Maximizing Formal ROI through Accelerated IP Verification Sign-off

Hao Chen, Kamakshi Sarat Vallabhapurapu, Scott Peverelle, Rosanna Yee,
Hee Chul Kim, Johann Te, Jacob Hotz

Intel Optane Group (IOG)
Intel Corporation

***Abstract*- Over the past few years, formal verification (FV) has become an essential piece of our verification sign-off methodology. We have successfully used FV to sign off several critical design blocks with zero escapes and have also set up a mature FV sign-off flow that is well integrated into our mainstream verification process. However, despite all the great return on investment (ROI) generated on each block signed off using FV, the overall impact to previous projects was somehow limited due to the scope of FV adoption. To unleash the full power of FV, we believe that 1) FV should be considered as a primary method to achieve block-level verification sign-off and 2) FV should also be leveraged at the system level to uncover "superbugs" which are beyond the reach of the traditional simulation approach. This idea is supported by our management team. As a result, we started to deploy FV on a much larger scale to accelerate the verification sign-off of a brand-new IP. This paper shares our recent experience on how we upscale FV usage to maximize formal ROI.**

I. INTRODUCTION

Formal Verification (FV) has been recognized as a mainstream verification sign-off methodology by our team for several years. To ensure sign-off quality, we have developed a mature FV methodology [1] [2] which seamlessly integrates with our simulation environment and coverage metrics. Meanwhile, most of our ASIC design and verification engineers were trained based on our FV methodology to enable adoption. As a result, several different IP teams started to use FV to verify suitable blocks within their IPs. For all the blocks verified by FV, no bug escapes were found after tape-in. However, for the past several projects, the overall FV adoption rate was still relatively low, thus the ROI impact by FV was limited. A survey was then performed to help understand the low adoption rate. Among all the obstacles perceived by our engineers, the biggest challenge was the belief that formal cannot handle large designs due to state space explosion. This has limited our team to only apply FV on small, targeted areas and use simulation to verify most of the logic within our SoCs.

To address these challenges and expand formal usage to the next level, we have done several proofs of concept (POC) projects to reveal FV's capabilities and improved our FV sign-off methodology based on learnings from the industry. With that, we convinced our management to support applying FV on a brand-new design IP to accelerate verification sign-off. This IP is a critical subsystem of our SoC with complex 3rd party IP integrated. Our strategy is to use FV to verify 10+ new design blocks as well as targeted features at the IP level. Our current results demonstrate significant ROI in terms of IP quality and time to market.

This paper describes how FV is used to accelerate IP development with collected ROI results. The rest of this paper is organized as follows. Section II of the paper explains our overall verification strategy for the targeted IP subsystem. Section III explains our latest FV methodology that enables us to unleash the full potential of FV to accelerate and improve IP verification. Section IV presents our current results of using the proposed methodology. Section V summarizes our key accomplishments and discusses some future work.

II. PROPOSED HYBRID IP VERIFICATION STRATEGY

Traditionally our team used a monolithic simulation-based verification strategy to verify ASIC IPs at two to three levels: block, cluster (if target IP is large and complex) and full IP. For this new IP, as shown in Figure 1, we decided to use a hybrid verification strategy where most of the block-level verification is done through FV, and simulation is used at the full IP level. As most blocks inside this IP are control-oriented or for data transportation, they are a perfect fit for FV. Since this IP is relatively large, we have divided the IP verification into two different cluster-level environments, with one verified through FV and the other one verified through simulation. This hybrid strategy allowed us to take full advantage of FV to exhaustively verify all corner-case scenarios for block-level features and critical interactions between blocks including 3rd party IPs. Meanwhile, it also provided us opportunities to optimize our execution processes to shift the development cycle left.

