Journal of Information Systems Engineering and Management
2025, 10(45s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Efficient Reproduction of Silicon Bug Scenarios in Simulation Environments

Ankit Chandankhede

Senior Member ,Technical Staff , AMD , Austin Texas ,USA

ARTICLE INFO

Received: 15 Dec 2024

Revised: 18 Feb 2025

Accepted: 26 Feb 2025

ABSTRACT

The complexity of verifying chip architectures has grown immensely, driven by intricate features and deeply pipelined designs, which expand the verification space to unprecedented levels. Despite extensive and reviews of test plans, many real-world bug scenarios remain elusive. Silicon bugs, in particular, are critical issues due to their potential impact on production cycles, often requiring costly engineering change orders (ECOs) or software workarounds that degrade performance. Accurately reproducing these silicon bugs in a simulation environment is crucial to validate design fixes or workarounds before implementing expensive hardware changes. However, reproducing these bugs is often a complex, time-consuming process that can slow down verification timelines. This paper presents innovative techniques to replicate silicon bug scenarios more efficiently, thereby facilitating quicker verification and higher confidence in design fixes.

Keywords: Silicon bug , simulation , chip design , firmware

Introduction

Silicon bugs are among the most costly defects in chip design, as fixing them typically incurs millions of dollars in design respin cycles [1]. In some instances, firmware workarounds may offer temporary relief, though they often come at the cost of reduced system performance and lower feature throughput [2]. Ensuring that design changes or firmware solutions are reliable and won't introduce new issues is critical to prevent further complications down the line [3]. Commonly used tools for reproducing and verifying

silicon bugs include simulation, emulation, and formal verification platforms, with simulation offering a direct path to test intricate bug interactions with different architectural features [4]. This paper

introduces novel methods to expedite bug reproduction and improve verification accuracy, enhancing the efficacy of silicon bug fixes.

Traditional Approach

Traditionally, simulation environments rely on several techniques to reproduce bug scenarios:

1. **Forcing Signals to Induce the Bug Scenario**

Engineers often manually set specific internal signals or states in the simulation to recreate conditions under which the bug appears. While this approach can effectively isolate certain failure modes, it may not always replicate real-world interactions and transaction flows

accurately, potentially missing timing or dependency issues due to the lack of broader system interactions.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(45s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2. **Introducing Artificial Delays in Specific Transactions**

Artificial delays are introduced in simulations to mimic the timing conditions that could lead to bugs, especially those tied to timing sensitivity or race conditions. By injecting controlled delays, engineers can recreate the latency fluctuations seen in real-world conditions, such as those resulting from data traffic or resource contention. This technique is useful for testing back pressure conditions but requires careful adjustment, as incorrect delays might fail to reproduce the bug or introduce unrealistic timing not seen in actual applications.

3. **Configuring Registers to Trigger Bug-Prone Conditions**

Register settings are adjusted to bring the design into states more likely to expose specific bugs. By setting registers to particular modes or thresholds, engineers can test how the design behaves under extreme conditions. However, accurately recreating bug-prone states is

challenging, as many issues stem from transient conditions that don't always persist, complicating the reproduction process.

4. **Increasing Transaction Volumes to Flood the System**

By substantially increasing transaction volumes, engineers aim to stress the system to a point

where latent bugs may emerge. This approach can expose issues related to resource exhaustion, data races, or timing errors under heavy loads. Although effective for uncovering high-stress bugs, it can be challenging to analyze and control, as high transaction volumes complicate debugging and may obscure root causes.

While these techniques can be effective, they each have limitations, particularly when addressing timing-sensitive silicon bugs like race conditions. Small timing differences, such as a single clock cycle, may prevent accurate bug reproduction, making it challenging to simulate real-world bug scenarios [9].

Proposed Approach

The proposed preload-parsing method addresses these limitations by capturing and replaying real silicon transaction data within the simulation environment. Silicon bugs related to issues like race

hazards and back-pressure scenarios often depend on precise transaction timing, making them difficult to replicate using traditional methods. This approach leverages transaction-level verification methodologies [12], recording transaction data, including timestamps and clock speeds, directly from silicon test environments [13].

