2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

When Last Minute FV Strikes Gold: A Case Study on Finding Starvations & Deadlocks in a Project Nearing Tape-in

Abhishek Potdar, Sachin Kumawat, Sudhanshu Srivastava, Anshul Jain
Intel Corporation
{abhishek.potdar, sachin.kumawat, sudhanshu.srivastava, anshul.jain}@intel.com

***Abstract*- As the complexity of system-on-chip (SoC) designs increases, the cost and criticality of bugs escalate significantly. Late-stage bug discovery or, worse, post-silicon detection leads to panic and paranoia, making confidence in design quality imperative. While formal verification (FV) is ideal for comprehensive bug detection, skepticism surrounds its application on design IPs from scratch due to perceived time and resource constraints. In this paper, we present a case study demonstrating the successful utilization of last-minute formal verification techniques near tape-in date. Through clever bug hunting strategies involving existing RTL assertions and high-level end-to-end checkers, we identified and resolved showstopper starvation and hang issues that could have caused significant delays if undetected. Our experience highlights the importance of formal verification and provides valuable insights for engineers and managers involved in critical projects, showcasing the benefits of employing formal bug-hunting techniques even in the late stages of a project.**

I. Introduction

As the development of modern-day System on Chips (SOCs) advances through various design and verification phases, the importance and cost of detecting and addressing bugs become increasingly critical. With each stage of the design cycle, the difficulty and expense of fixing these bugs grow exponentially, making it crucial to identify and resolve them as early as possible. If a bug is discovered late in the development process or, in the worst case, after the silicon has been produced, it can cause panic and lead to significant setbacks. In these situations, it is essential to have the utmost confidence in the quality and robustness of the design. Since formal technology is ideal for finding needles in the haystack due to its breadth-first search, it provides highest level of assurance about robustness of the fix.

There has been much skepticism about applying formal verification (FV) on designs IPs from scratch. It is considered as a time-consuming, effort-intensive and expertise dependent task. While these concerns are not entirely baseless, it is essential to recognize the significant benefits that FV brings to the table, outweighing its limitations. Formal verification has indeed revolutionized the verification landscape by providing rigorous, mathematical methods to ensure design correctness. So, however challenging, inclusion of formal verification can strike gold even when the runway is short, and it is crucial to de-risk the time-to-market (TTM) of flagship SOCs.

In this paper, we share our innovative strategy that has the potential to greatly enhance design quality, even when formal verification is introduced close to the tape-in date. We discuss our ingenious approach to bug hunting techniques, which involves utilizing existing RTL assertions and high-level end-to-end checkers in a project nearing tape-in, without requiring extensive design ramp-up time. Our experience report highlights the success of our last-minute formal verification process, which uncovered critical starvation and hang issues that could have caused significant delays if left undiscovered. We delve into the techniques used to pinpoint and resolve these issues, offering valuable insights into the design and verification process.

II. Related Work

A. RTL Embedded Assertions

The concept of assertions has been a part of computer science for many years; however, it only gained prominence in hardware design after the introduction of hardware description languages like VHDL and Verilog in the 1990s. In 2002, the Accellera Systems Initiative introduced SystemVerilog Assertions, which provided a powerful and flexible mechanism for specifying design properties. This innovation was widely adopted and integrated with modern verification tools and methodologies, significantly impacting the field of hardware design.

Despite these advancements, assertions are still primarily used in simulation, which is a valuable application but not the most optimal one. In this paper, we present an innovative approach where we effectively utilized well-written, abundant RTL embedded assertions with formal technology to verify critical aspects of design correctness in a more efficient manner.

By leveraging formal technology in conjunction with RTL embedded assertions, we were able to enhance the verification process, ensuring a higher degree of design correctness and reliability. This approach showcases the potential of combining traditional assertion-based verification methods with advanced formal techniques to achieve a more robust and effective verification process for hardware designs. As a result, designers and engineers can benefit from this powerful synergy, ultimately contributing to the development of more reliable and robust hardware systems.

B. Formal Coverage

Coverage is a handy utility to measure how much of the design's functionality has been exhaustively verified using formal methods. As we near tape-in, everyone in the design and verification team is laser-focused on the coverage number to feel confident about the design quality. In such situations, confirmations from formal coverage can help build confidence on design quality. In this paper, we used two types of formal coverage to augment overall verification coverage.

