2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Catching the Unseen: A Case Study on Conquering Caching and Ordering Verification Challenges in Release Critical Unit

Ayush Saraogi, Gaurav Borkar, Vivek Singh
NVIDIA, India
{asaraogi, gaborkar, viveks}@nvidia.com

***Abstract*- This paper sheds light on the innovative formal verification approach to overcome thorny challenges of verifying caching and ordering mechanisms within modern SoCs, where design complexity arises from highly interconnected components, diverse data structures, intricate pipeline stages, and sophisticated ordering rules governing request handling. Traditional methodologies including dynamic simulation techniques and testing struggle to expose the full spectrum of scenarios and subtle corner cases required for robust verification. In this paper, we introduce generic and scalable techniques that tackle these challenges, resulting in a holistic end-to-end solution for the formal verification of caching and ordering logic. Our proposed framework, validated through application, uncovered over 100 bugs—including numerous critical and deeply buried corner-case issues—and enabled timely signoff. These results underscore the significance and effectiveness of formal verification in handling the challenges of complex designs and achieving comprehensive verification in advanced SoC architectures.**

I. INTRODUCTION

Modern System-on-Chip (SoC) designs are rapidly evolving, pushing the boundaries of performance, efficiency, and integration. As these chips become the heart of everything from data centers to everyday consumer electronics, their internal subsystems—especially caching and ordering mechanisms—grow in complexity and importance. In these high-performance systems, ensuring that data moves efficiently and correctly between numerous interconnected design blocks is no simple feat**.** Caching and request-ordering logic now juggle a host of challenges: tightly coupled modules with diverse data representations, a maze of control decisions, and intricate cache operations that rely on sophisticated staging and deep pipelining.

Subsystems that manage multi-dimensional linked lists or replay buffers, where requests cycle in and out, vying for limited cache slots. Only select requests, filtered by intricate criteria, are granted space, while others are rerouted for another attempt—creating a dynamic, ever-changing flow of data. Compounding the situation are request buffers employing multi-virtual channel (multi-vc) backpressure techniques, dynamically responding to credit availability per channel to maintain system balance. All these operations intertwine with custom protocols, such as specialized adaptations of the CHI interface, making plug-and-play verification testbenches infeasible. On top of this, unique ordering requirements—often dictated by subtle relationships between request sizes and addresses—generate countless combinations of overlapping, blocking, and non-blocking transactions, vastly expanding the space of potential behaviors**.**

Traditional verification methods, grounded in dynamic simulation and directed testing, have proven invaluable—but their limitations are clear in the context of such sprawling complexity. The sheer number of possible interactions and rare corner cases make exhaustive coverage unattainable within project timelines, and subtle bugs may lurk undetected until late in the design cycle—or even after silicon hits the field. This is particularly true when verifying forward progress and correctness in blocks governed by deep pipelines and tightly coupled control logic.

Recognizing these challenges, we set out to develop a robust and scalable formal verification methodology, specifically tailored to manage the multifaceted demands of modern caching and ordering subsystems. Our proposed

approach breaks down the complexity of these systems, guiding systematic property development, classification of corner cases, and modular proof strategies.

We applied this methodology with great success to the NVIDIA's unit in the advanced tegra SoC family. Not only did our framework enable complete and confident signoff, but it also exposed around 100 unique issues—many buried deep in the design and unlikely to be discovered through simulation alone. This included a range of critical bugs and intricate corner cases involving replay logic, protocol handling, and fine-grained order management.

The results are clear: as SoCs continue to grow in scale and interconnection, robust formal verification is essential for delivering reliable, high-performance products. Our experience demonstrates that, with the right approach, even the most intricate designs can be verified efficiently and thoroughly, allowing teams to meet ambitious deadlines with confidence.

