End to End Formal Verification Strategies for IP Verification

Jacob Ryan Maas[1] Nirabh Regmi[2] Ashish Kulkarni[3] Krishnan Palaniswami[4]

Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052
[1]jamaas@microsoft.com
[2]nirabhr@microsoft.com
[3]ashishku@microsoft.com
[4]kp@aakay.net

Abstract: IP Verification traditionally includes some form of constrained random verification methodologies like UVM, and may also include formal verification for a portion of the design. However, there is a typical lead time to putting all verification infrastructure together before the first random tests are run, and coverage closure is also time consuming. Formally verifying a portion of the design helps to reduce the coverage space for the UVM testbench, but still requires a full UVM testbench infrastructure, and also requires additional resources for the formal verification. In this paper we present end to end (E2E) formal methodology as a viable option for verifying an IP using only formal tools. We use two IPs, crossbar (XBar) and interrupt controller (INTC), to present empirical evidence on the validity of our methodology, and contrast that with other IPs verified using standard UVM. We also present some of the challenges encountered in deploying E2E formal on these two blocks, and discuss our strategies to overcome the issues. We also provide some recommendations for deploying E2E formal methodologies and conclude that our E2E Formal methodology is able to provide at least the same verification quality as compared to UVM simulation based approach, while having a significant shorter verification cycle.

Introduction

In our approach for E2E formal verification, IPs are described fully using formal properties based off the IP specification. Registers are checked with automatically generated assertions, from an IPXACT XML file. Standard interfaces are checked with formal property proof kits, instead of VIP. The verification environment and the IP are refined in parallel as the DVE writes new assertions, and counterexamples to assertions are debugged. The quality and completeness of these assertions is checked by frequent reviews against the IP specification with the design team. Once all formal properties are signed off by design, and all assertions and proof kits have been proven, the IP is then considered fully verified.

We selected this approach primarily because we predicted that it could achieve the same quality of verification or better as UVM, for less effort. E2E formal verification does not carry the overhead that comes with writing UVM test benches, such as the authoring of custom agents, sequence libraries, and scoreboards. Also IP specification is available before RTL is delivered, allowing DVEs to get a head start on developing the verification environment. Our experience with E2E formal verification revealed several other advantages over simulation based test benches, which will be discussed in depth later in this document.

Careful appraisal of compatibility with the methodology is critical when selecting an IP for formal verification. Generally, IPs need to be small and relatively simple. We chose to verify an Interrupt Controller (INTC) and a Crossbar (XBar) using our methodology.

Interrupt Controller

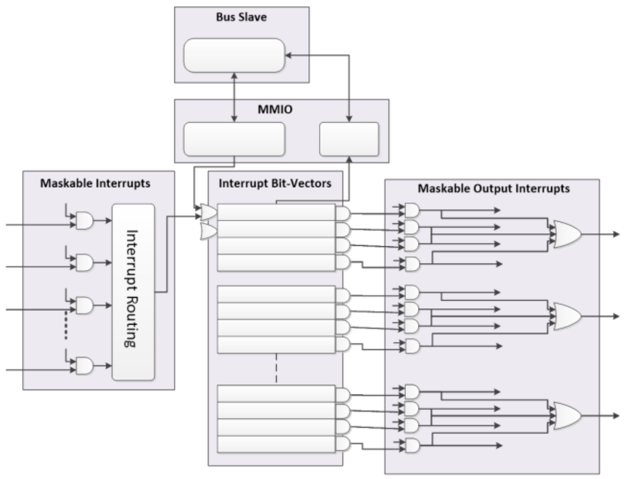

Figure 1 Interrupt Controller Architecture

The interrupt controller is responsible for routing device and SW interrupts to a dynamic location. It accepts level interrupt inputs, and routes them to the interrupt flag register and bit specified in the routing configuration registers, where that bit of the interrupt flag register is then set. Each interrupt flag register is mapped to a unique output interrupt, producing an interrupt any time the interrupt flag register is set to a nonzero value. Both input and output interrupts can be individually masked in the configuration registers. The configuration registers and interrupt flag registers can be manipulated at any time, allowing for SW controlled interrupts, dynamic routing, and interrupt masking.