1. Reachability Coverage is a valuable metric that aids in determining the extent to which various portions of a design can be reached during the verification process. By examining reachability coverage, we gain insights into the effectiveness of their verification efforts and identify any gaps that may need further attention.
2. Observability Coverage is an essential metric in verification process that assists in evaluating the completeness and effectiveness of properties used to verify a design. Observability coverage helps identify portions of the design that are not checked by any property. By analyzing observability coverage, we gain insights into the areas of the design that may require additional properties or adjustments to existing properties to improve their effectiveness.

C. Bug-hunting

Bug hunting is a well-established area in formal verification, with numerous techniques being developed and integrated into industrial formal verification tools. Formal bug hunting is particularly effective in identifying subtle or complex bugs that may be overlooked by other verification methods, ultimately enhancing the robustness and reliability of a design.

However, employing naïve bug hunting techniques can be a time-consuming and computationally intensive process, potentially delaying the overall verification timeline and requiring significant resources. In this paper, we will share some innovative and clever bug hunting techniques that have enabled us to find showstopper bugs in a limited amount of time and with limited compute resources.

These sophisticated techniques leverage the power of formal verification tools while optimizing the bug hunting process, making it more efficient and resource friendly. By adopting these strategies, designers can quickly and effectively uncover critical bugs that may have otherwise remained undetected, significantly improving the overall design quality and reducing the risk of costly issues arising later in the development cycle.

These clever bug hunting techniques not only demonstrate the potential of formal verification in identifying and addressing complex design flaws but also showcase how innovative approaches can overcome the limitations of traditional bug hunting methods. This paper serves as a valuable resource for designers seeking to enhance their verification processes and efficiently discover showstopper bugs, ultimately contributing to the development of more reliable and robust hardware designs.

III. Verification Challenge – Detect Deadlocks & Starvations

Formal verification was challenged with a unique task of addressing concerns related to the possibility of deadlocks and starvation issues in a complex bridge design between fabric and memory controller of a server CPU System-on-Chip (SOC). Deadlocks and starvation issues can have severe consequences on the overall performance and functionality of a system, making their detection and resolution critical in the design process.

These issues frequently escaped traditional verification environments in previous generations of the design, as they were difficult to detect and debug, particularly in large and complex designs such as the one shown in Figure 1. Traditional verification methods, such as simulation and testing, may struggle to uncover such issues due to the sheer scale and intricacy of the design, as well as the potential for rare or corner-case scenarios to trigger these problems.

Formal verification, on the other hand, offers a powerful alternative for detecting and addressing deadlocks and starvation issues in complex designs. By exhaustively analyzing the design's state space and rigorously checking its properties, formal verification can uncover subtle issues and scenarios that might be missed by other methods.