II. Problem Statement

Caching and Ordering mechanisms are integral components of many modern-day SOCs. Though they are key to boosting the performance of the system, their verification comes with its own set of significant challenges. Based on our experience of verifying these designs the following challenges stand out

1. *Design Complexity*: These designs usually have deeply interconnected components with complex interactions between the different design blocks, each with different data structures and control decisions. Verifying such complex web of interactions is a complex task

2. *Staging Pipeline*: The staging pipeline of the cache is susceptible to hazard which can be difficult to verify. The outcome of later requests in the pipeline might depend on the outcome of earlier requests.

3. *Ordering Rules*: There are intricate rules to define the order in which operations from the different system agents appear to execute. While these rules are essential for writing concurrent software, they can create the most difficult hardware verification problems

4. *Deep Corner Case Scenarios*: There can be numerous deep corner cases due to pipeline stage of cache, ordering rules and different agents trying to access the same resources. These deep corner cases are exceptionally challenging to verify because traditional dynamic simulation has a very low probability of generating the precise sequence of events and timing needed to trigger them.

These challenges combined with reducing time to market pose significant difficulties for dynamic simulations. It is less probable to encounter various scenarios and deep corner cases in dynamic simulations. Hence FV becomes mandatory to improve verification efficiency and ensure timely signoff in such units. In this paper, we present a structured formal verification (FV) strategy that enabled successful sign-off of a large unit. Our approach uncovered 100+ bugs with many critical and deep corner-cases and ensured timely sign-off, validating the effectiveness of FV in conquering design complexity and achieving robust verification results.

III. Methodology

A. Early Sanity Checks and System Oversight

We began by aggressively targeting vulnerabilities in the design's early stages, employing FV-friendly end-to-end modelling and checks. By leveraging advanced techniques such as Coloring, Pulse, and Free Variable methods, we validated critical system behaviors—including address interlocking guarantees—right from design bring-up. This proactive phase allowed us to catch "low hanging" bugs and design issues before they could propagate deeper into the system, setting a strong baseline of reliability early.

B. Hierarchical Testbench Development

Recognizing the complexity and interdependency of the design, we decomposed the system into functional blocks and constructed highly targeted internal testbenches for each. This modular approach enabled us to drill deep into individual components, exposing subtle corner-case scenarios and ensuring complete coverage. The structure not only reduced the verification burden but also created reusable assets that boosted future productivity.