Crossbar

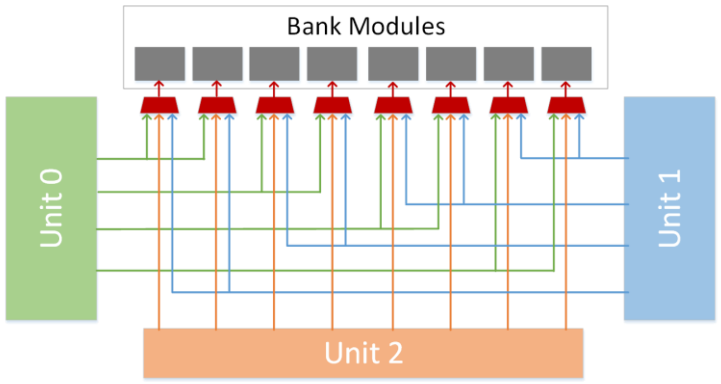

Figure 2 Crossbar Architecture

The crossbar is an arbiter that controls access to shared memory space to three different requesters. It supports round robin, fixed priority, and LFSR priority schemes. It allows a single requestor to lock exclusive access to a bank. Each bank module is responsible for the arbitration logic and contains most of the crossbar’s complexity.

Challenges

Since formal-only verification in general is difficult to accept as a replacement for constrained random verification flows, we encountered several challenges in deploying E2E formal methodology, from both technical and reporting perspectives. Some of these are described below:

1. *Quality*: For E2E formal methodology, we had to be sure that we have covered all features/aspects of the design through all the assertions that were written. At the time, the EDA tools did not have good coverage reporting (discussed below) and we had to spend additional time reviewing the assertions, compared to traditional simulation test benches. However, this was not a significant portion of the effort.

2. *Verification IP*: We had planned to write assertions for all the standard interfaces (e.g. APB, OCP etc) and configuration registers (i.e. CSRs). These would have added significant amount of effort to the assertion development. Luckily, most of the tool vendors have some 'Proof-kits' that we could leverage. These Proof-kits are analogous to VIP in simulation test benches. However, not all tool vendors have the same capability of formal proofkits. In some cases, there is additional glue logic needed to integrate the proofkit and/or additional configuration options.

3. *Convergence*: For the INTC, we chose to template the generation of the assertions due to the symmetry of the micro architecture. This resulted in 420K generated assertions for E2E formal and we quickly ran into several tool issues. These were not limited to any specific EDA vendors. There were two main underlying issues:
    a. Compile Time: It took an unreasonably large amount to time to compile the assertions even before the proof was started. Splitting the 420K assertions to smaller chunks of 10K-40K took close to several hours of compile time. We had to work closely with the AEs to debug the issues which ultimately resulted in tool R&D updates/patches that reduced the compile time to a reasonable amount.
    b. Proof Time: After resolving the compile time problems, we quickly ran into non linear run time growth vs number of assertions. Given the constraints of limited licenses & compute, there was only so far we could parallelize jobs for the INTC proof. To address the proof times even further, we had to deploy strategies as discussed below.

4. *Reporting***:** At the time of its development, the formal tools available did not provide coverage metrics that could be easily compared to simulation coverage metrics (e.g. code coverage). This warranted extra effort to develop scripts to reformat what metrics we had, into a more traditional format that would be easier for engineers, program & senior management to understand. Without a format that allowed apples-to-apples comparison with simulation reports, we were continuously challenged on verification quality and progress and had to frequently deep dive into the details of E2E formal results. In many cases, we had to work closely with the tool vendors to devise ways to post process the formal reports.

Strategies

Divide and Conquer

Certain IPs, such as the Interrupt Controller, were particularly susceptible to problems with nonlinear runtime growth with respect to assertion count. We mitigated this excessive runtime by putting a cap on the number of assertions per execution, effectively splitting the total set of assertions into smaller groups. Even when the set of capped assertion bodies were executed sequentially, total runtimes exhibited a growth pattern that was close to a linear pattern, in contrast to the nonlinear growth seen in uncapped proofs. For the Interrupt Controller, this difference amounted to a 25% reduction in runtime, with an increasing gap with higher assertion counts.