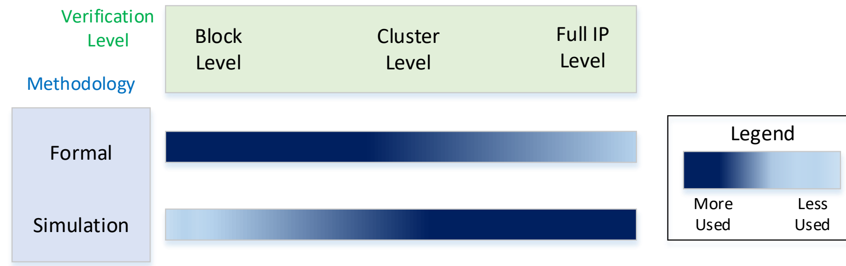

Figure 1 Proposed Hybrid Verification Strategy to Maximize Formal ROI

To help understand the benefits of this hybrid verification strategy, let us first review our typical ASIC product development cycle. As shown in Figure 2, there are in general three major design phases: ramp, develop and support. Throughout the design cycle, there are 5 major milestones:

- Design Kick-off (DK): design enters active development phase.
- Intermediate Milestone 1 (IM1): initial design delivery suitable for floor plan, I/O placement, and package development.
- Intermediate Milestone 2 (IM2): second design delivery ready for first timing closure.
- Tape-in (TI): final design delivery ready for backend implementation.
- Tape-out (TO): physical design shipped to a foundry.

RTL design starts from DK and will finish around IM2. Verification also starts from DK and it is required to finish by tape-in with all planned coverage closed, often consisting of two phases: phase 1 focuses on testing of normal data and control flows while phase 2 focuses on exceptions and error cases. After tape-in, verification will continue for bug hunting. In case any late bugs are found, design ECOs will be performed before tape-out. One major challenge is how to find design bugs, especially critical bugs, early in the design cycle as it becomes much more expensive to fix if found closer towards tape-in compared to when it is found during the early RTL development phase. Also, because of the long period of the ASIC development cycle, it is very common to receive late RTL change requests towards the end of the project. These are the two major reasons that cause schedule slip thus delaying time to market.

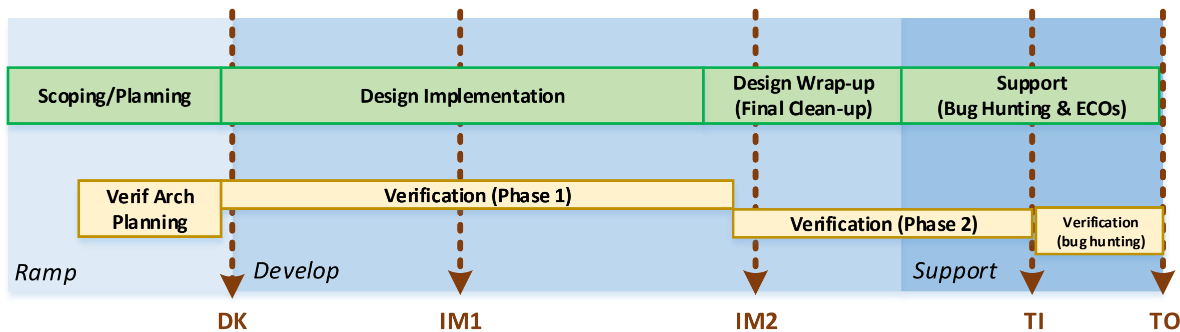

Figure 2 A Typical ASIC Product Development Cycle

To deliver high-quality design IP on time, we chose to implement this hybrid verification strategy. When formal is deployed at large scales on the block/cluster level, it can provide us great strategic benefits: First, by planning formal on all blocks and their interfaces, we get a chance to thoroughly refine the design specification to prevent specification issues to propagate into RTL coding. Second, by leveraging key formal advantages such as exhaustive stimulus and simpler testbenches, we can find most of the RTL bugs much earlier in the design cycle (ideally before IM2). In contrast, it usually takes much longer to get proper stimulus manually coded to uncover corner-case bugs in simulation, especially error cases which are unlikely to be exercised before IM2. Third, we can build highly abstracted formal models for design blocks to enable architectural formal proof before IM1. This will help us detect and fix critical bugs such as architectural bugs or interaction bugs with 3rd party IP in the early stages of development, reducing cost and preventing code churns that impact project schedule.