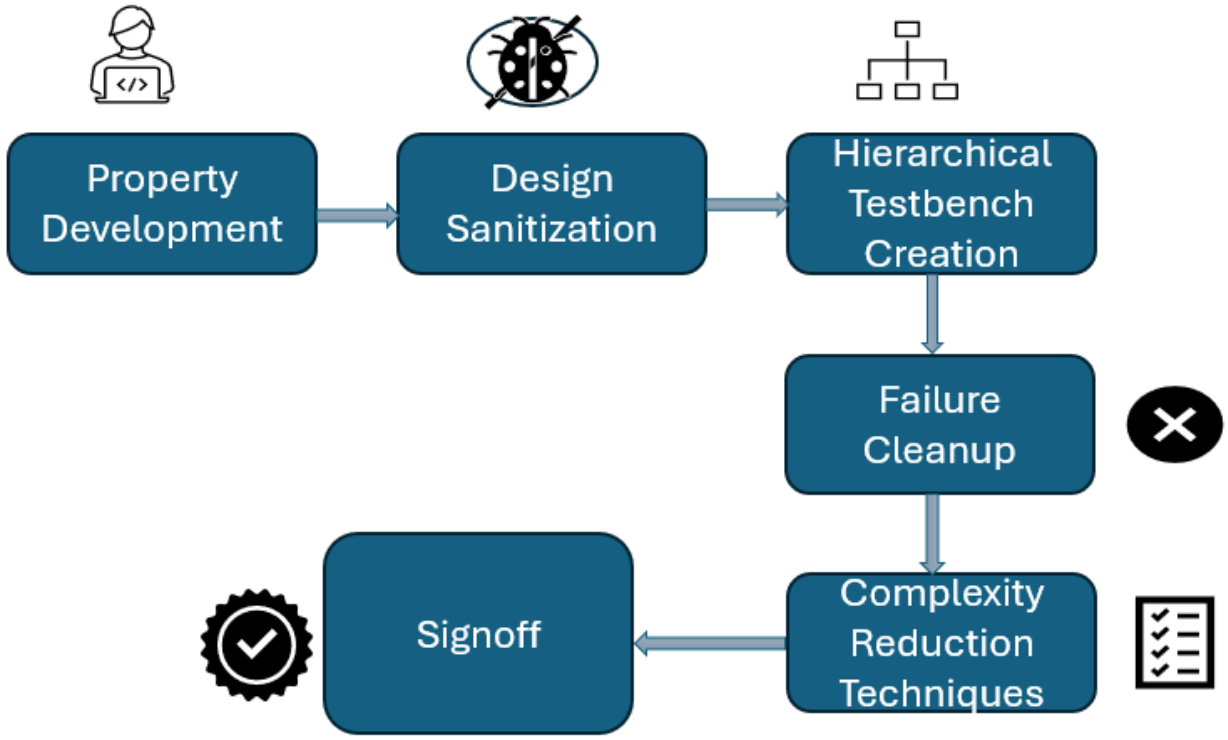

Figure 1: Formal Verification Methodology

C. Tackling Complexity with Scalable Techniques

The core strength of formal verification **[1][2]** is exhaustive, breadth-first sweep through all states, guaranteeing no reachable corner is ignored. However, this strength inherently leads to state-space explosion**[4],** where design complexity (size, COI) can cause an exponential growth in states. This can sometimes make it practically impossible for the formal tools to reach the interesting deeper scenarios.

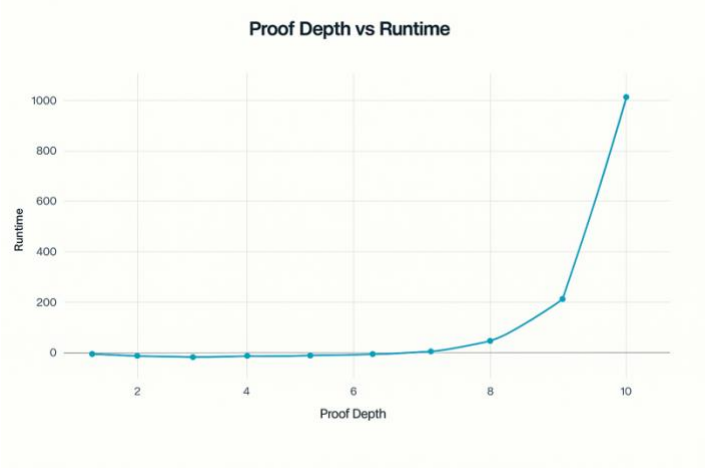

Figure 2: Complexity due to state space explosion

To manage the inherent intricacies of the caching and ordering subsystems, we approached complexity on several fronts:

1. Hierarchical Property Composition: Internal properties from module-level test benches were leveraged to help complexity for comprehensive, end-to-end properties, linking fine-grained checks to overall system correctness. Internal Properties from hierarchical testbench were used as helpers for End-to-End properties

2. Configurable and Diverse Traffic Generation: We engineered flexible traffic modes with various patterns, which enabled Cone-of-Influence (COI) partitioning. This helped to isolate different design structures and target challenging behaviors systematically.

3. Design Abstraction and Proof Convergence: Abstraction techniques were central to breaking verification bottlenecks—facilitating convergence and actively uncovering deep bugs. Key abstractions included:

    a. Carefully reducing the number of cache sets, ways, and other data structures to manage state space explosion, while not losing out on any scenarios.

    b. Preloading the cache using RVAs so that requests can be allocated in the cache from the reset state itself, hence exposing more cache control decisions related corner cases at lower bounds. This helps to reduce the Required Proof Depths for the different asserts.

    c. RVAs on FIFOs and Free list, thus bringing more scenarios closer to reset state.

    d. Interlocking structure Abstraction Model to maintain chaining of only the symbolic requests, saving on lot of design flops

    e. Different configurate modes with different traffic patterns for functional and COI splitting, enabling more bug-hunting at higher bounds

By layering these strategies—early system-level validation, modular testbench granularity, hierarchical property reuse, purposeful traffic modelling, and strategic abstraction—we created a robust, adaptable framework. This

structure empowered us to conquer design complexity, uncover deeply embedded bugs, and ensure complete, confident signoff, all while meeting aggressive project timelines.