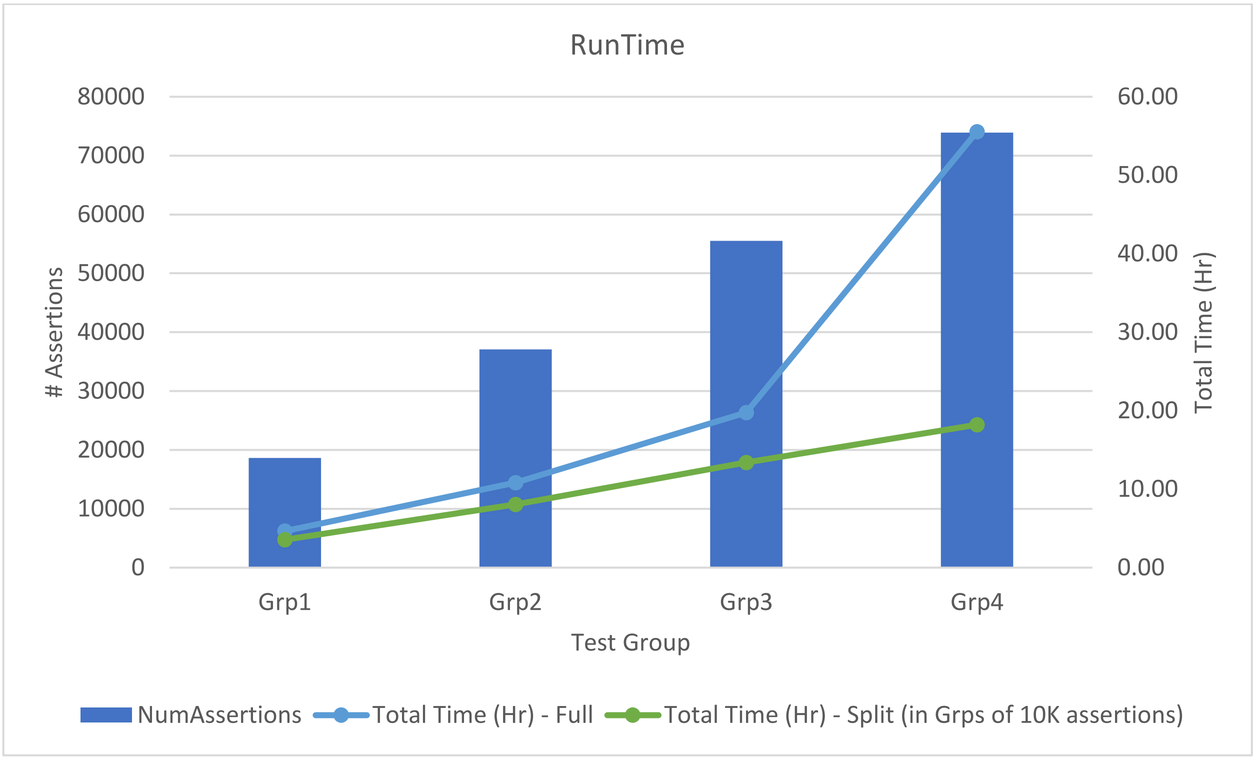

Figure 3 Runtime for Split Assertions vs Full Assertions

Hierarchical Strategy

It is common for IPs to contain multiple instantiations of the same submodule with the same parameters. When this is the case, it is efficient to verify the submodule once and verify the integration of the multiple instantiations at the top level. This reduces the number of assertions necessary to verify the IP, significantly reducing runtime, but keeps the same level of verification quality.

For example, each memory bank in the crossbar contained its own instantiation of a particular submodule that was responsible for priority logic and selection criteria. Checking this functionality is critical for the verification of the XBar, however it is unnecessary to perform the same checks across identical instantiations of the priority logic. Integration testing for these sets of submodules were done with connectivity checks at the top level.

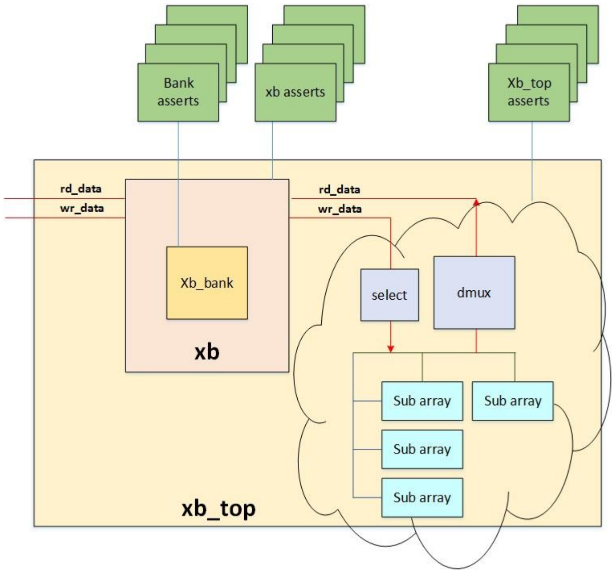

Figure 4 Illustration of Hierarchical Strategy in XBar Testbench

This approach reduces assertion complexity as well as accelerating debug time, by allowing the DVE develop, diagnose, and debug the test bench at the submodule level. It also has an advantage over UVM, in that submodule level verification requires less effort to implement using an E2E formal approach versus a simulation based approach. In UVM, it is difficult to perform submodule level verification without developing a separate test bench for the submodule. In E2E formal however, similar checks are performed by simply writing assertions for the submodule, which can be trivially imported at the top level of the test bench.

