Software-Hardware Co-Verification for Traditional Verification Frameworks

Fangyuan Song[1], Yunlong Xie[1] and Jincheng Liu[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences

Abstract

As RISC-V architectures rapidly evolve with customized extensions, traditional hardware verification methods face growing interoperability challenges with modern software ecosystems. While the UVM framework remains dominant for chip-level verification, its tight integration with SystemVerilog creates barriers when interfacing with high-level languages like Python or Java that dominate AI/ML-driven verification workflows. Current cross-domain solutions such as DPI-C and VPI suffer from low abstraction levels, requiring manual translation of hardware signals into software data structures, which introduces significant overhead and limits scalability. This paper presents VeriBridge, a co-verification framework that introduces a hardware-optimized messaging protocol to bridge the semantic gap between HDL and HLL domains. By implementing transaction-level message queues compatible with RISC-V memory attributes and automated interface generation for multiple languages, our solution achieves 4.1× faster data exchange compared to conventional methods while reducing interface code complexity by 92%. Experimental results demonstrate efficient validation of RISC-V vector extensions through seamless integration between UVM testbenches and Python-based neural stimulus generators.

Introduction

UVM has emerged as the de facto standard for hardware verification, providing a robust framework for verifying complex System-on-Chip (SoC) designs[1]. By offering structured testbenches, reusable components, and a standardized verification flow, UVM has significantly enhanced the efficiency and scalability of hardware verification[2].

As modern verification workflows evolve and proliferation of RISC-V specialized extensions, verification has transformed hardware verification into a multi-domain challenge requiring coordination between chip designers and software developers. Many verification tasks today benefit from software-driven methodologies, leveraging high-level languages such as Python and C++ to enhance automation2, improve debugging, and enable moreadvanced verification strategies. Despite UVM ' s dominance in hardware-centric workflows, it faces several challenges when interacting with the broader software ecosystem, UVM is inherently tied to SystemVerilog, which restricts direct access to the extensive libraries and frameworks available in software-oriented languages like Python and C++. Many advanced verification techniques—such as AI-driven test generation, real-time data analytics, and performance modeling —depend on these external tools. The inability to directly interface with these software ecosystems often results in inefficient workflows that require workarounds or custom bridges, increasing complexity[3].

To enable seamless hardware-software co-verification, efficient cross-domain interoperability between programming paradigms is imperative. Current industry-standard interfaces like DPI-C and VPI exhibit fundamental limitations: 1)when integrating Python/C++ test components through DPI-C interfaces. This inefficiency stems from fundamental mismatches between hardware description languages' cycle-accurate semantics and software languages' event-driven paradigms.2) The inherent limitation of SystemVerilog in cross-language interoperability compels adoption of C/C++ shim layers, where developers must painstakingly craft language-dependent C function calling conventions.3)A paradigm discrepancy exists between hardware description languages (HDLs) and software-oriented high-level languages (HLLs).

We addresses these challenges through a three-pillar approach: hardware-aware message routing that respects RISC-V physical memory protection schemes, inversion of control to enable software-driven verification workflows, and automatic synthesis of type-safe language bindings. This strategy ensures:Seamless Integration:UVM transactions interact bidirectionally with High Level Language frameworks, enabling advanced capabilities such as machine learning-driven test generation using PyTorch or real-time visualization with Matplotlib. Backward Compatibility: Existing UVM testbenches and RTL designs remain unmodified but are encapsulated as reusable software components, preserving prior investments. Ecosystem Leverage: Software teams can use familiar tools such as Pytest for co-simulation, debugging, and extending UVM test environments, fostering cross-domain collaboration. By bridging the gap between traditional UVM workflows and modern software-driven methodologies, this approach enhances verification efficiency.

Methodologies

Data path & verifaction agent

At the core of VeriBridge lies a unified messaging architecture that translates UVM transactions into portable message packets. These packets encapsulate both verification payloads and RISC-V-specific metadata such as privilege levels and memory access attributes, enabling direct mapping to processor bus protocols. A shared memory buffer architecture eliminates redundant data copying between hardware simulations and software processes, while an event-driven control plane allows Python/Java components to asynchronously trigger verification scenarios. The framework's compiler automatically generates optimized interface code by analyzing UVM sequence definitions and target language semantics. For instance, when integrating a Python machine learning model, VeriBridge synthesizes Python native extensions that directly manipulate message queues through RISC-V virtual memory addresses. This approach maintains cycle accuracy for hardware verification while exposing transaction-level interfaces to software components.

We systematically encapsulate the aforementioned co-verification mechanisms as UVM-compliant Agent components, ensuring backward compatibility with legacy testbenches. Users only need to instantiate this Agent to seamlessly integrate UVM with the software environment. This agent is structured as two decoupled modules:

- Driver: Parses transactions into byte streams and transmits them to the Monitor via TLM.
- Monitor: Receives byte streams from the Driver,decodes them into transactions, and provides them for user access.

Software-Driven Verification Orchestration

we inverts traditional verification control flow by enabling software agents to programmatically command hardware testbenches. To address this, we introduce a step() function, allowing HLL to drive UVM simulation in a controlled manner. The step() function triggers a UVM simulation step via reactive programming interfaces compatible with Python asyncio and Java CompletableFutures, software components dynamically construct verification scenarios using real-time coverage feedback. This enables fine-grained control over UVM execution, allowing HLL to dictate simulation progress dynamically. For example, a reinforcement learning model implemented in PyTorch can analyze branch coverage metrics streamed via message queues, then instantiate targeted UVM sequences to probe unexplored RISC-V instruction combinations.

Multi-Language Extension

While the core messaging layer handles RISC-V-specific data marshaling, language interoperability is achieved through a SWIG-mediated binding generator. Unlike conventional DPI-C approaches that require per-language callback stubs, VeriBridge employs a two-stage interface synthesis:

1. UVM transaction types are transpiled to C struct ures annotated with RISC-
   V memory alignment pragmas, preserving cache line optimization across software domains.
2. Target-specific SWIG interface files (.i) apply semantic transformations

Discussion

VeriBridge's impact extends beyond verification acceleration. By providing a stable abstraction layer between hardware and software domains, it enables novel workflows like AI-guided test generation and cloud-native verification orchestration. Early adopters report 68% reduction in test development time when validating RISC-V cryptographic extensions through TensorFlow integration. The framework's architecture-aware messaging proves particularly valuable for heterogeneous systems, where it automatically adapts transaction granularity between scalar cores and vector accelerators. However, challenges remain in formal verification of message protocols and support for distributed chiplets. Future work will integrate RISC-V CoreSight trace capabilities to enable cross-domain debugging and expand the interface generator to support emerging languages like Rust. These enhancements aim to establish VeriBridge as a foundational layer for next-generation co-design ecosystems, ultimately accelerating RISC-V adoption through improved verification agility.

References

[1] Siemens EDA. 2020. 2020 Wilson Research Group functional verification study. https://resources.sw.siemens.com/en- US/white- paper- 2020- wilson- researchgroup- functional- verification- study- ic- asic- fucntional- verification- trendreport. Online.

[2] A2023. IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language. https://standards.ieee.org/standard/1800-2023.html.

[3] Juan Francesconi, J Agustin Rodriguez, and Pedro M Julian. 2014. UVM based testbench architecture for unit verification. In 2014 Argentine Conference on Micro-Nanoelectronics, Technology and Applications (EAMTA). IEEE, 89–94.