Silicon Bug Hunt with "Deep Sea Fishing" Formal Verification

Ping Yeung, Jin Hou, Siemens EDA
ping_yeung@mentor.com, jin_hou@mentor.com

Success Factors of Formal Verification

To success with formal verification, users need to wrestle with multiple success factors:

- The complexity of the design [3][8]
- The quality of the sub-goals and target assertions [6]
- The completeness of the interface constraints [5]
- The control and orchestration of the formal engines [8]
- The quality of the initial states for formal exploration [7]
- The formal expertise of the users [2]

The radar chart guides the deployment of formal verification to find deep silicon bugs

Phase 1: "Initial"

- gather all the information, and set up the formal verification environment.

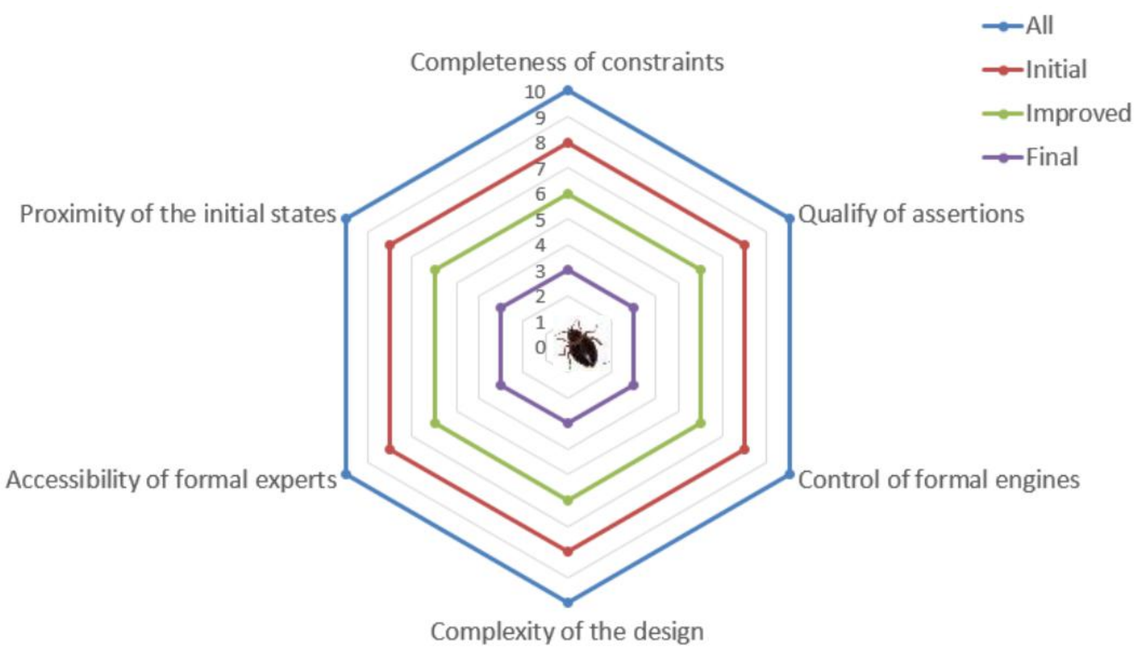
Phase 2: "Improved"

- improve each of the success factors to define the scenarios close to the bug.