D. Forward Progress Solution

Traditional forward progress verification often encounters false failures at the lower bounds, primarily due to known limitations of caching control decisions and dynamic prioritization within the system. These challenges are further compounded by the limited observability of cache hit/miss events and other internal control operations at the end-to-end boundary, making it difficult to mask false failures in the conventional strategies effectively. Our EOT-based approach directly addresses this gap. It overcomes these obstacles by orchestrating a four-step sequence as shown in Figure 3

Figure 3: EOT Framework

A floating pulse is used to mark the End of Traffic. This EOT pulse can come arbitrarily at any point. No further input transactions are sent to design after this end of traffic pulse is seen. Due to ordering requirements, the transactions inside the design might be waiting for the completion responses of the previous transactions. These completion responses are also sent quickly and not stalled after the EOT pulse is seen.

As shown in Figure 4, in step 4 an assert is written to check that all transaction should be drained out of design once EOT pulse is seen within N cycles. This Worst-Case Latency (N) needs to be calculated based on the design micro-arch and the chaining requirements supported.

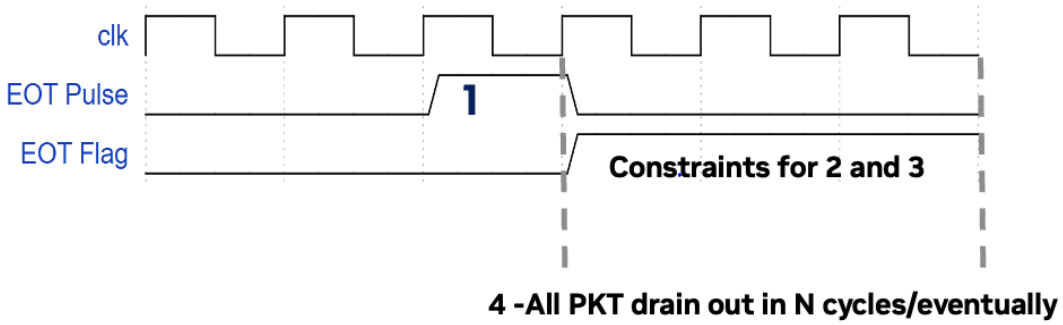

Figure 4: End Of Traffic Pulse

IV. Case Study

Our work is on a design that has complex array of logic blocks collaborates to manage multi-dimensional linked lists and replay buffers, ensuring that every memory request is either swiftly admitted into limited cache slots or cycled back for another chance.

The sophistication does not stop there: Ever-evolving backpressure, with credit-based resource arbitration across multiple virtual channels, employing fine-grained pipelining and staging to ensure that priority and bandwidth needs are balanced for each client. The structure also enforces special ordering requirements.

A. Addressing Verification Challenges (Complexity)

The Figure 5 below shows a generic design which implements catching and ordering related features along with formal techniques used to address complexity. The green dotted lines represent separate independent TB boundaries like E2E, cache, interlocking structure, etc. The blue dotted line shows the various abstractions like design reductions, preloading, interlocking structure abstraction models etc.

