2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Cherry-picking Assertions to Enhance Convergence

Ayush Vajpeyi
NVIDIA Corporation
Bengaluru, India
avajpeyi@nvidia.com

Muralikrishna Kukkupuni
NVIDIA Corporation
Bengaluru, India
mkukkupuni@nvidia.com

Sangamesh Kokatnur
NVIDIA Corporation,
Bengaluru, India
skokatnur@nvidia.com

***Abstract*- This paper explores a targeted approach to formal verification of the Miscellaneous Node in an ARM CHI-based chip, which manages the correct ordering and movement of Distributed Virtual Memory (DVM) operations. A key verification goal is to ensure forward progress of DVM invalidation snoops under constrained conditions. During verification, convergence issues arose due to the complexity of the design and the interdependent assertions. To address this, we propose a method of 'cherry-picking' key assertions – selectively choosing and enabling them using Compositional Assume-Guarantee (CAG) reasoning and formal profiling techniques. This strategic assertion refinement significantly improves convergence without compromising coverage. We further outline a generalized methodology for applying this approach to achieve verification completeness in complex designs.**

I. Introduction

In Formal Verification for complex systems achieving convergence is a significant challenge [1]. There are many convergence techniques known industry wide, for example counter abstraction, black-boxing large arrays, writing helper properties, assume-guarantee, case split etc. One can apply these techniques based on the design knowledge [2]. Due to the high Cone of Influence (COI) of the assertions, convergence issues arise, often resulting in bounded verification results. This paper focuses on verifying a chip based on ARM CHI, specifically the Misc Node, which ensures that Distributed Virtual Memory (DVM) operations are ordered correctly and are free from deadlock. A key assertion checks the forward progress of DVM invalidation snoops under specified conditions. This paper talks about how the combined use of Compositional Assume-Guarantee (CAG) and Formal profiler helped in solving the convergence issue. Here we are proposing a standardized solution for solving the convergence issue and how one can achieve verification completeness.

II. Related Work and Application

A. Compositional Assume-Guarantee (CAG)

This novel method allows a group of selected assertions (or all assertions in a task) to act as helpers for each other. With CAG, each property in the selected property set acts as a helper for all other properties. But CAG does not just assume each one and use it to prove the others that would be circular reasoning. Instead, it does the following:

- Prove that each assertion is independently true at cycle 1, just after the reset.
- Prove each assertion is true at bound N+1 and assume all assertions are true up to N.

Thus, the properties are helping each other to 'climb the inductive ladder' [3]. The below illustration helps to summarize the method:

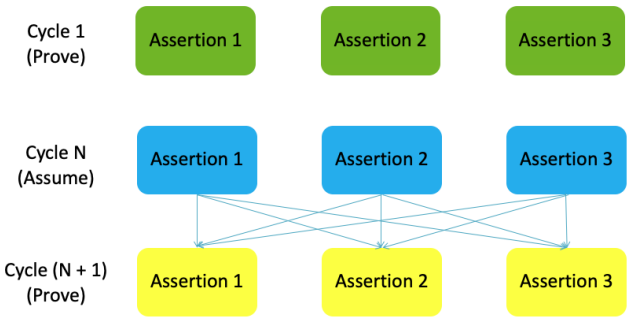

Figure 1. Compositional Assume Guarantee

If a set of assertions is proven through CAG and any of the proofs are bounded, the complete set of proofs is only valid to the depth of the lowest bound. If all are fully proven, the proofs are logically valid. Proof propagation is considered here. The assumption of all properties being true at one cycle to prove the next also means that a single failing property in the CAG set breaks all the other proofs, they cannot be considered valid past the bound that represents the counterexample. The picture below illustrates this situation:

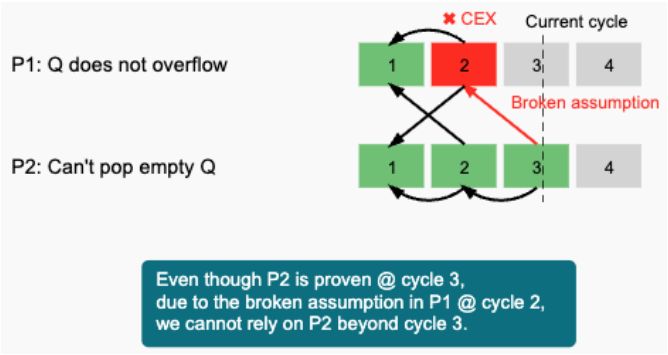

Figure 2. Scenario when a failure is encountered in CAG

This means that using CAG introduces mutual dependencies between properties that are involved in the operation. Hence, if we simply add all our testbench assertions into a single CAG node, it will not yield good results, and any bounded assertion invalidates the proof of other assertions.