Figure 3 shows our predicted shift left using the proposed hybrid strategy. We hypothesize that if we apply formal systematically and widely on IP development, we will deliver high-quality design with a predictable schedule.

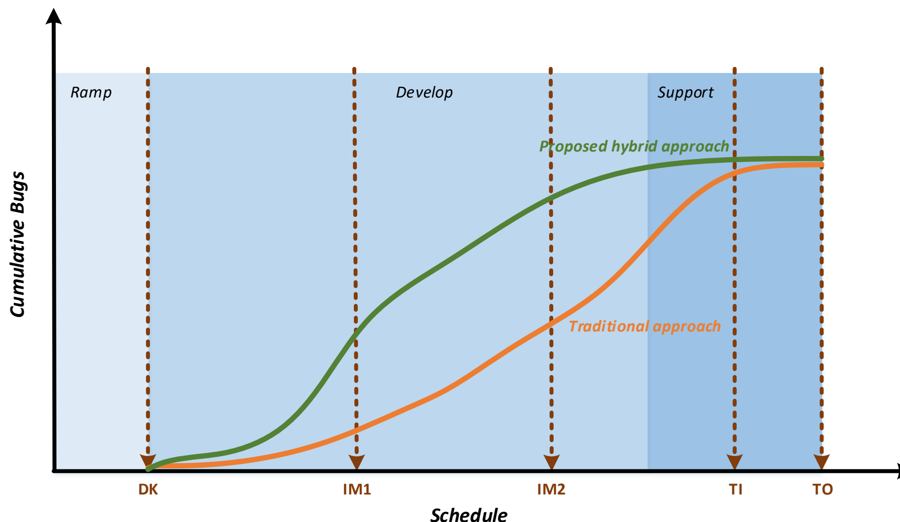

Figure 3 Predicted shift left using proposed verification strategy

While this hybrid verification strategy enables us to take full advantage of FV to find RTL bugs earlier to shift left in the development cycle, it also faces several new challenges. Among those, the most significant one is how to ramp up a dedicated formal team to execute the proposed strategy while ensuring signoff quality. Since our verification team is in general more experienced with constraint-random simulation methodology, most of the team members had no previous FV experience. To overcome this challenge, we had experienced FV engineers to help define the formal testbench architecture and closely mentor others who are new to formal. More importantly, we required everyone on the formal team to follow the standard methodology to ensure consistency and quality of work. More details of our FV methodology will be discussed in the following section. On the other hand, we also got great support from our design team. For example, our micro-architect created partitioned requirements at block level to meet end-to-end ordering rules. Our design team provided finer design partitions which allowed us to reduce the DUT scope for new FV engineers to manage complexity. All CDC crossings within the IP are grouped into a dedicated block to help reduce verification effort on metastability testing. All these great design-for-verification considerations had helped us to achieve our goals.

III. Formal Methodology to Ensure Rapid and High-Quality Sign-off

A. *Formal Property-Driven RTL Development*

Traditionally, ASIC designers need to first write the RTL, then perform some sanity testing before handing off RTL code to verification. In the meantime, verification engineers also need substantial time to bring up the testbench to exercise the design in simulation. Therefore, typically most RTL bugs are only detected after IM1. With formal, we are now able to uncover all possible corner-case bugs as soon as the RTL is written. Inspired by the test-driven development concept (TDD) [3], we deploy formal in parallel with RTL development. This allows us to test new design RTL features by feature without waiting for it to be fully developed. As a result, we can find a large percentage of bugs before IM1 and most of the design bugs before IM2.

Table 1 summarizes our formal property-driven RTL bring-up and exploration flow. Compared to the traditional RTL bring-up & exploration process, this flow has some major advantages:

- Overall, this flow enables valuable parallel development between design and verification and allows our design and verification engineers to collaborate more closely during the initial RTL development phase.
- From a design perspective, we eliminate the need for our designers to create separate sanity test environments. Because such environments are generally basic which can only cover limited design state space, and they are never reused in any downstream verification platforms. Instead, we believe it is a better usage of our designer’s time to start design exploration and functional debug directly using FV testbenches provided by the verification team. In contrast, these FV testbenches can explore the entire design state space.