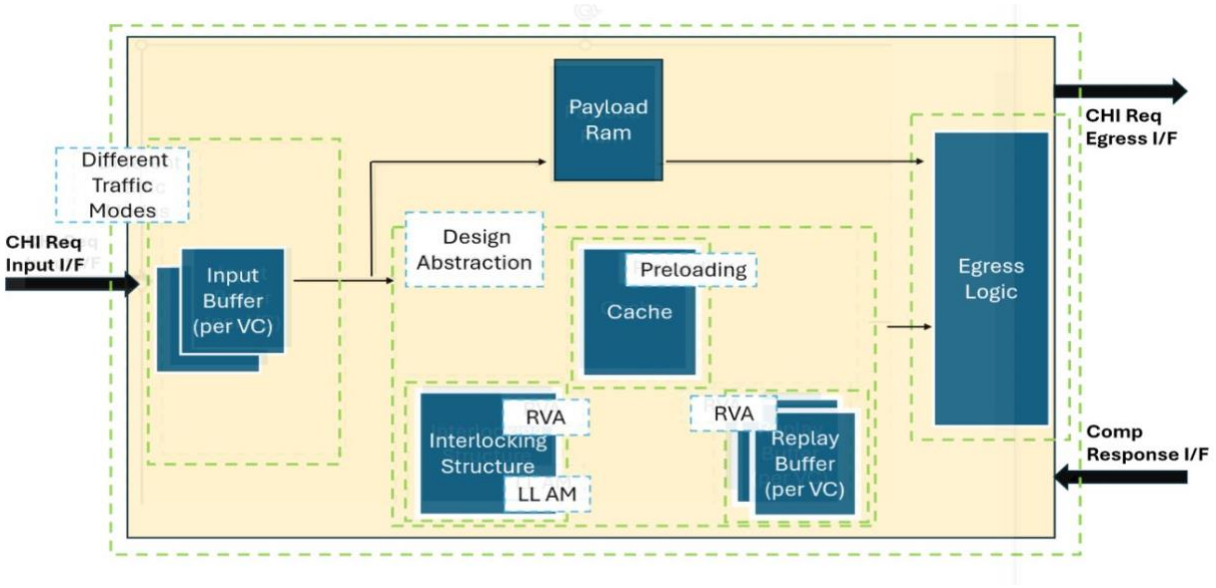

Figure 5: Template Caching and Ordering Based Design with Complexity Reduction Techniques

A. Interesting Bugs

Bug #1: Request Stuck in Interlocking Structure

In rare, timing-sensitive scenarios, a subtle race condition can leave a request stranded inside the interlocking logic. Here, request R1 arrives at the cache, misses, and is sent out. Later, as R1's response returns, a new overlapping request, R2, arrives in the very same cycle. R2, spotting the overlap, correctly enters the interlocking structure, expecting to be released when R1's completion arrives.

But here's the twist: since R1's response and R2's arrival coincides, the interlocking structure pops for R1 and completely misses R2's mark. With no corresponding completion to trigger its release, R2 becomes trapped stuck

indefinitely and never making progress. Hitting this corner case in practice takes precise cycle-for-cycle timing, making it hard to produce, but critical to avoid hidden hangs deep in the pipeline. As shown in figure 6