Abstractification

Abstractification is the process of identifying patterns in the IP or test bench, and describing the pattern in simpler terms via a higher level of abstraction. This shift from fine granularity to a broader granularity can result in fewer assertions, which yields performance benefits.

Abstractification had proven itself to be very useful for reducing proof time in the Interrupt Controller. This IP accepts interrupt lines which are activated by a rising edge. A naïve approach to verifying this edge detection functionality would be to write one assertion for every interrupt line. However, because system functions such as $rose() operate on single bits, this is only possible at a bit-level granularity without applying abstractification. It would be much simpler to instead test the same function across all interrupt lines, in one assertion where all interrupt lines are treated as a bit vector. This was achieved by introducing edge detector logic into the glue logic and then leveraging these structures when rewriting the assertion edge detection checks, so that they test at a word level of granularity. This shift in granularity realized a 32x reduction in assertion count for edge interrupt checks for the Interrupt Controller.

The advantage that this method gives is that it significantly reduces the assertion count without having to lose any functional verification. Even more reduction can be achieved if there are functions which do not need to be verified, such as the various possible values of the data bits fed into a mux. However, as is the case with the hierarchical strategy and blackboxing, significant caution must be practiced to ensure that no unverified functionality is being excluded.

Conventional Methods

We also leveraged some traditional formal verification strategies in our methodology. Despite the importance of exhaustive testing in E2E formal, with careful consideration, components may be safely excluded or constrained if they do not change the functionality of the IP. For example, it is unnecessary to test all possible data or address values of a bus, where only bus connectivity is being tested. These signals can be constrained to not randomize to reduce runtime. Other traditional approaches, such as utilizing parallelism and performing tool specific optimizations were also used to reduce runtime.

Results

We compared various metrics between traditional simulation (UVM) based environments vs E2E formal verification to analyze the quality and ROI on E2E formal verification methodology. In order to normalize the complexity of various IPs verified by different methods, we consider gate count as a good first order approximation for comparative analysis. We then consider the efforts required to verify each gate based on resources, schedule, testbench lines-of-code (LOC) and bugs that escaped these IP testbenches to gauge the relative efficiency & quality of each testbench. Finally, we compare the efficiency between simulation and E2E formal testbenches.

The table below highlights some key metrics for various testbenches. All data is from a recent project that has taped out.

- Multiple IPs were selected that included image processing, signal processing and general control intensive blocks
- The gate counts do not include any memory but represent the total combinatorial+sequential logic in the IP
- The duration is work days only (i.e. Mon-Fri)
- Lines-of-Code DV does not include blank or commented code in the testbench, but includes all UVM components, sequences, assertions, testcases, coverage constructs etc. This does not include 3rd party VIP code.
- Bug Escape is the number of RTL bugs that escaped IP verification and were caught at a higher level testbench after the IP RTL was frozen
- Bug Density is Total number of RTL bugs found per 1K LOC-DV
- It is assumed that licenses & compute are available and is not a factor for comparison.
- The UVM DV resources had a good mix of experience ranging from 1-10 yrs of exposure to constrained random verification and 1-7 years of exposure to UVM/OVM methodologies. The E2E formal DV resources had SVA and formal verification experience of about 1 yr.