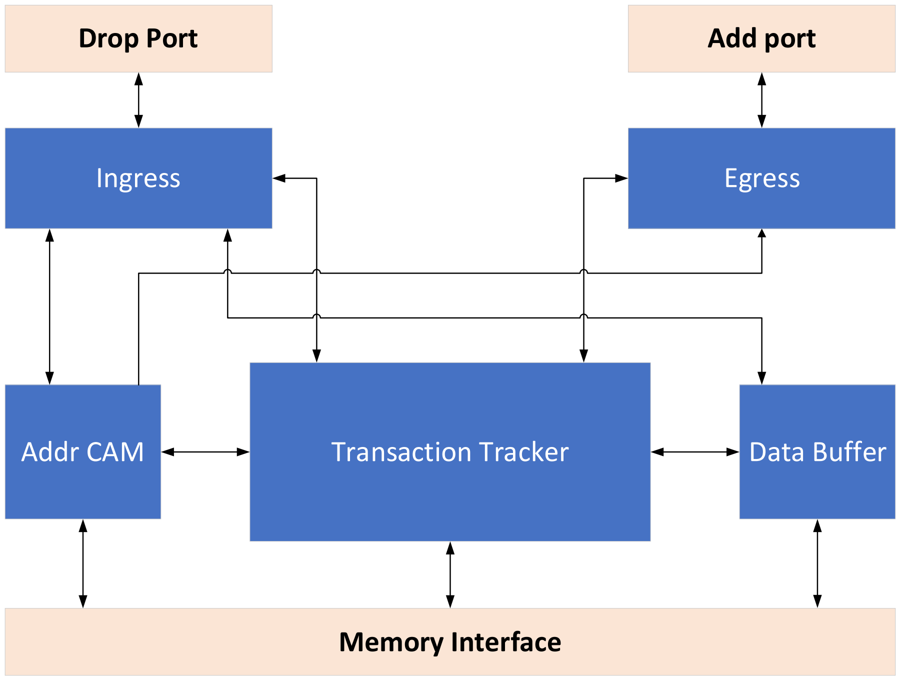

Figure 1: Design Block Diagram

Since the project was nearing tape-in, formal verification had a very short runway to execute. Without a proper understanding of the specifications and sufficient design ramp-up time, aiming for a formal signoff would not be feasible. Formal signoff typically requires a thorough understanding of the project requirements, detailed design documentation, and a comprehensive review process. It also involves investing significant time and effort to ensure that all aspects of the project meet the necessary standards and criteria.

IV. VERIFICATION SOLUTION – ABC (AMPLIFY BUILD CONCENTRATE) FORMAL

In the case of the complex bridge design between the fabric and memory controller of the server CPU SOC, formal verification was employed to systematically analyze the design and identify potential deadlocks and starvation issues. We adopted the approach called ABC formal where “A” stands for “Amplify”, “B” stands for “Build” and “C” stands for “Concentrate”.

A. *Amplify the impact of available RTL embedded assertions*

Formal verification planning typically starts with standard practices like browsing the design and specification files, understanding the hierarchy of modules, understanding the flow of transactions through the design, identification of micro-architectural details etc. During all this process, we identified a unique "goodness" in the design under test. The DUT was blessed with many detailed RTL embedded assertions which were curated over the course of RTL development and were run in dynamic simulation for flagging unexpected behavior of design's microarchitectural components such as counters, state machines, arbiters etc.

Since these RTL embedded assertions were run in dynamic simulation alone, there ability of finding bugs was limited by the test vectors run in dynamic simulations. In order to unlock the full potential of RTL embedded assertion and amplify their impact in terms of finding bugs, we setup the DUT in formal property verification tool, and ran the same RTL embedded assertion with exhaustive input vectors. This approach helped increase the effectiveness of the RTL embedded assertions many folds.

DUT had many assertions, and we needed a systematic approach to manage the clean-up process effectively. To initiate this effort, a triage of the RTL assertions was conducted, considering their individual characteristics. The triage aimed to identify and categorize the assertions based on their importance, complexity, feature and potential impact on the design. This evaluation revealed that out of approximately 300 assertions in the design, around 40 distinct failures were present.

Formal verification tools are known for bombarding the DUTs with all possible input scenarios, which include both valid and invalid scenarios. Therefore, many of the initial assertion failures are false failures due to invalid input scenarios. We decided to tackle such failures in a phased manner. This strategy involved prioritizing the clean-up based on the features associated with each assertion. By organizing the assertions according to their priority, the verification team could focus on addressing the most critical issues first, while still ensuring that all assertions were eventually verified and cleaned up. Clean-up exercise involved debugging, analyzing, and resolving bogus failures by adding appropriate assumptions on input ports of the DUT.

To rule out any mistake in formulation and implementation of assumptions, source code of assumptions was integrated in dynamic simulation environment and validated in simulation regression runs. Through this exercise, we were able to identify unintentional over-constraints in FV environment which could affect the effectiveness of formal runs adversely.

B. *Build high-level forward progress checkers*

Forward progress checkers are an essential component in the verification process, particularly when addressing deadlock and starvation issues in a design. These checkers help ensure that incoming requests within the design can progress through the system without encountering deadlocks (where two or more processes are waiting for each other to release a resource) or resource starvation (where a process is unable to acquire the necessary resources to execute due to competition with other processes).

