UnityChip Verification: Scale Out Hardware Verification with Software Developers

Abstract

Hardware verification represents a major portion of chip development. While prior work focuses on accelerating verification (Scale-Up), leveraging software developers participation (Scale-Out) remains key. Existing tools face three barriers when verifying the complex hardware designs: (1) timing alignment between software-native event-driven programming and hardware execution paradigm, (2) reusing hardware verification IPs (VIPs) in software environments, and (3) performance-debugging trade-offs in open-source simulators.

This paper presents *UnityChip Verification*, a multi-language hardware verification platform with software-native optimization designed for software developers, enabling software developers to adopt event-driven programming or synchronous methods while reusing hardware VIPs. Key innovations include: (1) a software-native clock scheduler that aligns software events with hardware timing; (2) cross-environment transaction/event mapping techniques that facilitate VIP reuse; and (3) non-intrusive simulator enhancement that balances performance, debuggability, and scalability. Evaluated on the XiangShan and Rocket-Chip RISC-V processor, our framework achieves **up to 20× runtime speedup** and **76%** memory savings over Cocotb, offers **zero-overhead support** for C++, Python, and other languages, and reduces code by **12%** and accelerates speed by **16.6%** while reusing VIPs.

Keywords

Hardware Verification, Software Testing, Open-Source Hardware

1 Introduction

Hardware verification plays a pivotal role in chip development, taking approximately 70% of the total project duration [38]. In particular domains, such as CPU design, the ratio of verification engineers to design engineers can even reach 5:1. The high demand for verification has driven research aimed at improving efficiency, making it a key topic in current studies.

In the past, the industry mainly focused on speeding up the verification process to improve efficiency (Scale Up). From enhancing reusability with Universal Verification Methodology (UVM) [17] to accelerating the development of reference models with SystemC [1], such work has improved the development efficiency of hardware verification engineers. However, there is more than one way to boost efficiency. Lowering the barriers to enable more people to participate in hardware verification with ease is also a promising path to improve efficiency (Scale-Out). Tools like Cocotb [33], PyMTL [24], Fault [41], and ChiselTest [28] have attempted to leverage high-level programming languages, such as Chisel and Python, to improve verification productivity. They have also attracted software developers to step into the hardware verification area [15].

However, software developers still face several challenges when leveraging these tools to build their verification environments from the ground up.

❶ **Paradigm mismatch.** Both hardware design simulation and transaction-level verification [1, 2] are *event-driven*. However, current tools like PyMTL and ChiselTest do not provide event support for verification. Although Cocotb implements a simple event scheduler using simulator callbacks, it risks reading erroneous data [16] by awakening software programs before hardware stabilization, which is due to the delta cycles [11] characteristic of simulator iteration convergence. Therefore, there is a need for software-native support for Event-Driven Programming (EDP) to bridge the mismatch and assist software developers in handling hardware ports with complex timing at appropriate moments.

❷ **Legacy IPs compatibility.** Software languages cannot handle the hardware event and transaction transmission. However, most industry-proven verification IPs (VIP) are fundamentally based on event and transaction drivers, leading to inherent technical incompatibility. For instance, although PyUVM [31] is a Python-level reimplementation of UVM, it cannot support traditional SystemVerilog UVM-based VIPs. Developers ultimately face a dilemma between redeveloping reference models or manually building complex communication relays, which is a huge burden for software developers.

❸ **Performance or debuggability.** Open-source hardware tool chains trade off performance for functionality in their design. For example, even the state-of-the-art simulator Verilator [40] faces conflicts between simulator optimization and Verilog Procedural Interface (VPI). Enabling VPI's internal signal debugging capability results in at least a 70% performance loss and doubles the program size. Furthermore, some simulators [7, 9, 43, 48] currently do not support VPI Force/Release like functionality for locking hardware states, which additionally limits various debugging methods.

To address these issues, we propose UnityChip Verification (UCV), a software-native, multi-language hardware verification platform designed to enhance efficiency and accessibility in chip verification workflows. It enables developers to write cycle-accurate verification cases using mainstream software languages (e.g., Python, Java, C++) through synchronous or event-driven programming modes, while seamlessly integrating hardware events and transaction-level communication into software environments. Additionally, it implements a low-overhead simulator enhancement layer that adds internal signal debugging capabilities without sacrificing performance or modifying existing simulator code. Specifically, we make three technical contributions to achieve these goals.

① **Software-native timing and interaction.** To support EDP in software-based hardware verification, we need to synchronize between software event schedulers and hardware cycles. While simulators employ virtual time to sequence hardware events during

emulation, software events unmanaged by simulators cannot be properly ordered. To resolve this, we designed a *clock type* that implements event scheduling through software-native asynchronous libraries while recording simulation time in software for event ordering. During runtime, this clock controls simulator execution until reaching target timestamps and processes software events after hardware convergence.

② **Transparent hardware-software mapping.** Reusing existing hardware VIPs requires cross-environment sharing of exclusive resources. Event handling and transaction transfer tasks require exclusive access, preventing concurrent processing by software and hardware. This conflict may induce resource contention and potential deadlocks. UCV introduces a registry-based synchronization mechanism that constructs mirrored objects across different environments and synchronizes states during switch operations.

③ **Software-defined simulator enhancement.** Modifying the simulator itself or its generated artifacts entails prohibitive development and maintenance costs. To circumvent this, UCV adopts external software augmentation modules for feature integration. Specifically, UCV implements a data port type in software that directly accesses simulator memory, while abstracting VPI-related logic into reusable software code to enable on-demand loading.

We evaluated the effectiveness of UCV on open-source processors XiangShan [45] and RocketChip [6]. The results show that EDP is the preferred method for software developers during verification, with acceptable overhead across different languages. While preserving internal signal debugging capabilities, our approach achieves up to 20× speedup and 76% memory savings compared to Cocotb and similar solutions. Furthermore, when reusing VIPs, it demonstrates 16.6% higher throughput and 12% reduction in lines of code (LOCs) compared to manual relay methods. Most importantly, unlike Cocotb's compromised compatibility in event-driven scenarios, UCV maintains full compatibility with existing software ecosystems.

Our contributions can be summarized into four aspects:

- We developed a software-native hardware verification platform that integrates software developers using different languages, enhancing development efficiency and execution speed through optimized software components.
- We designed a software asynchronous runtime-based hardware clock to assist event-driven programming (EDP) verification in software environments.
- We proposed a cross-environment mapping method for events and transactions, leveraging existing industry-level VIPs and auxiliary tools to enable complex, real-world verification in software.
- We constructed an on-demand dynamic loading-based data debugging method that enhances performance and scalability while preserving simulator debuggability.