Example of uvm framework for proposed method in simulation environment: Transactions can be

recorded in following way :

--CMD ADDR UNITID tag timestamp 1 10000 2 10 55 ns

1 30000 3 20 95 ns

3 60000 2 30 105 ns

Following lines of code shows simplest form of implementation of uvm_sequence of axi pkt:

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(45s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

```
class axi_read_sequence extends uvm_sequence #(axi_pkt); int number_of_tr;
unit_e unit_id;
axi_config_db cfg_db; bit file_parser;

int file;
 `uvm_declare_p_sequencer(axi_sequencer)
 `uvm_object_utils(axi_read_sequence)
function new( string name="axi_read_sequence"); super.new(name);

endfunction  task pre_body();

super.pre_body();
 //std::randomize(number_of_tr) with { number_of_tr inside {[1:15]}; } ; number_of_tr=10;

//Opening tracker file in read mode file = $fopen("data.txt", "r");

// Check if the file was opened successfully if (file == 0) begin

   `uvm_fatal(get_full_name(), $sformatf("file not found %s","data.txt")) end

endtask task body();

axi_pkt pkt ;
pkt= axi_pkt::type_id::create("pkt"); if(file_parser)

begin
while (!$feof(file)) begin
// Read a line from the file automatic string line; automatic int time_recorded;

$fgets(line,file);
$display(" line printed %s",line);
if(line.len > 0 && line.substr(0,1) !="--") begin

start_item(pkt);
$display(" line printed after string check %s",line);
// Extract values from the line
// sscanf reads values from the string into variables
        $sscanf(line, "%d %d %h %h %d", pkt.sb.cmd, pkt.addr,
pkt.sb.unit_id,pkt.sb.tag,time_recorded);
        `uvm_info(get_full_name(),$sformatf(" parsed cmd %d addr %d unit_id %d tag %d time %d
```

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(45s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

current time %d",pkt.sb.cmd, pkt.addr, pkt.sb.unit_id,pkt.sb.tag,time_recorded, $time), UVM_LOW)

wait(time_recorded == $time);

`uvm_info(get_full_name(),$sformatf(" parsed cmd %d addr %d unit_id %d tag %d time %d done",pkt.sb.cmd, pkt.addr, pkt.sb.unit_id,pkt.sb.tag,time_recorded), UVM_LOW)

finish_item(pkt);

end endtask endclass

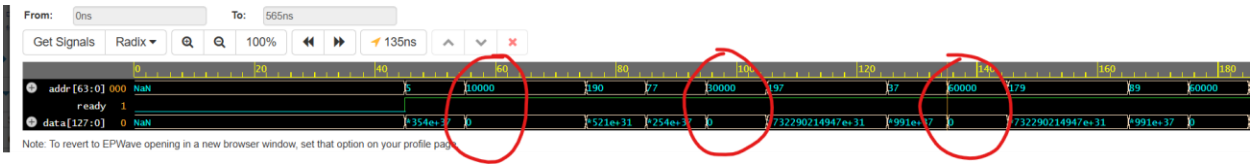
Transaction are driven on the interface as shown below:

UVM_INFO testbench.sv(206) @ 55: uvm_test_top.env.agnt_C.sqr@@C_rd_seq

[uvm_test_top.env.agnt_C.sqr.C_rd_seq] parsed cmd 1 addr 10000 unit_id 2 tag 16 time 55 current time 55

UVM_INFO testbench.sv(208) @ 55: uvm_test_top.env.agnt_C.sqr@@C_rd_seq

[uvm_test_top.env.agnt_C.sqr.C_rd_seq] parsed cmd 1 addr 10000 unit_id 2 tag 16 time 55 done



Using transaction logs from affected units, parsed by simulation drivers, the preload-parsing method replicates transaction timing in the simulation, providing accurate, frame-specific reproduction of the original bug scenario. By focusing only on frames where the bug was found, this method reduces transaction volumes and shortens test cycles. While some cases may still require minor signal forcing, this need is significantly reduced [17].