B. Profiling the effort spent by Formal Engines

The optimization of formal verification proof times follows a systematic, iterative flow that commences with the initiation of a formal proof, during which the verification engine begins processing the specified properties. As the proof execution progresses, the system may encounter properties that exhibit unexpectedly long verification times, triggering the need for performance analysis. We then transition to a profiling phase, where formal profiling tools (example: Formal Profiler provided by Jasper [4]) are deployed to analyze the computational effort of the verification engine. This analysis reveals specific bottlenecks, that is, signals in the testbench that require the most computational effort. Based on these identified bottlenecks, the verification problem is strategically simplified through the application of additional assumptions, helper assertions, or abstraction techniques. This new verification problem is then fed back into the proof engine, initiating a new verification cycle that produces updated results. These results either indicate successful verification completion or signal the need for further optimization through consecutive iterations.

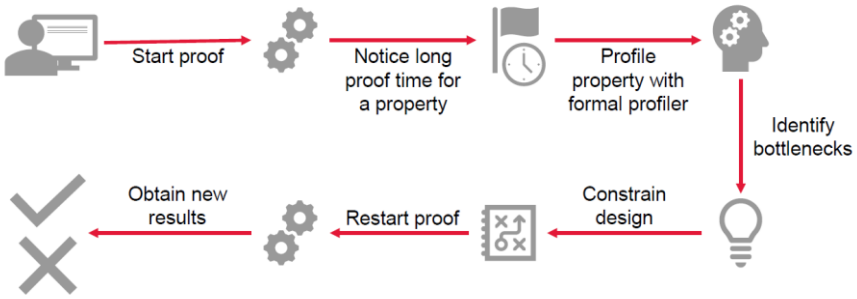

Figure 3. Flow detailing how profiling formal effort can be used

III. Proposed Solution – Use of CAG and Profiling

We introduce an approach to improve convergence during formal verification by distilling proven assertions from Compositional Assume-Guarantee (CAG) nodes and leveraging the formal profiler to filter out bounded assertions, concentrating efforts on proving the CAG node effectively. CAG involves using assertions as helpers for each other in an inductive proof methodology. By distilling proven assertions into new CAG nodes and using the formal profiler, we can identify critical bounded assertions and focus on those that impact proof complexity and convergence.

The workflow can be illustrated as:

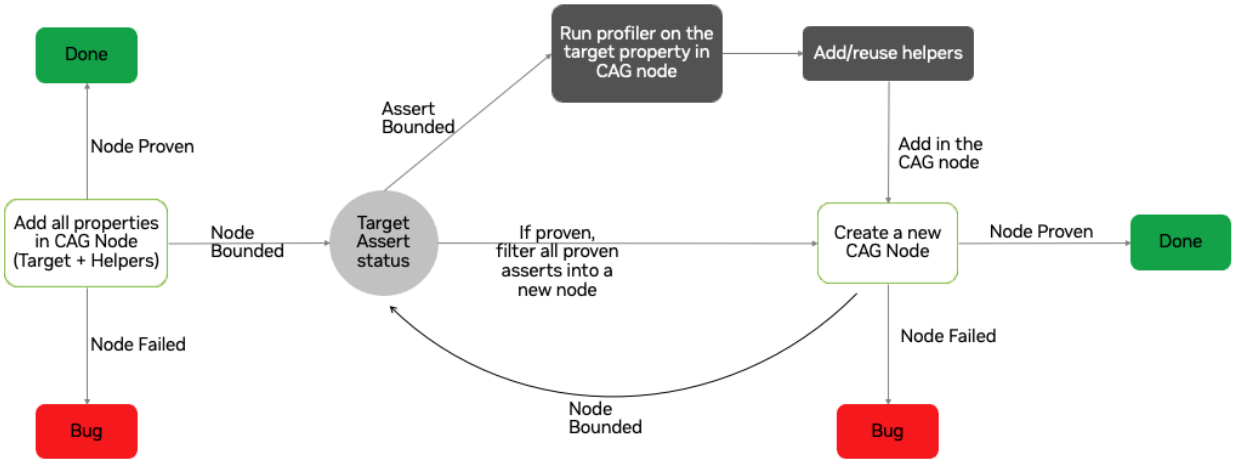

Figure 4. Workflow on how to distil CAG results

According to CHI spec, "All previous DVMOp requests whose completion needs to be guaranteed by the DVMOp(Sync) must have received a Comp response before the Request Node can send a DVMOp(Sync)". As a result we have to ensure that the DVMOp requests are deadlock-free in MN else we can create a system-level hang.

