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

Ziqing Zhang[1,2], Weijie Weng[3*], Yaning Li[4*], Lijia Cai[5*], Haoyu Wang [6*], David Boland[7], Yungang Bao[1,2] Kan Shi[1,2]

[1] SKLP, Institute of Computing Technology, CAS [2] University of Chinese Academy of Sciences [3] Xiamen University of Technology [4] University College Dublin
[5] Hong Kong University of Science and Technology [6] Zhejiang University [7] The University of Sydney

✉: zhangziqing23z@ict.ac.cn , shikan@ict.ac.cn

Background

With the growing complexity of hardware designs, functional verification has become the bottleneck throughout the entire chip development cycle. Existing platforms face the dilemma of verification **efficiency** and **effectiveness**.

- **Software RTL Simulation**
  - Pros: high visibility with many practical verification tools and methods.
  - Cons: extremely slow speed, for large design < 10kHz.
- **FPGA Prototyping**
  - Pros: enable thorough verification with the fastest simulation speed.
  - Cons: debugging disaster, lack of debugging capability, and error-checking mechanism.

Functional Verification with Hassert

The **simulation-based functional verification** employs the design on a specific verification platform, and then simulates it for **error checking** and **fault localization.**

Hassert improves verification in both aspects:

- **Multilevel effective self-checking**
  - Coarse-grained: Check against a system-level reference model.
  - Fine-grained: Check with Assertions(SVAs).
- **Agile fault localization with the snapshot**
  - Hassert can automatically snapshot the The design states when necessary.
  - Identify the bugs by replaying the progress from the closest snapshot.

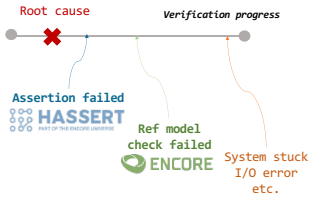

Figure 1: *Multilevel effective self-checking*

Hassert Overview

Hassert Compiler

The Hassert Compiler facilitates the automatic translation of **SVAs** to their corresponding digital circuits with the same functionality. It comprises three primary tools.

- **Hassert Parser:** Gather the AST from the input design files.
- **Hassert Transformer:** Map the SVA to RTL with the operator in the Hassert Operator Library.
- **Hassert Connector:** Connect all the assertions with design at the netlist level.

Hassert Operator Library

Hassert provides multiple operators' semantic equivalence implementation for functional verification.

- Sample Function, e.g., `$past`
- Sequence Operator, e.g., `and, or`
- Delay & Repetition, e.g., `[*n], ##`
- Property operator, e.g., `|=>,|->`

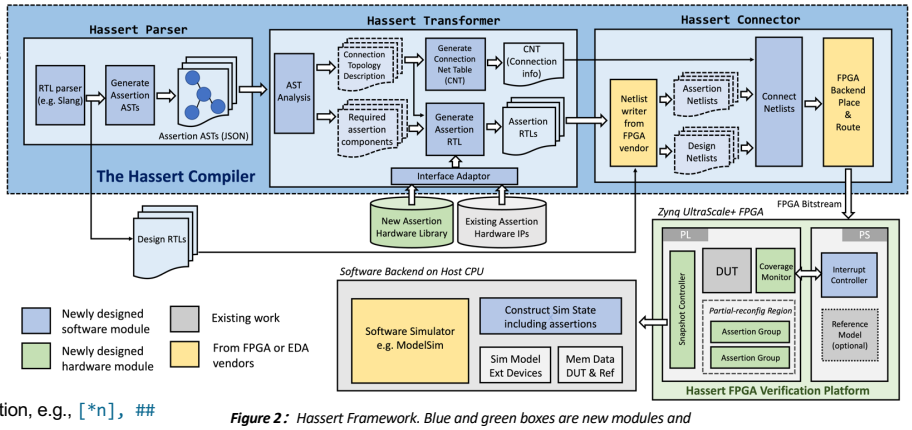

Figure 2: *Hassert Framework. Blue and green boxes are new modules and tools; yellow boxes are from FPGA/EDA vendors.*

Hassert for Fault Localization

Assertion Coverage

- Assertion in Hassert serves as a probe in ROI (region of interest), monitoring the ***μ***Arch state.
- Coverage in Hassert consists of a hit counter and a sub-expression toggle monitor.