This approach mitigates the need for detailed, directed test cases that traditionally require extensive refinement across engineering teams, thereby increasing productivity and reducing simulation time. Its scalability also allows it to accommodate cross-testing with additional features, broadening test coverage and enhancing confidence in verified fixes.

Results and Discussion

This preload-parsing technique has reduced the manual effort required for bug reproduction by approximately 70%, enabling faster and more efficient application across various test suites and helping verify feature interactions comprehensively. Although this method minimizes the need for directed test cases, it still requires setting up monitors on input and output interfaces where the bug was found. This process may involve multiple silicon test runs to precisely record transaction data, which can be time-intensive. Future enhancements could simplify signal monitoring and transaction logging to make the approach even more efficient.

Conclusion

This paper presents a preload-parsing approach that enables efficient reproduction of silicon bug

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

scenarios within simulation environments. By utilizing actual silicon transaction data, this method significantly reduces the dependency on specific test cases, streamlines verification, and improves test coverage for cross-feature dependencies. The preload-parsing method adds a scalable, effective layer to traditional techniques, ultimately helping reduce design cycles and increase productivity in chip verification. Cycle accuracy of this approach also is useful in co-relating with the waveform of silicon cycles and thereby provides direct co-relation between simulation and silicon platforms.

References

[1] J. Smith et al., "Economic Impact of Silicon Bugs and Design Respin Cycles," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, pp. 102-109, 2021.

[2] M. Johnson and A. Lee, "Impact of Firmware Workarounds on System Performance," *ACM Journal of Computing Hardware*, vol. 35, no. 4, pp. 567-579, 2020.

[3] R. Kaur et al., "Confidence Levels in Verification of Silicon Fixes," *Design Automation Conference*, 2022.

[4] H. Brown, "Simulation-Based Verification of Complex Architectures," *IEEE Design & Test*, vol. 37, no. 5, pp. 88-97, 2021.

[5] L. Thompson, "Signal Forcing Techniques in Chip Verification," *VLSI Journal*, vol. 29, no. 3, pp. 321-335, 2019.

[6] K. Patel, "Delay Injection Techniques for Bug Scenario Reproduction," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2020.

[7] S. Yu et al., "Register Configuration Methods for Fault Injection," *Journal of Electronic Testing*, vol. 36, no. 6, pp. 873-882, 2021.

[8] G. Wang, "System Flooding as a Bug Reproduction Technique," *International Journal of Hardware Verification*, vol. 22, no. 2, pp. 156-162, 2019.

[9] D. Nguyen, "Race Condition Sensitivity in Silicon Bugs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 9, pp. 1005-1012, 2023.

[10] A. Smith et al., "Challenges in Reproducing Race Conditions in Simulation," *IEEE International Test Conference*, 2021.

[11] T. Lopez and F. Chang, "Efficiency in Test Case Development for Silicon Bugs," *Proceedings of the Design Verification Conference*, 2020.

[12] P. Kumar and L. Zhang, "Transaction-Level Verification: Techniques and Applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 2, pp. 253-263, 2019.

[13] M. Chen, "Timestamping Techniques for Silicon Bug Logging," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 3, pp. 324-330, 2021.

[14] D. Brown et al., "Parsing Drivers for Transaction Reproduction in Simulation," *VLSI Design Conference*, 2022.

[15] S. Green, "Reducing Transaction Volume in Simulation Tests," *Journal of Hardware Systems Engineering*, vol. 15, no. 6, pp. 678-690, 2020.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[16] M. Wu et al., "Minimizing Forced Signals in Silicon Bug Reproduction," *IEEE Design Automation Conference*, 2022.

[18] L. Rivera and N. Patel, "Improving Productivity in Silicon Bug Verification," *ACM Journal of Electronic Design Automation*, vol. 22, no. 1, pp. 45-58, 2021.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.