"Transaction Tracker" design mentioned in block diagram, involved multiple Finite State Machines (FSMs) interacting with each other in parallel based on input information. In such complex systems, the potential for deadlocks and starvation issues increases due to the numerous interactions and dependencies between the FSMs. As a result, it becomes imperative to conduct thorough checks using forward progress checkers to identify and prevent any occurrences of deadlock or resource starvation.

A sample implementation of forward progress check is shown below. It operates by monitoring the ingress (start) and egress (stop) of the transaction at DUT boundary and fail when transactions indefinitely stuck in the design.

```
module fwd_prog_check #(parameter MAX_COUNT = 8)
(
   input clk, rst,
   input start, stall, finish
);
reg [$clog2(MAX_COUNT):0] count;
always @(posedge clk) begin
   if (rst) count <= 'b0;
   else if (finish) count <= 'b0;
   else if (stall && (|count)) count <= count;
   else if (|count) count <= count + 1'b1;
   else if (start && !(|count)) count <= 'b1;
end
// Counter should not cross max allowed latency
count_must_not_cross_max: assert property (
   @(posedge clk) disable iff (rst)
   count <= MAX_COUNT
);
```

C. Concentrate on bug-prone microarchitectures for bug-hunting

The process of bug-hunting primarily relies on simulation-based proof engines, which explore proof cycles to randomly encounter bug scenarios without examining the entire state space in each cycle. Conversely, formal proof engines begin from cycle 0 and do not progress to the next cycle until they exhaustively check the full state space within that cycle as shown in the figure 2.

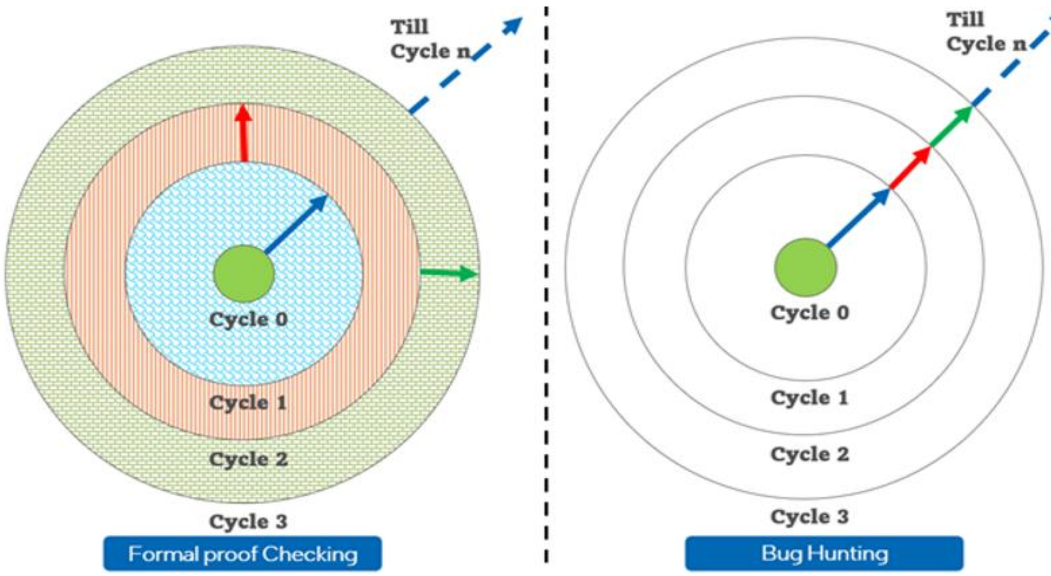

Figure 2: Comparison of formal proof and formal bug-hunting

Given the substantial size of our design, approximately 50% of our RTL assertions faced challenges due to their complexity and were not fully proven. To uncover deep issues, we conducted bug-hunting exercise on RTL assertions and the forward progress checkers using interesting functional covers as re-located start points of the formal search.

In order to increase the reached proof depth of bounded assertions, a technique involving the use of already proven assertions as helper assertions was employed. This technique involved converting already proven assertions into assumptions, which were then used in conducting regressions with the remaining assertions. By leveraging proven assertions as helper assertions, the following benefits were achieved:

1. More full proofs: Full proof was achieved for additional 3% of bounded assertions, indicating that more aspects of the design were verified to be correct.
2. Enhanced proof depth: More than 15% of the bounded assertions attained higher proof depth, further enhancing the confidence in the design's correctness.