Table 1 Testbench Metrics

TestBench	Gatecount	DV Resources	Duration (Work days)	LOC-DV	Total Person-Days	Gates verified per LOC-DV per person day	Bug Escape	Bug Density per 1K LOC
IP1 (UVM)	3170202	2	178	9795	356	0.91	1	4.39
IP2 (UVM)	381608	2	182	24725	364	0.04	0	4.45
IP3 (UVM)	1255701	3	215	33311	645	0.06	2	4.23
IP4 (UVM)	75593	1	156	12564	156	0.04	0	3.10
IP5 (UVM)	108912	1	142	6630	142	0.12	0	3.62
IP6 (UVM)	541479	1	172	25729	172	0.12	0	1.98
IP7 (UVM)	299693	1	146	12488	146	0.16	0	0.48
IP8 (UVM)	129607	1	151	8914	151	0.10	0	1.68
IP9 (UVM)	89230	1	109	7786	109	0.11	7	3.98
IP10 (UVM)	41387	1	127	5873	127	0.06	0	0.85
IP11 (UVM)	486092	1.25	147	35376	183.75	0.07	0	0.40
INTC	304174	1	87	1300	87	2.69	0	8.46
XBAR	61118	1	82	5000	82	0.15	1	3.40

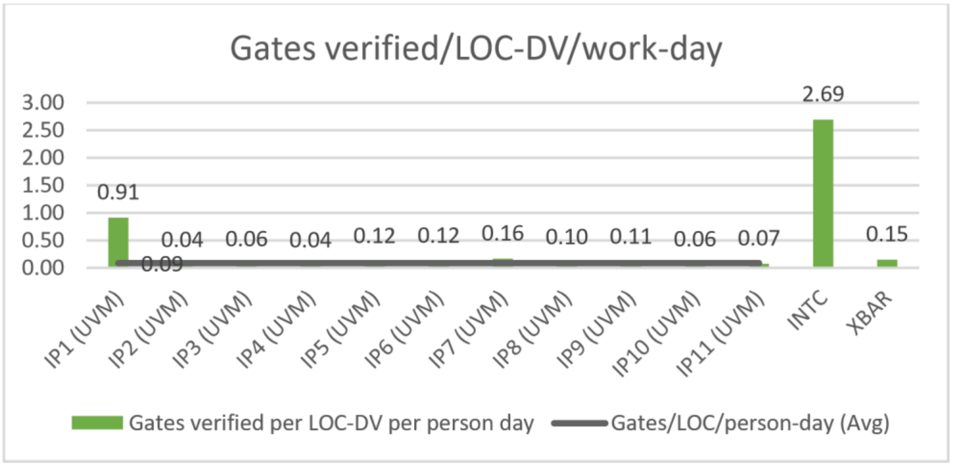

Figure 5 DV efficiency

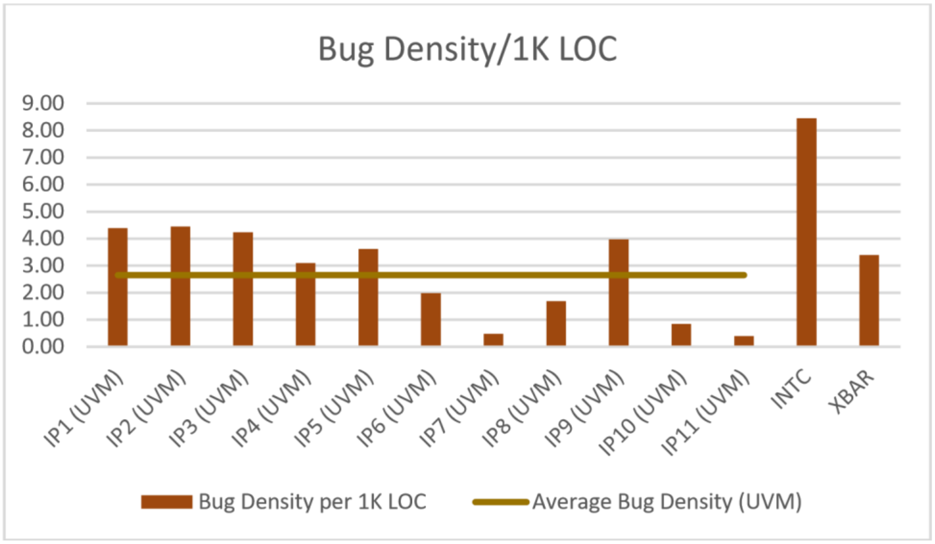

Figure 6 DV Quality

Additional comments on the data:

1. The average gates verified per LOC per work day is about 0.09, not considering IP1 which is an anomaly due to the IP micro-architecture that is heavy on usage of register files. If IP1 is considered, then the average is 0.16
2. For INTC, the total LOC-DV also includes the code/scripts needed for assertion generation using templates.
3. The higher number of bug escapes for IP9 is attributed to ownership churn where the block was owned by several different designers through the life of the IP development.