Dynamic Switching of Assertion

- Hassert supports dynamic switching by utilizing partial reconfiguration.
- Hassert will schedule the assertions based on the Assertion Coverage and Aera overhead of assertions.

Microarchitecture-guided snapshot

- **Periodic Snapshot:** Perform the snapshot periodically, and locate the bug by replaying from the closest snapshot.
- ***μ*Arch Guided snapshot:** Based on the assertion coverage information, only trigger snapshot when necessary.

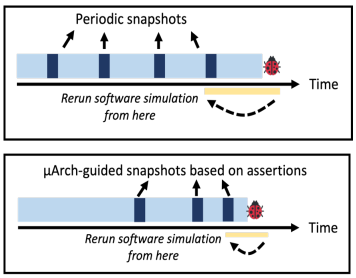

Figure 3: *Comparison between different snapshot strategies*

Debugging with Hassert

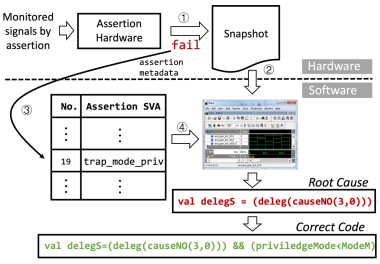

Figure 4: *The debugging workflow with Hassert*

① Assertion detects a violation and a fail flag issue, initiating a snapshot.

② The simulator replays the simulation with the snapshot

③ Assertion metadata is passed to the software driver to look up the exact assertion triggered.

④ Identify the bug according to the assertions.

Evaluation

Platform:

FPGA: Fidus Sidewinder board

- with ZYNQ UltraScale+ XCZU19EG FPGA Chip and two 16GB DDR memory

Server: Dual AMD Ryzen 5950x 16 cores Processor

Configuration:

- System Clock Frequency: 100MHz.
- ISA Emulator: NEMU
- DUT: Nutshell

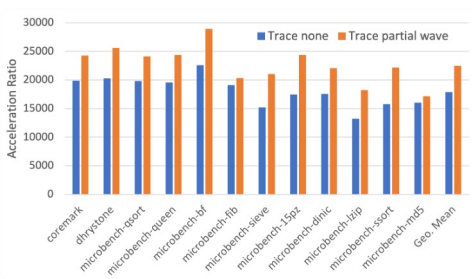

Figure 5: *Comparison against ModelSim simulations with different debugging options. (Trace none: disable the wave dump, Trace partial: only dump the signal monitored by assertion)*

Resource	Hassert	ENCORE	ILA (config1)	ILA (config2)
LUTs	2820 (0.55%)	1597 (0.31%)	2266 (0.43%)	3749 (0.72%)
Block RAMs	12 (1.22%)	12 (1.22%)	8 (0.81%)	512 (52.03%)
Registers	2426 (0.23%)	1493 (0.14%)	3833 (0.37%)	4243 (0.41%)

Table 1: *Resource Usages Comparison*

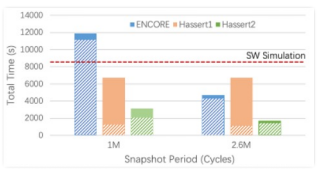

Figure 6: *Different debugging approaches reproducing a known bug. (Shaded parts: time for snapshot; solid parts: SW simulation time.)*

*: Weijie Weng, Yaning Li, Lijia Cai and Haoyu Wang finish this work during internship at the SKLP, Institute of Computing Technology, CAS