Designers are also highly encouraged to write embedded assertions in their RTL and prove them in the FV testbench.

- From the verification perspective, study [4] has shown that on average verification engineers spend more of their time debugging (>40%) than any other activity. This can be a significant challenge when planning the schedule since debugging is unpredictable and varies significantly from case to case. Therefore, having designers co-own the RTL bring-up and exploration tasks in the same FV environments can help reduce the overall debug effort required from the verification team, thus reducing unnecessary delays of the project.

TABLE 1

FORMAL PROPERTY-DRIVEN RTL BRING-UP AND EXPLORATION

Flow Step	Performer	Description
FV Planning	Verification	This is a prerequisite step before RTL bring-up and exploration When this step is done: • Micro-architecture discussions complete and verification feedbacks have been taken on design spec. • Verification plan and testbench architecture reviewed by design.
FV Initial Setup with RTL stub	Design & Verification	At this step: • Designer works on RTL stubs. These stubs only contain interface definitions and auto-generated register blocks. Block-specific features are not implemented at this stage. Therefore, most outputs are tied off except the register path. The main goal of having RTL stubs is to facilitate parallel development of the verification testbenches. • Verification engineer creates initial FV testbench with RTL stubs integrated. Initial testbench contains mostly interface properties and global constraints. When this step is done: • RTL stubs complete. • Initial testbench complete and elaboration is clean. Register access testing can be performed at this stage.
FV Bring-up with baseline RTL	Design & Verification	At this step: • Designer implements required features in the design block. Meanwhile, RTL bring-up takes place in the initial FV environment. The focus is to ensure desired behaviors happen in the design. • Verification engineer starts to add assertions based on design spec. Debug initial assertion failures to rule out testbench issues. When this step is done: • Baseline RTL complete. This means main design functions are implemented and the end-to-end path is working • Desired behaviors witnessed using covers in the FV environment. It is very important to have sanity covers witnessed first to ensure the FV environment is healthy before analyzing any assertion failures [1]. • Testbench upgraded with assertions for main design functions. No assertion failures.
FV Exploration with feature-complete RTL	Design & Verification	At this step: • Designer finishes RTL with embedded assertions. Design exercise starts to debug and clean up any assertion failures with verification support. • Verification engineer completes all required assertions and starts to implement coverage model in the FV testbench. The focus is to exercise all assertions and debug

		RTL/spec/testbench issues with designer When this step is done: • Feature complete RTL delivered • FV testbench implements all required assertions. • Debug and fix any assertion failures. Assertion bounds are not analyzed at this step.
FV Sign-off	Verification	Follow-on step to achieve verification sign-off. See the next sub-section for more details.

Once the FV exploration step completes, we would expect the design to be very clean and the block to be ready for sign-off.

B. Formal Sign-off Flow

The goal of verification sign-off is to provide confidence that the design under test (DUT) has been fully verified in a deterministic and measurable way. For formal, we need to ensure: 1) the FV testbench does not contain any over-constraints to disallow legal stimulus. 2) we should have the complete set of assertions to cover all the design features. 3) in case some assertions cannot be fully proven, what are the sufficient assertion bounds for safe sign-off [5]? All the above questions can be answered by formal coverage analysis. First, we leverage the automated code coverage metrics including stimuli reachability and assertion COI/proof core coverage. Meanwhile, we also need to define complete functional coverage based on the design specification. In the end, we shall reach 100% coverage before sign-off. For more information about our coverage-driven sign-off methodology, please refer to [1].

Besides the sign-off flow, we have also defined a consistent FV testbench architecture that enables standardization and reuse. The full details of our testbench architecture can be found in [2]. To summarize, for every formal testbench, we will create a formal verification component (FVC) with reusable elements and a formal environment (ENV) that contains additional block environment-specific or formal specific logic. The idea is to separate reusable elements from non-reusable ones. All major testbench components are inside an FVC which is a layered, synthesizable, and configurable SystemVerilog module. As shown in Figure 4, an FVC consists of a few interface FBMs (Formal Bus Model), end-to-end constraints, a checking model, and a coverage model. The FVC has a mode parameter that can be configured into *ACTIVE*, *PASSIVE,* or *ARCH_FPV* mode. When it is in *ACTIVE* mode, all assertions and covers are enabled to run proofs in the formal tools. whereas in *PASSIVE* or *ARCH_FPV* mode, only a subset of reusable properties is enabled.

