC
Flexibility	Reconfigurable; can be updated post-deployment	Fixed design after fabrication
Development Cost	Low NRE cost, suitable for prototyping	High NRE cost, optimized for mass production
Performance	Moderate; depends on clock frequency and routing delays	Very high; customized logic and optimized routing
Power Efficiency	Less power-efficient	Highly power-optimized
Time-to-Market	Rapid prototyping and deployment	Longer fabrication and verification cycles

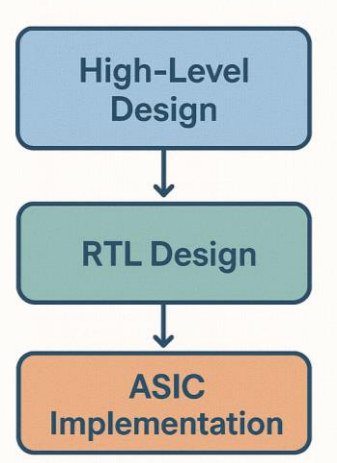

Figure 1: General Architecture of FPGA/ASIC Co-Design Workflow

Design Partitioning

A critical step in co-design is determining which system components should be implemented on FPGA and which should be fabricated as ASICs. High-performance, low-power modules are often targeted for ASIC implementation, while reconfigurable or experimental components are mapped to FPGA. The partitioning depends on factors such as computational intensity, data transfer rate, and real-time constraints.

Hardware–Software Interface Design

The interface between software and hardware is a major determinant of system performance. Efficient communication can be achieved using AXI (Advanced eXtensible Interface) buses or custom interconnects. Software drivers are designed to manage data transfer, synchronization, and control signals between processors and FPGA logic blocks.

Design Space Exploration (DSE)

Co-design leverages DSE techniques to optimize trade-offs between performance, power, and area. Through iterative simulations, designers evaluate multiple architectural configurations. AI-assisted design tools can predict hardware behavior, identify bottlenecks, and recommend optimal mapping of computational kernels.

Hardware Acceleration and Prototyping

FPGAs serve as rapid prototyping platforms for evaluating ASIC-bound modules. FPGA prototypes enable real-time validation, allowing designers to test algorithms, verify performance metrics, and gather runtime data for ASIC synthesis refinement. This process shortens design cycles and minimizes post-fabrication corrections.

HARDWARE–SOFTWARE CO-VERIFICATION METHODOLOGIES

Table 2: Key Components of Hardware–Software Co-Verification Framework

Component	Functionality	Example Tools/Technologies
Hardware Simulator	Simulates RTL models and timing behavior	ModelSim, VCS, Riviera-PRO
Software Simulator	Executes compiled software and monitors API calls	QEMU, ARM Fast Models
Co-Simulation Bridge	Synchronizes hardware and software simulators	SystemC TLM Interface

Component	Functionality	Example Tools/Technologies
Emulation Platform	Real-time verification using FPGA hardware	Cadence Palladium, Synopsys ZeBu
Verification Management	Regression testing and coverage reporting	Jenkins CI, Questa Verification IP

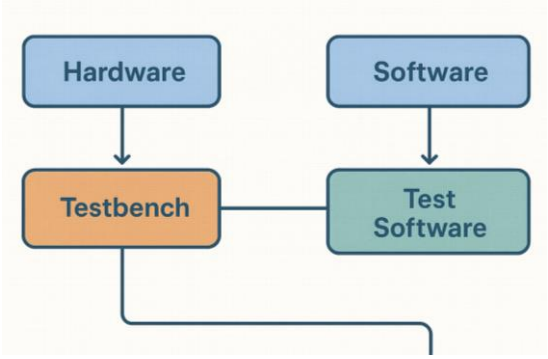

Figure 2: Hardware–Software Co-Verification Environment

As modern SoC (System-on-Chip) designs integrate multiple processing cores, accelerators, and software layers, ensuring the functional correctness of both hardware and software components has become increasingly challenging. Traditional verification, which tested hardware and software separately, is insufficient for today's tightly coupled systems. Hardware–software co-verification provides a unified verification approach where both domains are verified concurrently, allowing designers to detect integration issues early in the development cycle. The following methodologies form the foundation of this co-verification ecosystem.