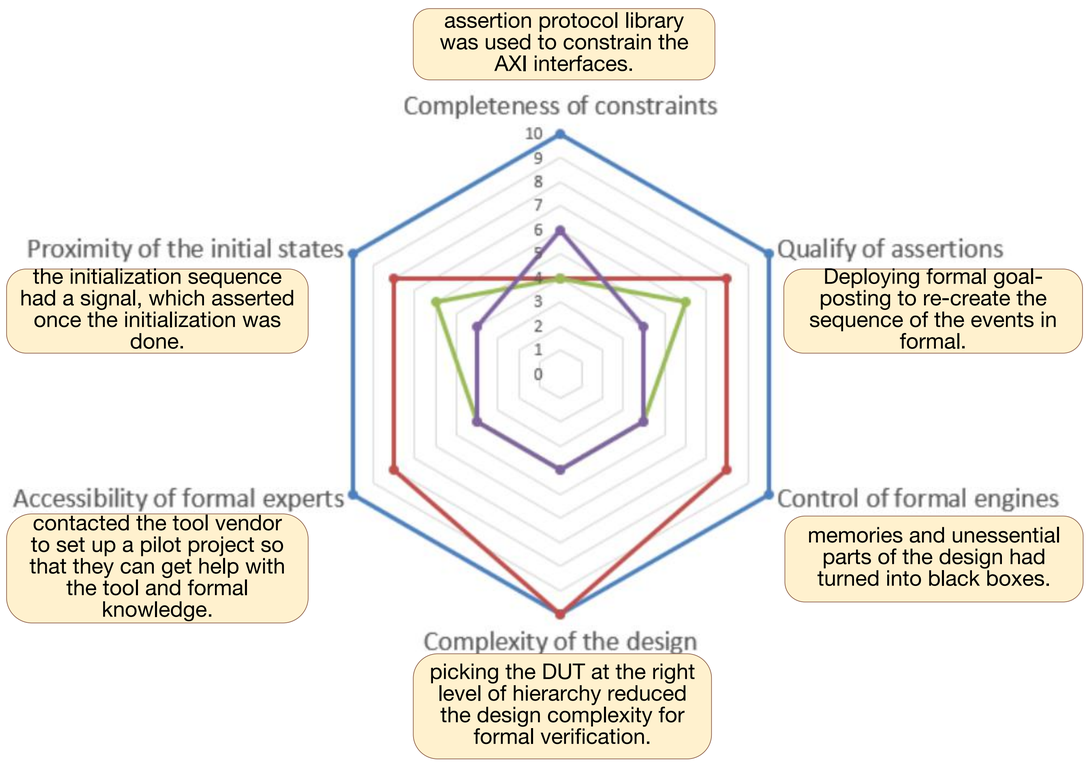
Phase 3: "Final"

- optimize the critical success factors to find the bug(s).

Formal verification radar chart



Post-Silicon Bug: DDR3 controller

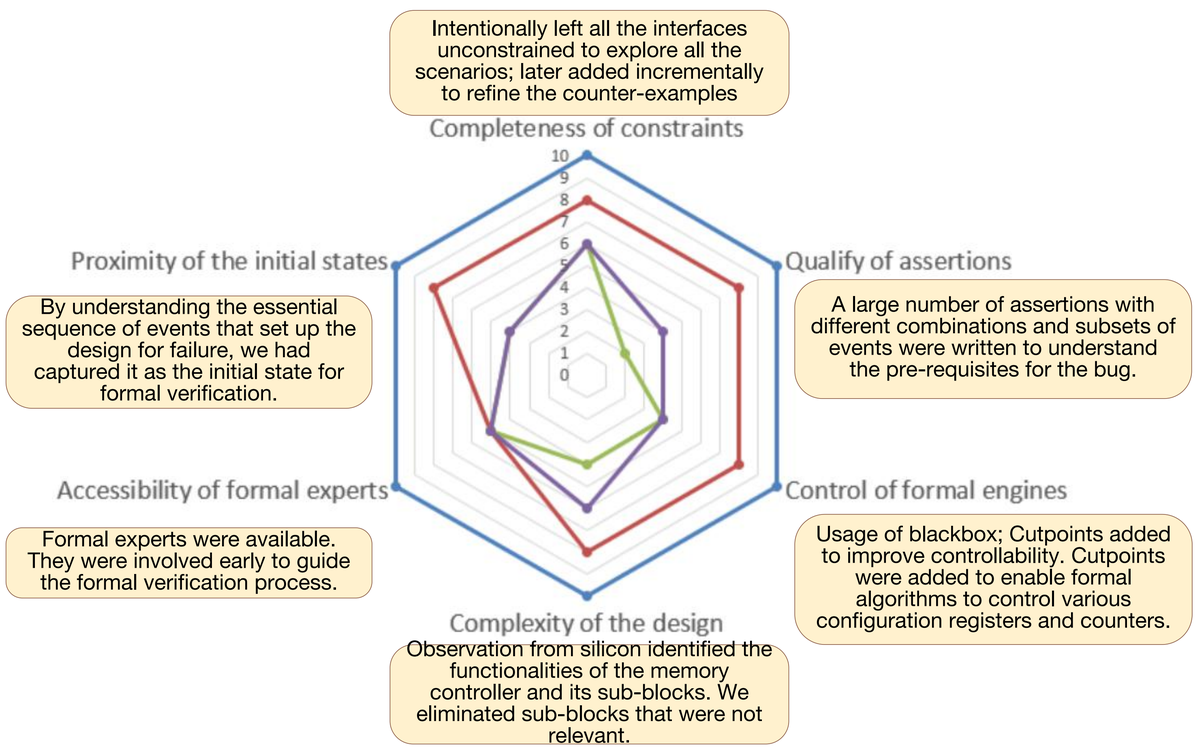


Post-Silicon Bug: DDR3 controller

Scenario: a sequence of write commands to the specific memory bank and row combinations would cause a DDR3 protocol violation related to pre-charge timing.