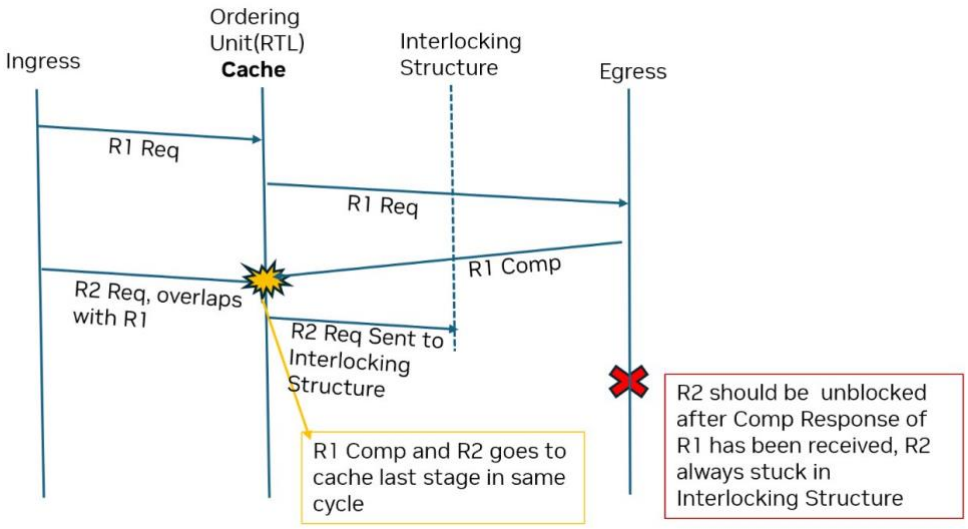

Figure 6: Request Stuck in Interlocking Structure

Bug #2: Replay Buffer Never Gets Popped

Here the design implementation is such that once the comp rsp comes, all the entries of replay buffer start popping sequentially and some gets allocated in cache and rest are sent back to replay buffer. But here's what happened: as shown in Figure 7, when comp for R1 comes, replay buffer starts popping and R12 to R20 goes to cache. Only the first replayed request (R12) secures a vacant way; the rest (R13 to R20) are shuffled back into the replay buffer. Now, with no outstanding completions left, the replay bit stays set, yet there are no new triggers to pop the buffer. The remaining requests are left stranded caught in a limbo with no way out.

This elusive scenario, only revealed through abstraction techniques. To uncover such a bug: you need to fill the set's ways, funnel enough requests into the replay buffer, and align comp responses with uncanny timing. It's a classic example of a corner case hidden at deeper bounds.

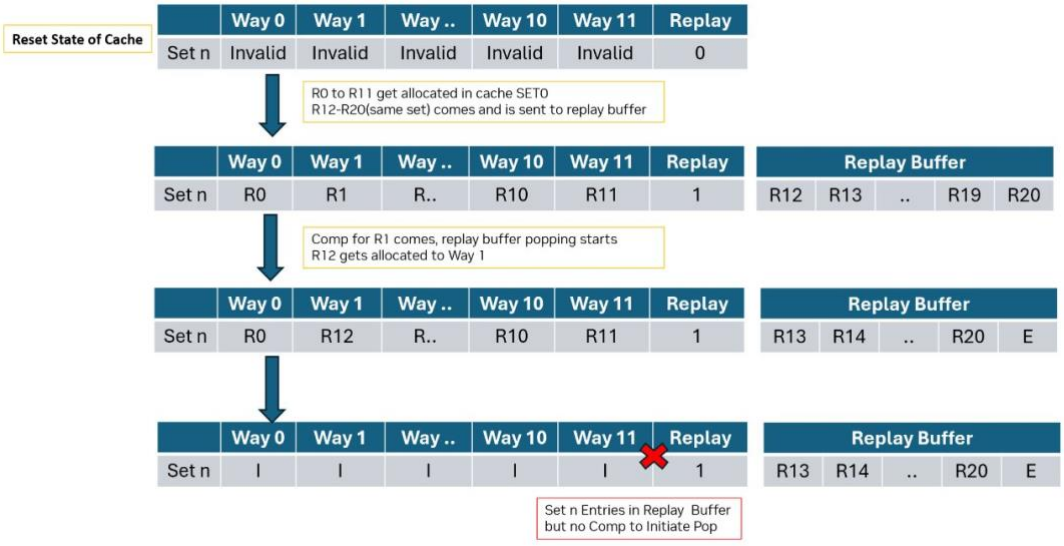

Figure 7: Replay Buffer Never Gets Popped

Bug #3: Queue Full Condition Not Considering Reserved V

Hidden deep in the design, we found a rare request drop in interlocking structure that can quietly trip up both ordering and forward progress issue. Here's how it unfolds: requests overlapping in the cache are funneled into specialized queues—one per address—each with a reserved slot for every virtual channel. Now, a scenario where nearly every slot is packed with VC1 traffic, except for one spot reserved for VC2. With just two empty spaces left and three requests surging down the pipeline, something serious happens—the final queued request overwrites its predecessor, breaking the expected order and leaving data behind.