Co-Simulation Environment

A co-simulation environment enables simultaneous execution of hardware and software within an integrated testbench. In this setup, the hardware model—represented at RTL (Register Transfer Level) or at a higher behavioral abstraction—is simulated using tools such as ModelSim, VCS, or Riviera-PRO. Simultaneously, the software stack runs on an Instruction Set Simulator (ISS), such as QEMU or ARM Fast Models, which emulates the processor behavior at the instruction level.

The synchronization between these two simulators is achieved through Transaction-Level Modeling (TLM) interfaces, often implemented in SystemC. These TLM channels enable high-

speed data exchange between the software and hardware models without requiring signal-level toggling, thus improving simulation performance.

This environment allows engineers to verify that software drivers correctly interact with the hardware registers, test the response of embedded operating systems, and validate interrupt and I/O transactions. Furthermore, co-simulation supports debugging in both domains simultaneously, allowing developers to trace bugs that may arise from timing mismatches, incorrect register mappings, or communication protocol errors.

Emulation and Virtual Platforms

Hardware emulation and virtual prototyping are critical methods for accelerating verification and enabling early software integration.

In hardware emulation, the synthesized hardware design is mapped onto large FPGA arrays to create a cycle-accurate replica of the ASIC hardware. Platforms like *Cadence Palladium*, *Synopsys ZeBu*, or *Mentor Veloce* provide hardware emulators capable of executing complex SoC designs at millions of cycles per second—much faster than pure simulation. This allows the actual software, including operating systems and firmware, to be executed in real time on the emulated hardware, helping identify performance bottlenecks and hardware–software integration errors early.

On the other hand, virtual platforms use abstract, high-level models to simulate the behavior of entire SoCs without requiring RTL-level detail. They enable early software development long before the hardware design is complete. Virtual models of processors, peripherals, and interconnects can run compiled code, allowing engineers to test drivers, bootloaders, and system initialization routines.

The combined use of emulation and virtual platforms creates a hybrid verification environment, balancing speed and accuracy—emulation validates the final hardware timing, while virtual platforms focus on early software validation and functional testing.

Formal Verification and Assertion-Based Verification (ABV)

While simulation-based techniques provide coverage of functional scenarios, they cannot guarantee complete correctness for all possible states in a complex design. Formal verification

addresses this limitation by using mathematical methods to prove the correctness of design properties against a set of logical assertions.

In this process, formal verification tools automatically explore all possible input combinations and internal states, ensuring that specified conditions—such as data integrity, protocol compliance, or deadlock-free communication—are always satisfied. This approach is especially useful for verifying control logic, finite state machines (FSMs), and inter-module communication protocols.

Complementary to formal methods, Assertion-Based Verification (ABV) employs user-defined properties written in SystemVerilog Assertions (SVA) or PSL (Property Specification Language). These assertions are embedded in the RTL design or testbench and actively monitor hardware behavior during simulation or emulation.

When combined, formal verification and ABV provide a comprehensive correctness assurance. Formal tools use assertions as constraints to prove logical consistency, while simulations check for violations dynamically. This hybrid approach enables early detection of design bugs, protocol mismatches, and interface timing violations, particularly in hardware–software communication channels.

Regression and Coverage Analysis

Verification of complex SoCs is not a one-time activity but a continuous, iterative process. To ensure that new modifications do not introduce unforeseen errors, regression testing is performed throughout the design cycle.

Regression involves repeatedly executing an extensive suite of test cases whenever the design or software is updated. Automation frameworks like Jenkins, Questa Verification Management, or UVM (Universal Verification Methodology) are employed to automate these test runs, compare results, and generate reports.

