Reproducing a Deep Corner-Case Bug using Formal

Abhishek Anand
SiFive

Bug Overview

Bug Overview

- What was the bug ?
  - A lost RAW hazard in OOO LSU was causing older data to be returned

- Why was this a corner-case bug ?
  - Required multiple hazard dependencies to be created between Multiple Instructions and Cached Data
  - Cache-line needed to be evicted within a specific window of multiple dependencies being resolved
  - Finally, there was a single cycle window where store resolve needed to happen for this bug to occur

LSU Formal TB Overview

LSU Formal Test-bench Overview

- Why is LSU a complex block for Formal ?
  - LSU is an out-of-order processor unit which tracks hazards and dependencies for all ongoing instructions
  - A given instruction in LSU can go through multiple paths depending on cache states and other instructions in the pipeline, leading to huge state space
  - High end-to-end Latencies

- How did we tackle complexity ?
  - Parameter Reduction
    - Store/Load trackers reduced significantly to the order to ~4
  - Memory Abstraction
    - Store data only for tracked address (under-constrained for everything else)
  - Reset Abstraction
    - Allow Store/Load trackers and Caches to be filled up out-of-reset

LSU Formal Test-bench Overview

- Results from LSU Formal Test-bench
  - 60+ bugs reported through Formal Verification
    - ~25% of reported bugs were found to be high-impact and corner case scenarios
- Why did LSU Formal miss the bug ?
  - Parameter Reduction
    - Given bug was unique to larger parameter design causing extensive hold at a given Arbiter
  - Huge Latency
    - Proof depth not being sufficient was a known limitation at the time of this bug
    - End-to-end assertion was hitting depth = 14; bug required depth = ~25 (even with reset abstraction)
    - This was even after running with ~96 workers for 60+ hours

Formal Effort to Reproduce Bug

Formal Effort for Bug

- Why was it important to reproduce the bug in Formal ?
  - The bug needed to be reproduced either in Formal or Simulation to confirm designer's hypothesis
  - Fix Verification and further bug-hunting was required to ensure absence of similar bug

- Further steps taken
  - Under-constrained Arbiter to allow extensive holds possible only with larger configs
  - Used Local assertion to reduce scope of assertion
  - Replaced generic reset abstraction with FSDB loading at reset

Under-constraint

- Why was it required ?
  - For larger design there were ~32 requestors at an internal arbitration point
  - For parameter reduced design #requestor dropped to ~4
  - Bug required the Arbiter to hold request for a given requestor for 8+ cycles

- How was it achieved ?
  - Added snip on grant and constrained it as

```
~(|req) |-> ~gnt
```

  - Modified behaviour
    - Grant can remain de-asserted even if there is a pending request
    - Discounted this behavior for all dependent Forward Progress Assertions

Local Assertion

- Why was it required ?
  - Original end-to-end load correctness assertion had huge COI
    - 1,000+ flops in COI (even after all abstractions)
- What was the new assertion ?
  - New assertion focused on the hazard dependencies being correctly maintained until a given instruction completes
    - Only 100+ flops in COI
- Should this be the recommended approach ?
  - Ideally we would want our end-to-end assertion to be robust enough to cover everything
  - However, since this bug required quick resolution, moving to local assertion helped
  - Later the Formal Setup was made more robust using divide and conquer approach

Reset Abstraction

- Why was it required ?
  - Reset Abstraction was very generic allowing
    - All store/load trackers to hold instructions out-of-reset
    - Complete L1 Cache to be occupied with data present out-of-reset in any coherence state
  - This caused the assertion to observe huge state-space out-of-reset and state-space explosion to slow down the assertion
- What was the alternative method ?
  - Wrote covers to generate waveforms for pre-condition of the buggy scenario to occur
  - Loaded the generated waveforms at reset and ran targeted assertion

Results

- Able to hit the bug in ~1 minute as compared to not being able to hit it in 60+ hours !!
- Fix was verified using same approach
- Further bug-hunting was done using same approach to further improve the confidence

Further Takeaways

- Employ under-constrained behaviour on Arbiters across all designs
  - This was adapted and added to Simulation flow as well
- Add more Local Assertions in designs to help hit bugs faster
- Employ Bug-Hunting with FSDB loaded waves for Formal setups with known proof-depth issues


THANK YOU
Our Technology, Your Innovation™