V. Results

"ABC Formal" approach was deployed by four engineers spending half of their bandwidth over two quarters to produce stellar results on a simulation clean design. Using this approach, seven bugs were found during this activity with two showstopper bugs – one deadlock and one starvation each. We were also able to improve the stimuli coverage by 49% and checker coverage by 24%. Overall, the results detailed in the section helped improve the design quality many folds with minimal resources and within limited time.

A. *Sign-off status*

Table 1: FV environment status at sign-off

Type	Statistics	Status
RTL embedded assertions	6019	60% proven, 40% bounded
RTL embedded covers	7892	100% covered
End-to-end Checkers	9	100% bounded
Interface assumptions	402	100% validated in dynamic simulation
Design Assumptions	389	100% validated in dynamic simulation

B. *Deadlock bug*

A corner-case deadlock scenario was flagged by forward progress checker when two independent (but later realized as conflicting) request called "force_bridge_to_idle" and "block_ingress_req" sent to the bridge in the exact same cycle.

"block_ingress_req" is a request from fabric to bridge to drain all transactions from ingress buffer. Bridge is expected to acknowledge "block_ingress_req" with "block_ingress_ack" once ingress buffer is empty. "force_bridge_to_idle" is a request made by memory interface to bridge to quiesce the interface and remain asserted till bridge declares that the "bridge_is_idle". However, bridge cannot declare "bridge_is_idle" till there is a pending transaction in bridge.

As one request is trying to bring bridge to idle and another one preventing that, design gets stuck in an inescapable state. This bug was a result of an omission in the architecture/specification of the bridge; hence design implementation never considered this case and did not implement a tiebreaker.

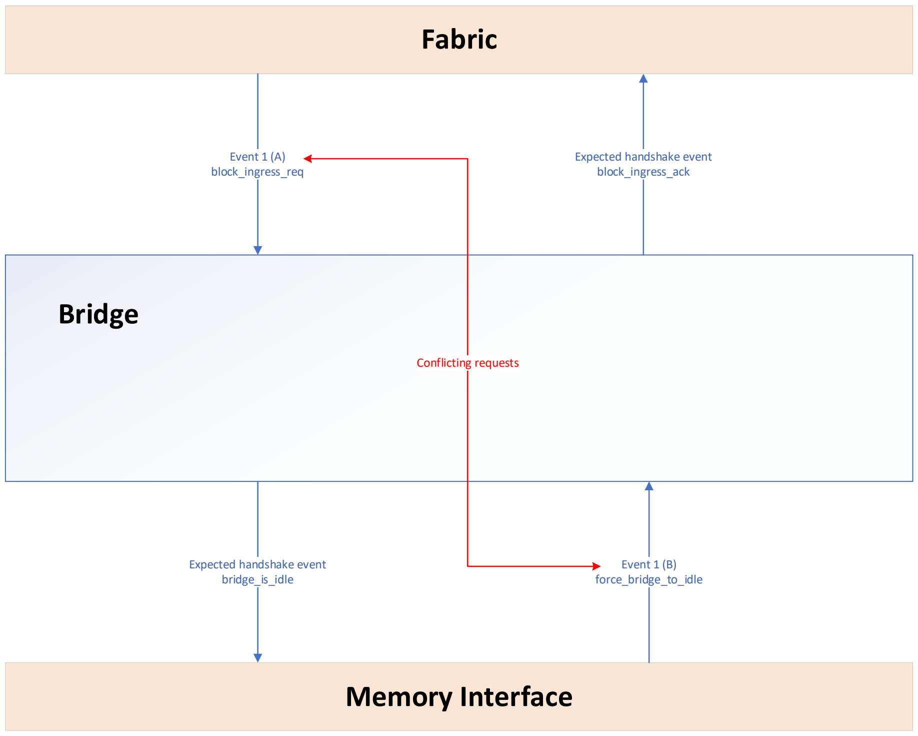

Figure 3: Deadlock bug explanation

C. Starvation bug

A corner case starvation scenario was caught by an RTL embedded assertion where anti-starvation timer of entry 0 of “transaction tracker” could never timeout due to incorrect implementation in the RTL. Bug details are as follows:

1. Bridge allocates a unique tracker id to each transaction. The “alloc” signal is asserted for every tracker id that is being allocated at different clock cycles.
2. When a transaction is allocated to entry 0 of the tracker, tracker allocation signal “alloc_vec[0][0]” gets asserted and “entry_valid[0]” goes high indicating a valid entry for tracker id 0.
3. Since the “alloc_ptr” signal is tied to zero at instantiation therefore, the “entry_enable[0]” signal will keep getting asserted every time a new allocation happens
4. This will keep on refreshing the value of the signal “entry_time_stamp” and thus the transaction on tracker id 0 will never timeout as indicated by “timeout_entry_vec[0]” as far as new transaction keeps coming in.
5. If a transaction is allocated to the tracker entry id 0 then it is vulnerable to starvation which can lead to a need of cold reset of the entire SoC.

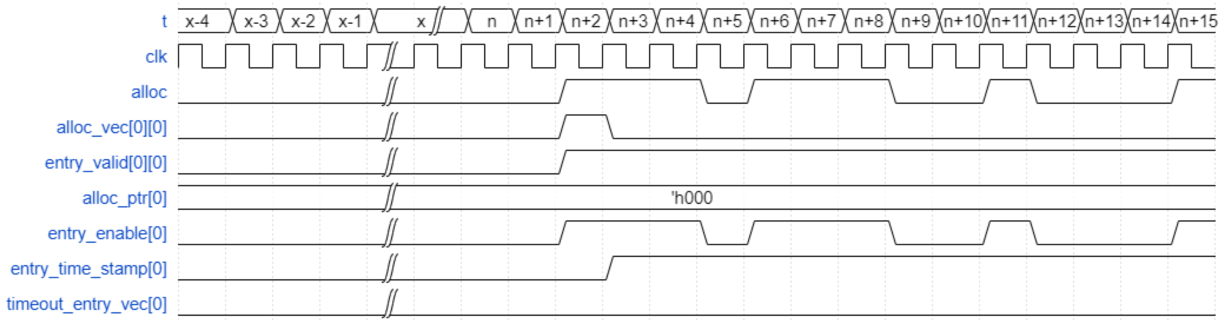

Figure 4: Starvation bug waveform

```
trkr_timeout (
   .clk,
   .reset,
   ...
   .alloc (trkr_req_vld),
   .alloc_ptr('0),             // Bug
   .alloc_vec(trkr_entry_vec),
   .entry_vld(trkr_entry_vld),
   ...
);

for (genvar entry=0; entry < NUM_ENTRIES; entry++) begin
   always_comb begin
      entry_enable[entry] = '0;
      for (int i=0; i < ALLOCS_PER_CYCLE; i++) begin
         entry_enable[entry] |= alloc[i] & (alloc_vec[i][entry] | (alloc_ptr[i] == entry)); // Bug
      end
   end
end
```

VI. Conclusions

We presented a comprehensive guide for improving design quality through formal verification, even when starting near the tape-in date with impactful results highlighted in the case study. We show-cased how bug hunting techniques using existing RTL assertions and end-to-end checkers can be done without much design ramp-up time. In the case study, showstopper issues were discovered just in time that saved millions of $ by preventing re-spins. And emphasizes the importance of formal verification and the benefits of formal bug-hunting techniques in the late stages of a project for engineers and managers involved in critical projects. This last minute FV approach helped us in getting a good head start in the next generation of design.

Acknowledgment

Thanks to the design team and FVCTO Team for the motivation and the valuable feedback.

References

[1] Viraj Athavale; Sai Ma; Samuel Hertz; Shobha Vasudevan, "Code coverage of assertions using RTL source code analysis," 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC).

[2] Ana Magazinius; Niklas Niklas Mellegård; Linda Olsson, "Bug Bounty Programs – A Mapping Study," 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)

[3] N. Wu; MengChu Zhou, "Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems," IEEE Transactions on Robotics and Automation (Volume: 17, Issue: 5, October 2001)

[4] Rajeev K. Ranjan; Claudionor Coelho; Sebastian Skalberg. "Beyond verification: Leveraging formal for debugging," 2009 46th ACM/IEEE Design Automation Conference