A crucial aspect of verification closure is coverage analysis, which measures how thoroughly the design has been tested. Coverage can be classified into:

- **Functional Coverage:** Verifies that all functional scenarios and design features have been exercised.
- **Code Coverage:** Measures the extent of executed RTL code, including statements, branches, and conditions.
- **Assertion Coverage:** Determines whether all embedded assertions have been triggered during simulation.

High coverage percentages indicate that the verification process is exhaustive, minimizing the probability of undiscovered defects. Automated tools track coverage metrics, identify untested portions of the design, and generate targeted test cases to close verification gaps.

Together, regression testing and coverage analysis form the backbone of verification quality assurance—ensuring that the integrated hardware–software system performs reliably under diverse operational conditions.

CHALLENGES IN FPGA/ASIC CO-DESIGN AND CO-VERIFICATION

Table 3: Challenges and Mitigation Strategies in Co-Design and Co-Verification

Challenge	Impact	Mitigation Strategy
Design Partitioning Errors	Suboptimal performance and resource imbalance	AI-guided design partitioning and profiling tools
Tool Incompatibility	Integration delays and manual conversions	Use of standard exchange formats (SystemC, IP-XACT)
Verification Bottleneck	Slow regression cycles	Parallel verification and cloud-based test automation
Debugging Complexity	Reduced visibility in emulation	Use of trace buffers and transaction-level debugging

Design Complexity and Heterogeneity

As SoCs integrate multiple IP cores, memories, and accelerators, ensuring compatibility between FPGA and ASIC components becomes complex. Co-design requires accurate modeling of timing, synchronization, and power behavior across heterogeneous elements.

Verification Bottlenecks

The verification process often consumes over 70% of total design time. Maintaining synchronization between hardware simulators and software execution models is challenging, particularly for large-scale systems.

Tool Integration and Standardization

The ecosystem of co-design and co-verification tools is fragmented. Different vendors provide distinct design environments with proprietary formats, making interoperability difficult.

Performance vs. Flexibility Trade-Offs

While FPGAs offer adaptability, they are limited in frequency and power efficiency compared to ASICs. Co-designers must carefully balance flexibility during prototyping with performance requirements for final deployment.

Debugging and Visibility Issues

Observability in FPGA-based emulation is often limited due to restricted probe points and on-chip memory. Debugging co-simulated environments that include both hardware and software requires sophisticated trace analysis and debugging tools.

SCOPE AND FUTURE TRENDS

AI-Driven Co-Design Automation

Machine learning models are increasingly being integrated into EDA workflows to automate partitioning, placement, and timing optimization. Predictive models can estimate design performance early, guiding architects to efficient configurations.

Reconfigurable SoCs (RSoCs)

The future of co-design lies in hybrid SoCs that embed both programmable logic and ASIC-grade processing cores. Devices such as Xilinx Zynq and Intel Agilex exemplify this convergence, enabling runtime reconfiguration for adaptive applications.

Cloud-Based Verification Environments

With the rise of cloud computing, hardware–software co-verification can now leverage distributed resources for faster regression testing and parallel simulation. Cloud-based verification platforms enhance scalability and reduce hardware costs for large teams.

Formal–Emulation Hybrid Frameworks

Combining formal verification with emulation offers a powerful solution to achieve both accuracy and scalability. Formal proofs ensure functional correctness, while emulation validates system-level behavior in near-real-time conditions.

Energy-Aware Co-Design

Power efficiency is a primary concern in embedded systems. Future FPGA/ASIC co-design methodologies will incorporate dynamic power modeling and adaptive voltage scaling, optimizing designs for both performance and energy usage.

CASE STUDIES AND APPLICATION DOMAINS

High-Performance Computing (HPC)

FPGA/ASIC co-design has found extensive use in accelerating computational kernels in HPC workloads such as matrix multiplication, deep learning, and cryptography. Co-verification ensures functional correctness before large-scale hardware deployment.