The Formal Verification (FV) Testbench has an overall deadlock check on DVM transaction making progress when all the requirements are met [5]. This property has poor bounds due to the high COI associated with it. Additionally, multiple assertions are made on internal components of the design, such as FIFOs, arbiters, and counters. When combined, these assertions result in a node with 100 properties, including the top-level deadlock check and various checks to ensure proper functionality of internal design components. A simple batch run of all properties proves some, while others remain bounded at different cycles. Subsequently, assuming the proven assertions and re-running the bounded properties yields incremental improvements, but these results are still insufficient for signoff.

IV. Results

Results for MN				
Node	Total Assertions	Proven	Bounded	Failing
CAG Node 1	100	54	46	-
CAG Node 2	54	49	4	1 **(true fail)**
CAG Node 3	47	47	-	-
CAG Node 4	5 (4 bounded asserts from Node 2 + 1 bounded asserts from Node 1)	4	1	-
CAG Node 5	9 (Node 4 + 4 new helpers)	-	-	-
CAG Node 6	59 (Node 2 + 1 asserts from Node 1 + 4 new helpers)	59	-	-

By applying this approach, we reduced the number of assertions from an initial set of 100 (CAG Node 1) to 54 proven and 46 bounded assertions. After selecting the 54 assertions for a new CAG node (CAG Node 2), we observed 49 proven, 4 bounded, and 1 failing assertion, which was a deep sequential scenario never encountered in regression. From 49 proven assertion, 47 assertions were filtered into a new CAG node (CAG node 3) after structural analysis, and the node was completely proven. From CAG node 2, the bounded properties were analyzed in Formal profiler and a bounded assertion from node 1 was matched with where most proof effort went. A new CAG node (CAG node 4) with 4 bounded properties + original bounded assert proved the 4 bounded assert but kept the original bounded assert in the same state. Passing the original bounded assert in Formal profiler, helped us come up with 4 new helper asserts that converged CAG node 4 and CAG node 2 completely.

Results for Linked-List based design				
Node	Total Assertions	Proven	Bounded	Failing
CAG Node 1	9	3	6	-
CAG Node 2	3	2	1	-
CAG Node 3	59 (Node 2 + 56 helper assertions)	59	-	-

The same technique was applied to another design containing linked lists. Linked lists are known to be difficult for Formal Tools to converge on. Running all the properties in the DUT under CAG Node 1 proved 3 linked-list properties but also had 6 bounded asserts. A critical property related to adding a new entry to the linked list, which had a low bound in regressions, was successfully proven in CAG Node 1 (but since the node had bounded properties, the proof is over-constrained). We distilled the proven assertions into CAG Node 2 and saw that the critical property was again bounded. By using Formal profiler, we identified that an arbiter contributed the most to the proof complexity. By combining the arbiter-related properties (56 helpers) with the 3 proven properties in CAG Node2, we were able to prove the entire CAG node (Node 3). Initially, this property would have required multiple abstractions and several days of effort; however, the technique reduced the process to just one hour of runtime.

V. Conclusion

By profiling the formal effort in conjunction with CAG, we significantly reduced the number of bounded properties in the verification process, focusing only on those that directly impact the target assertion, thereby improving verification efficiency and convergence. Thus, for proof with higher cone of influence, verification using above methods one can achieve verification completeness without tampering the exhaustive nature of formal verification. The problems like complexity of the algorithm, convergence can be there due to many other reasons, but this paper gives a generic and unique way of approaching them without doing any design modification.

VI. Future Work

The iterative nature of this method makes it the perfect candidate for automation. An AI workflow can be put in place to distil the CAG nodes and monitor the progress of proofs over a fixed amount of time and decide on next steps.

Secondly, extracting high effort signals from proof profiling and designing non-tautological and non-vacuous helpers is another area where an automated workflow can assist.

REFERENCES

[1] Pritam Roy, et al, “Hierarchical Formal Verification and Progress Checking of Network-On-Chip Design”, Nvidia, DVCon 2025
[2] Ipshita Tripathi, et al., “Process & Proof for Formal Sign-off - Live Case Study,” Oski, DVCon 2016
[3] Erik Seligman, Karthik Baddam (2024), “Next-Generation Formal Property Verification: Lightweight Theorem Proving Integrated into Model Checking”, DVCon US
[4] “Jasper Platform and Formal Property Verification App User Guide”, Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA
[5] Saurabh Chaurdia, Arun Khurana, et al., “Detecting Circular Dependencies in Forward Progress Checkers”, Nvidia, DVCon 2021