- Complexity of the design: picking the DUT at the right level of hierarchy reduced the design complexity for formal verification. Standard interfaces at the DUT also help constrain the design.
- Accessibility of formal experts: the project team did not have formal expertise. They contacted the tool vendor to set up a pilot project so that they can get help with the tool and formal knowledge.
- Control of formal engines: memories and unessential parts of the design had turned into black boxes. Preliminary formal runs were done to confirm that the formal engines have adequate control of the design.
- Qualify of the assertions: Assertions were written to capture the bug scenario and the sequence of events leading up to the bug. This is important. By using the sequence of assertions as sub-goals, we were able to deploy formal goal-posting to re-create the sequence of the events in formal. It helped guide the formal engines to target the bug scenario(s).
- Proximity of the initial states: it is essential to configure the design for proper operation. The serial nature of the design made it challenging to apply formal verification directly. We were fortunate in that the initialization sequence employed in the design had an "init_ok" signal, which asserted once the initialization of the design was complete. Also, by using the initial states from the sub-goals, it significantly improved the proximity of the initial states that lead to the bug scenario.
- Completeness of the constraints: assertion protocol library was used to constrain the AXI interfaces. Although the DUT has 5 AXI interfaces, initially, we disabled 4 of them to reduce complexity and to focus all transactions on one interface. Later, we enabled more interfaces to study the interactions between the different interfaces.

Post-Silicon Bug: Memory Controller



Post-Silicon Bug: Memory Controller

Scenario: When the read/write transactions were re-ordered by control logic inside the memory controller, old data was read before the location had been updated.

- Complexity of the design: The failing scenario was first observed in the lab during post-silicon testing. Based on the bug triage in the lab, the memory controller was identified as the source of the problem. We continued to eliminate sub-blocks that were not relevant.
- Accessibility of formal experts: it was fortunate to have formal experts in the team. They were involved early to guide the formal verification process.
- Control of formal engines: Memories and unessential parts of the design had turned into black boxes. Cutpoints were added to enable formal algorithms to control various configuration registers and counters. We had added cutpoints to some state machines as well.
- Quality of the assertions: This was the most important factor of this bug hunt. As simulation was able to hit only a small subset of events that lead to the bug, a large number of assertions with different combinations and subsets of events were written to understand the pre-requisites for the bug. Formal verification was able to hit 85 percent of the pre-requisites conditions. After some refinements, it was able to find the bug with a few well-timed transactions and external triggers.
- Proximity of the initial states: Although the MTBF in silicon was at least 2 hours, we were able to understand the essential sequence of events that set up the design for failure. We had captured it as the initial state for formal verification.
- Completeness of the constraints: Initially, we intentionally left all the interfaces unconstrained to explore all the scenarios and ensure formal verification could find the combinations of events that lead to the bug. After formal verification had found the silicon bug, interface constraints, setup, and configuration constraints were added into the formal runs to refine the counter-example.

[1] Ram Narayan, "The future of formal model checking is NOW!", DVCon 2014.
[2] M Achutha KiranKumar, et al., "Making Formal Property Verification Mainstream: An Intel® Experience," DVCon India 2017
[3] Mandar Munishwar, Vigyan Singhal, et al., "Architectural Formal Verification of System-Level Deadlocks", DVCon 2018.
[4] Richard Ho, et al., "Post-Silicon Debug Using Formal Verification Waypoints," DVCon 2009
[5] Blaine Hsieh, et al., "Every Cloud - Post-Silicon Bug Spurs Formal Verification Adoption," DVCon 2015
[6] Jin Hou, et al., "Handling Inconclusive Assertions in Formal Verification", DVCon China 2018
[7] Mark Eslinger, Ping Yeung., "Formal Bug Hunting with "River Fishing" Techniques", DVCon 2019
[8] Jeremy Levitt, et al., "It's Been 24 Hours - Should I Kill My Formal Run?", Workshop, DVCon 2019

Conclusions

Finding silicon bugs with Formal Verification is a potential challenging process
The "Deep Sea Fishing" radar identifies the success factors and guides users of the process
Successive refinement of the critical factors to "zero-in" on the critical bugs.
The process is not just technical. It includes human factors and organization considerations as well.