Formal Bus Models (FBM)

FBMs are self-contained and reusable Assertion Based Verification IPs (ABVIPs) for interface protocol modeling and checking. Assertions are implemented based on interface design requirements and they are bi-directional depending on individual signal's directions. This enables assume-guarantee proofs at the interfaces between neighboring blocks in FV. FBMs also provide reusable interface coverage.

End-to-End Constraints

Besides interface protocol constraints provided by FBMs, sometimes a design may also need end-to-end constraints for design behaviors involving multiple interfaces.

Checking Models

The checking model is the core piece of FVC. It contains end-to-end and white-box checkers that verify design functionalities. Formal scoreboards are often used to check data integrity.

Coverage Models

The coverage model is another important part of FVC. It is used to measure verification completeness. In the coverage model, functional cover points are implemented using covergroups or cover properties to observe exercised design activities.

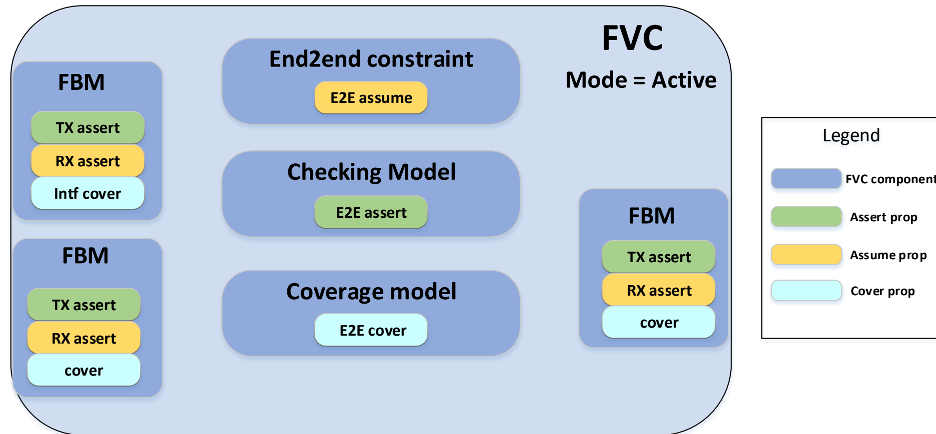

Figure 4 A Formal Verification Component (in Active Model)

In addition, there are a few key considerations that helped us to apply FV to 10+ design blocks efficiently. First, it is essential to leverage the assume-guarantee method to cross prove interface constraints' assumptions. For example, the assumptions on a block's inputs are converted into assertions on the neighboring block's same output signals to guarantee/check the desired behaviors. Previously we were only able to validate them in the upper-level simulation environment. For this new IP, because we build FV environments for every design block, we can verify the interface handshakes formally. In fact, by using the interface assertions in the FBMs, we were able to find many RTL bugs in the early bring-up stage. Second, whenever possible, we try to create reusable FVCs on the sub-block level to verify common functionality across blocks. To do this, we must carefully examine the design's micro-architecture to divide certain end-to-end checks into several independent parts. Finally, when proving end-to-end checkers on large-scale design blocks or clusters, it is very common to run into state-space explosion problems. To overcome the complexity challenge, we consider two formal techniques: First, semi-formal bug hunting can be leveraged at an early design phase to search for bugs. The idea here is that non-exhaustive formal engines can help us explore deep design state space without waiting for any design abstractions/reductions. As soon as we have enough assertions and they are free of failures from normal formal proof, we can kick off bug hunting to search for deeper bugs. It is an essential step to shift left the process to find RTL bugs. Later, towards the end of the project, it is also important to run bug hunting on assertions that cannot be fully proven to gain additional confidence. There are many different hunt strategies available in the formal tools, but it is beyond the scope of this paper. Second, architectural formal verification methodology can also be used to provide exhaustive analysis. More details can be found in the next sub-section.

