Hassert: Hardware Assertion-Based Agile Verification Framework with FPGA Acceleration

Ziqing Zhang[1,2], Weijie Weng[1], Yungang Bao[1,2] and Kan Shi[1,2]

[1]SKLP, Institute of Computing Technology, CAS [2]University of Chinese Academy of Sciences

Abstract

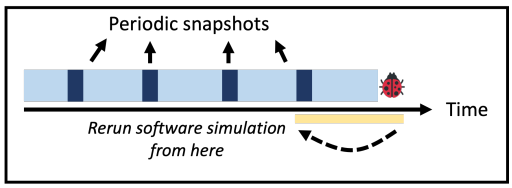
Functional verification is typically the bottleneck of the chip development cycle, mainly due to the burdensome simulation and debugging process using software simulators. For RISC-V, verification becomes even more critical to support a wide range of applications and extensions. Assertion-Based Verification (ABV) has been widely adopted to provide better visibility and detect unexpected behaviors. While ABV enhances efficiency but is limited to slow software simulations. FPGA prototyping offers faster alternatives but lacks fine-grained debugging for error analysis. To address these challenges, we present Hassert, an efficient ABV framework that combines high-performance verification on FPGAs with fine-grained debugging in software. Hassert automates the scheduling and mapping of SystemVerilog Assertions (SVAs) to the FPGA, allowing for extensive hardware testing. To further improve debugging efficiency, Hassert enables dynamic switching of assertions and μArch-guided snapshot based on the assertion. We demonstrate that these contributions significantly enhance verification efficiency over software simulations for various designs, with minimal area overhead and full debugging visibility.

Introduction

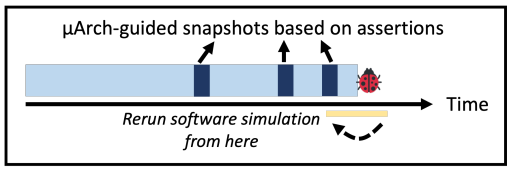
Functional verification is a major bottleneck in chip development, consuming up to 70% of effort, largely due to the slow nature of RTL software simulation. While suitable for early stages, it becomes inefficient for large benchmarks. FPGA-based testing offers faster performance but has limited debugging capabilities. Tools like ILA can monitor only a few signals, forcing engineers to rerun simulations in software to trace root causes. Moreover, while ISAs like RISC-V are highly configurable and adaptable to diverse acceleration scenarios, these advantages also create further needs for more efficient verification methods.

Assertions enhance observability by enabling monitoring at any design level and proactively identifying potential bugs to prevent future issues, saving time and effort. Unfortunately, assertions are unsynthesizable structures and cannot be directly used in FPGA prototyping. Additionally, some software simulators like Verilator do not natively support assertions.

We introduce Hassert, a hardware assertion-based agile verification framework that can address verification scenarios for both RISC-V processor cores and accelerators. Hassert integrates SystemVerilog Assertions (SVAs) with the DUT on FPGA fabric, alongside a reference model on CPUs. This enables efficient DUT testing over large benchmarks with significant performance gains compared to software simulation. Hassert combines automatic self-checking at the system and μArch levels using the reference model and assertions. It includes an open-source library for hardware-equivalent versions of unsynthesizable SVAs and supports existing SVA implementations. To further enhance debugging, Hassert leverages FPGA partial reconfiguration (PR) to support dynamic assertion switching, cutting down the re-compile time and the area overhead. Moreover, Hassert supports μArch-guided snapshots as in Figure 1(b), avoiding performance penalties from periodic ways in Figure 1(a). These snapshots can be offloaded to simulators like ModelSim for deeper analysis.



(a) *Previous approach with hardware snapshots.*



(b) *Proposed verification flow using Hassert.*

Figure 1: *Comparison between verification flows*

Hassert is designed for engineers familiar with traditional verification methods like SW simulation and FPGA prototyping, enabling agile and efficient verification experiences. This paper contributes the following:

- Hassert provides both coarse-grained checking in system-level and fine-grained checking at the μArch level with assertion.
- Automatic switching for assertions and hardware snapshot schemes can further improve efficiency.
- An open-source, synthesizable hardware library for SystemVerilog Assertions to support different SVA hardware implementations.
- Demonstrated performance improvements on an Xilinx UltraScale+ FPGA with realistic workloads of up to 28941× over software simulation, with minimal area overhead and no impact on the DUT timing.