We selected three metrics for comparing quality & ROI between UVM and formal E2E methodologies:

1. Gates verified per LOC-DV per day (i.e DV Efficiency)
2. Bug density per 1K LOC-DV (i.e. DV Quality)
3. Bug escapes from IP testbench

Based on the above criteria, we draw the following inferences:

1. We can achieve the same, if not better, efficiency using formal E2E compared to simulation/UVM TB.
2. In terms of bug density, formal E2E can find more bugs per LOC-DV than UVM testbenches, primarily due to the fact that we only need assertions and minimal glue logic (if any) for the formal testbench. We also don’t need to develop any UVM/VIP components.

3. E2E formal testbenches can achieve the same quality of IP verification as UVM testbenches in terms of bug escapes
4. Even though we spent additional time reviewing the assertions in E2E formal methodology, the savings are realized through:
    a. Testbench coding: Less code required compared to UVM testbench and the testbench is ready on day 1 of RTL deliveries. First RTL bugs can be discovered as soon as the first RTL release.
    b. Coverage closure: Proofs are exhaustive and thus there is no coverage closure step
    c. Regressions: Random testing & nightly regressions are not required due to exhaustive proof
    d. Bug Reproduction: This is trivial and debug can start immediately without waiting to rerun a failing test as in simulation
5. We found that repeated patterns, symmetries, and reused sub-blocks allowed us to simplify the test bench using the strategies described earlier, saving runtime without having to compromise the quality of the verification

Overall, we find that E2E formal verification provides a compelling cost benefit against UVM while achieving at least the same, if not better, quality of verification vs simulation/UVM testbenches.

Future Work

Our experience with E2E formal verification revealed several areas that we believe have room for improvement. They are as follows:

- *Coverage Reporting*: The coverage reporting solutions available at the time of the INTC and XBar's DV cycle, were limited in functionality, and did not compare well against simulation based coverage reporting. Having coverage reporting that is easily compared against simulation based coverage reporting will make E2E formal verification more accessible from a management perspective.

- *Tool Assisted IP Appraisal:* The selection process of candidates for E2E formal verification can be enhanced with a tool that can report information about an IP, relevant to E2E formal verification fitness. Access to key data points such as maximum depth of cone of logic, duplicate submodule counts, and gate count allow the DVE to have a better understanding of the complexity of the block, and can help steer them towards sound judgment of the fitness of the IP for E2E formal verification.

- *Integrated Runtime Reductions*: Some of the aforementioned strategies can be automated, either in part or in full. Next generation of formal tools can add support for grouping assertions a la the Divide and Conquer approach, and can identify when a block is well-suited for the Hierarchical strategy. Having these features pre-integrated in existing formal tools will reduce the work load of DVEs using our methodology, accelerating time to sign-off.

- *Proof Kits:* Proof kits accelerate time to sign-off by offloading the work to verify standard interfaces, much like VIP in simulation based verification. As more proof kits become available, E2E formal verification will become accessible to more IPs.

Recommendations for Deploying E2E Formal Verification

- *IP Selection:* Not all IPs are well suited for E2E formal verification. When considering this methodology, it is critical to carefully select IPs that are compatible with the methodology. As a general rule, we recommend that any IP which can be fully specified in SVA is a good candidate for E2E formal verification. More specifically, ideal candidates for this methodology should have a shallow cone of logic, low gate count, and low complexity.

- *Enforcing Assertion Quality:* At the time of our research, existing formal tools did not offer a solution for coverage tracking similar to that of simulation based metrics. Without a coverage solution, extensive reviewing between DEs and DVEs must be done to enforce test bench quality. Reviews should check that all features have explicit checks, that checks should fail in the event of unspecified behavior, that checks should fail if any defined behavior is not preceded by explicitly defined triggers, and that all constraints do not exclude any legal and meaningful interactions with the IP. However when formal verification coverage solutions are available, we strongly recommend pairing coverage tracking with test bench reviews.

- *Aforementioned Strategies:* We also recommend employing the aforementioned strategies when deploying E2E formal verification, on IPs where those strategies apply. Use of these strategies reduces assertion counts and runtimes, without compromising the quality of the verification.

Conclusion

We find that End-To-End Formal Verification is well suited for certain types of highly symmetric and control path intensive designs. Compared to a traditional UVM based approach, our methodology is worth the savings realized in verification effort and resources, and is able to achieve at least the same quality of verification as with traditional UVM testbenches.