To uncover, it needs a perfect lineup: a nearly full queue, a trio of requests arriving in sync, and all of them tagged for VC1 and in the same cycle, cache pipelines should be filled, and those requests must go to queue and those should be sent over vc1 not on vc2. Shown in figure 8

Queue				
0	1	2	..	Q
VC1	VC2	Any		

Faulty Queue Full Condition – Num_Used_Queues > Q – NUM_CACHE_PIPES(3)

All Requests come with same VC

Queue							
0	1	2	..	Q-3	Q-2	Q-1	Q
VC1	Empty	VC1				Empty	Empty

3 REQs in cache pipelines with VC1 all going to new queues. Last push to queue would overwrite ✖
Ordering and Forward Progress Issue

Queue							
0	1	2	..	Q-3	Q-2	Q-1	Q
VC1	Empty	VC1				VC1	VC1

Figure 8: Queue Full Condition not considering Reserved VCs

V. RESULTS

By deploying our formal verification techniques on two key units, we accelerated the verification cycle and achieved guaranteed FV sign-off, which proved instrumental in mitigating risks for critical IPs. This methodology led to a remarkable time savings of **12–14 weeks**. **102 critical bugs with 20 deep corner cases** were discovered with 5 months of single engineer efforts, giving us a very high ROI. The comprehensive scope of the bugs uncovered and no late stage FV escape also confirm the efficiency and effectiveness of our strategy.

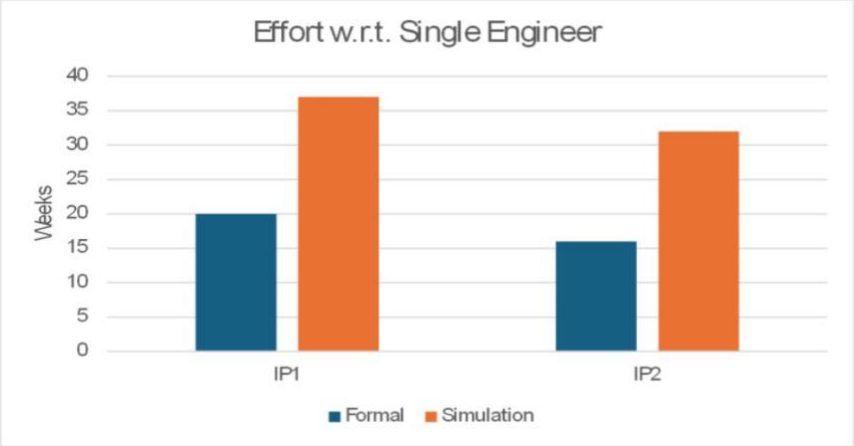

Figure 9: Simulation vs Formal efforts

VI. CONCLUSIONS

This paper validates a holistic and highly effective formal verification strategy that encourages a formal-centric design methodology with early FV planning. The former addresses the inherent challenges of formal verification by implementing clean interface definitions, modular hierarchies, and systematic parameterization to simplify the problem space. The latter complements this by leveraging design decomposition and strategic abstractions to enable comprehensive and complete sign-off of large, complex units. The synergy of these two pillars—a well-architected design and a proactive verification plan enables guaranteed FV sign-off, high ROI, and the consistent discovery of critical bugs. This work provides a clear and repeatable blueprint for overcoming the most pressing challenges in modern verification using formal techniques.

References

[1] E. Seligman, T. Schubert and M.V.A. Kiran Kumar, Formal Verification: An Essential Toolkit for Modern VLSI Design, 2015

[2] M. Girish, G. Gopakumar and D. S. Divya, "Formal and Simulation Verification: Comparing and Contrasting the two Verification Approaches," *2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS)*

[3] IEEE Std 1800™-2017, IEEE Standard of System Verilog – Unified Hardware Design, Specification, and Verification Language

[4] Z. Xin-feng, W. Jian-dong, L. Bin, Z. Jun-wu and W. Jun, "Methods to Tackle State Explosion Problem in Model Checking," *2009 Third International Symposium on Intelligent Information Technology Application*, Nanchang, China, 2009, pp. 329-331, doi: 10.1109/IITA.2009.50.