C. Architectural Formal Verification

Architectural formal property verification (FPV) [6] is an extremely powerful technique to ensure exhaustive verification of IP-level requirements. It helps to overcome complex barriers to enable FV on much larger scale designs. As shown in Figure 5, there are 3 steps in the Arch FPV flow:

- Step 1: create an architectural model (AM) for each block. These AMs are supposed to only model the necessary functionality contributing to the architectural requirement. Therefore, they are much smaller than the actual RTL implementation.
- Step 2: create an architectural FPV test environment to verify end-to-end requirements using only the AMs. No RTL is needed in this environment.
- Step 3: to ensure RTL implementation matches the assumptions made in the AMs, we need to prove the block-level assumptions against the RTL.

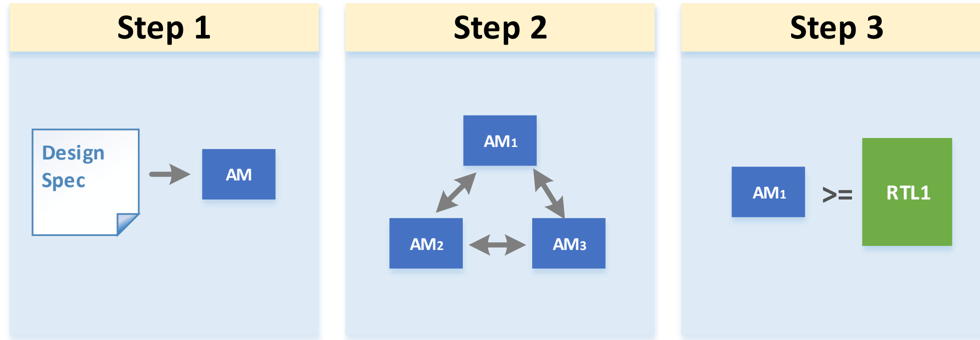

Figure 5 Architectural FPV Flow [6]

In theory, this flow can be used by our micro-architects to exhaustively explore design architecture during the project planning phase without any RTL implementation. However, it requires a significant amount of effort as well as formal expertise to build this environment. For our new IP, we chose to have the formal verification team build the architectural modeling capability into our block-level FVCs. This helps us to verify IP-level requirements following the architectural FPV flow. Once the model is built, we can further leverage it for architectural exploration in future projects. Currently, we use this approach to verify IP requirements such as command ordering rules, absence of deadlock, and system latency. Meanwhile, certain requirements such as system throughput cannot be tested using the FV environment. They are covered in the IP-level simulation environment.

D. Formal Reuse Methodology

A systematic reuse methodology is important to ensure verification quality and efficiency. In the past, we have had the experience of integrating FVCs into simulation and emulation platforms [2]. For this new IP, because we also use FV to verify one design cluster, it allows us to reuse FVCs in the FV environment as well. In general, our FVC architecture provides a clean interface to make the integration process simple and easy. Because each FVC uses the identical ports of the DUT, we just need to bind the FVC to its DUT and set the mode to *PASSIVE*. As shown in Figure 6, when an FVC is used in passive mode, we convert all assumptions to assertions at the cluster level. It is very important to validate all block-level constraints in a higher-level environment. Occasionally, over-constraints escape block-level coverage analysis but will be exposed at the IP or SoC level. On the other hand, it is not necessary to reuse all assertions and covers in an FVC when they are fully exercised at the block level. For example, we usually choose to only propagate important end-to-end and/or bounded assertions in passive mode. Given all the FV constraints are thoroughly validated in both FV and simulation environments, we are very confident that the fully proven assertions achieve exhaustive results. Another major reason is to reduce reuse overhead. A typical block-level FVC may contain tens of thousands of covers and assertions. Reusing all of them can significantly impact simulation/emulation performance.