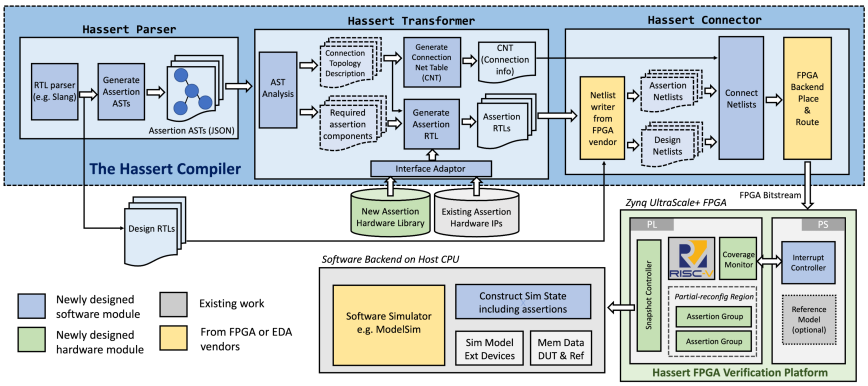


Figure 2: *Hassert Framework. Blue and green boxes are new modules and tools; yellow boxes are from vendors.*

The Hassert Framework

As a co-design framework for agile ABV, as shown in Figure 2, Hassert's key features include the following:

Multi-level self-checking. Hassert combines the self-checking mechanism in different design hierarchies, including the system-level and μArch-level. We expand ENCORE to perform coarse-grained differential checking of the key signals from both the DUT and the reference model. Through assertions, Hassert introduces additional observability by checking internal signals automatically.

Microarchitecture-guided hardware snapshots. Hassert enables debugging with a snapshot mechanism to transfer FPGA status to the simulator for error replay by utilizing the FPGA readback feature. Furthermore, Hassert can make a more precise snapshot that is closer to the root cause based on μArch status that is monitored by assertions, which are named assertion coverage. We define our coverage based on the assertion structure, checking if a sub-expression in assertion is triggered. When an assertion is fired or the coverage meets certain conditions, a snapshot will be created and debugging can be performed with the same fidelity as a software run to the same point.

Hassert compiler. To further facilitate ABV on FPGAs, Hassert compiles unsynthesizable assertion language such as SVA into synthesizable hardware with equivalent functionality. Our toolchain, Hassert compiler has three primary tools: 1) *hassert_parser*: Parses the original design to generate abstract syntax trees (ASTs) for all SVAs. 2) *hassert_transformer*: Using ASTs, this tool identifies atomic units and their connection topology in each assertion. Operators in the synthesizable SVA library then replace these units. 3)*hassert_connector*: Generates and connects netlists from DUT and SVA circuits to create the final design. This is sent to generate bitstreams. We also implement a library of the hardware SVA operators with what we believe to be the widest range of hardware-implemented SVA operators to date.

Dynamic switching of assertions. Vendor-provided debugging tools require re-compiling an entire FPGA design to test new signals; Hassert avoids this using the PR capabilities of modern FPGAs. The FPGA is divided into static regions for the DUT and dynamic regions for assertions, Thus Hassert can dynamically swap assertions without recompiling.

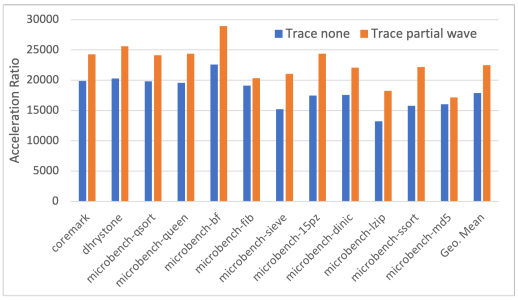


Figure 3: *Comparison against ModelSim simulations.*

Experimental Results

We implement and evaluate Hassert on the Fidus Sidewinder board, using two case studies from open-source projects: 1) a 64-bit RISC-V processor core, and 2) hardware accelerator designs from macsuite, which can be integrated with RISC-V processors.

Performance evaluation. For these practical accelerator designs in case study 1, the geo. mean of the acceleration ratio achieved by Hassert is 2932.9× at 100MHz and 12986× at the maximum frequency. For case study 2, we again implement the DUT on the FPGA at 100MHz and test over realistic workloads and benchmarks. The comparison results are demonstrated in Figure 3, where two different debugging options are enabled in ModelSim. Hassert achieves an acceleration ratio of 13242× $\sim$ 22589× and 17154× $\sim$ 28941× over ModelSim in these situations respectively, with average speedups of 17858× and 22508×.

Efficiency and Overhead Analysis. We evaluate the area overhead of Hassert regarding the resource usage on FPGA. We compare our result Xilinx ILA for the same set of signals. Hassert only added 0.55% of LUTs, 0.55% of Registers, and 1.22% of BRAM compared to the FPGA total resource, while ILA added notable BRAM usages to trace signals.