Automotive and Aerospace Systems

Safety-critical domains like autonomous vehicles and avionics leverage FPGA prototyping to verify control algorithms. ASICs are later deployed for production due to their robustness and power efficiency.

Telecommunication Systems

With the evolution of 5G and 6G architectures, FPGA–ASIC hybrid designs are employed for baseband processing and beamforming. Co-verification frameworks ensure interoperability across complex communication protocols.

Healthcare and Biomedical Devices

In medical instrumentation, FPGA-based rapid prototyping enables early validation of diagnostic algorithms, while ASICs provide reliable deployment for continuous operation. Hardware–software co-verification ensures safety and compliance with regulatory standards.

CONCLUSION

FPGA/ASIC co-design and hardware–software co-verification have emerged as critical enablers for the next generation of embedded and cyber-physical systems. By integrating

reconfigurable prototyping with custom hardware synthesis, designers achieve the optimal balance between flexibility and performance. Meanwhile, co-verification frameworks ensure early detection of design flaws, reducing costly post-fabrication errors.

As design complexity continues to escalate, AI-assisted tools, hybrid verification models, and reconfigurable SoCs will redefine the design landscape. The convergence of FPGA prototyping, ASIC optimization, and intelligent verification workflows will drive innovation in domains ranging from aerospace to artificial intelligence, shaping the future of hardware design methodology.

REFERENCES

1. Amuru, D., Vudumula, H. V., Cherupally, P. K., Gurram, S. R., Ahmad, A., Zahra, A., & Abbas, Z.
   AI/ML Algorithms and Applications in VLSI Design and Technology (2022). A comprehensive review on applying AI and machine learning to automate and optimize VLSI design and manufacturing tasks. arXiv
2. Fang, W., Wang, J., Lu, Y., Liu, S., Wu, Y., Ma, Y., & Xie, Z.
   A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA (2025).
   Survey paper covering foundation AI models for circuit design and EDA. arXiv
3. Patra, A., Rout, S., & Ravindran, A.
   AiEDA: Agentic AI Design Framework for Digital ASIC System Design (2024). Proposes an autonomous AI-based design framework to integrate HDL generation and tool workflows. arXiv
4. Ren, Y., Peng, B., Xue, C., Guo, K., Wang, Y., Cheng, G., Lin, Y., Zhang, L., & Sun, G.
   Orthrus: Dual-Loop Automated Framework for System-Technology Co-Optimization (2025).
   Introduces a system-technology co-optimization framework improving delay and power metrics in VLSI design. arXiv
5. Jiménez, A., & Muñoz, A.
   Very-Large-Scale Integration (VLSI) Implementation and Performance Comparison of Multiplier Topologies for Fixed- and Floating-Point Numbers (2025). *Applied*

Sciences.

Compares array, Wallace tree, and Booth multipliers in a VLSI design context. MDPI

6. Attaoui, Y., Chentouf, M., & Ismaili, Z. E. A.
   Machine Learning in VLSI Design: A Comprehensive Review (2024).
   A review on state-of-the-art machine learning applications in VLSI CAD/EDA workflows. JICS
7. Amuru, D.
   AI/ML Algorithms and Applications in VLSI Design and Technology — Elsevier paper (cited ~98 times).
   Discusses AI/ML trends in automating various aspects of VLSI design and manufacturing. ScienceDirect
8. (IJIRT) Design for Testability in VLSI Systems
   Design for Testability in VLSI (2024).
   Examines how testability techniques are integrated into VLSI design flows for better fault coverage. IJIRT
9. ScienceDirect: AI/ML in VLSI (Amuru et al.)
   AI/ML algorithms for VLSI design and manufacturing — highly cited review in EDA automation research. ScienceDirect
10. (Optional foundational reading) Weste & Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*.
    Though not a journal article, this textbook remains a classic reference for VLSI design fundamentals often cited in research. www.slideshare.net