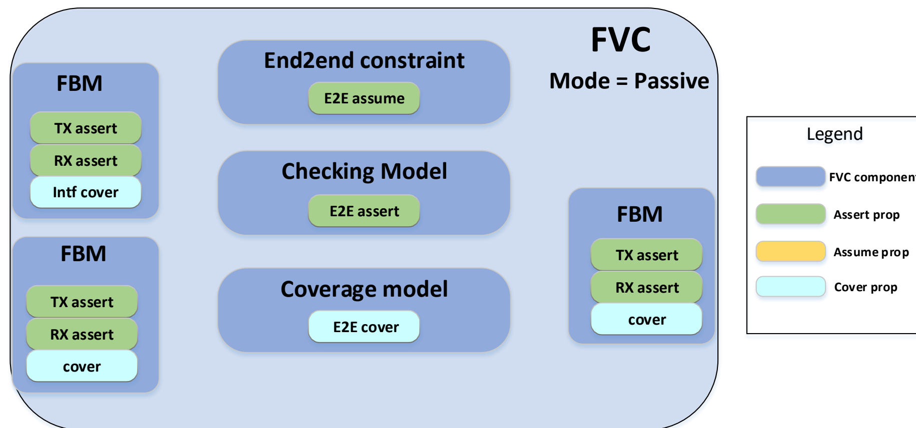

Figure 6 A Formal Verification Component (in Passive Model)

An FVC can also be configured in *ARCH_FPV* mode. This enables only a subset of functionality required to perform inter-block verification. For example, data integrity scoreboards are disabled in *ARCH_FPV* mode since our

architectural FPV proof is focused on deadlock detection. As shown in Figure 7, both end-to-end and interface assertions are converted to assumptions to define legal behaviors on the block's output signals. Meanwhile, original interface assumptions are not needed thus they are disabled.

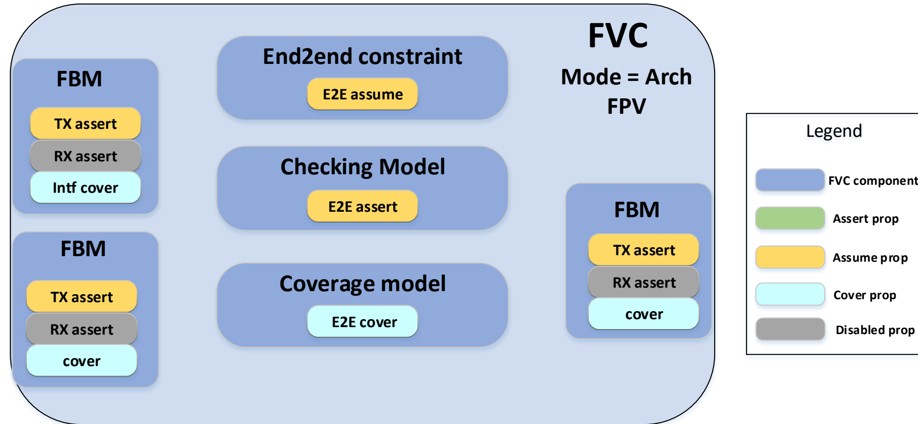

Figure 7 A Formal Verification Component (in Arch FPV Model)

E. *Leveraging Formal Apps*

To completely sign off a design block, we also rely on specialized formal apps to verify suitable design requirements. The major advantage of formal apps is the automation provided to reduce verification effort. Most of the formal apps only require users to provide RTL without a testbench.

Figure 8 illustrates a typical design block with its essential parts. Besides the core logic which implements block specific functionality, many standard parts can be verified using formal apps:

- Dynamic clock gating logic can be verified using a sequential equivalence checking (SEC) app
- Register block can be verified using a specialized CSR app
- Connectivity app can be used to verify inter-block connections
- A specialized XPROP app can be used to perform an X-propagation check on the design's outputs
- An CDC app can be used for metastability-aware functional verification. Different from other automatic formal apps mentioned above, we need to first build our FV testbench with user-defined assertions, then use the CDC app to inject metastability exhaustively on all CDC crossings to analyze its functional impact. We had a successful experience using this approach to find unique metastability-related issues in our design.

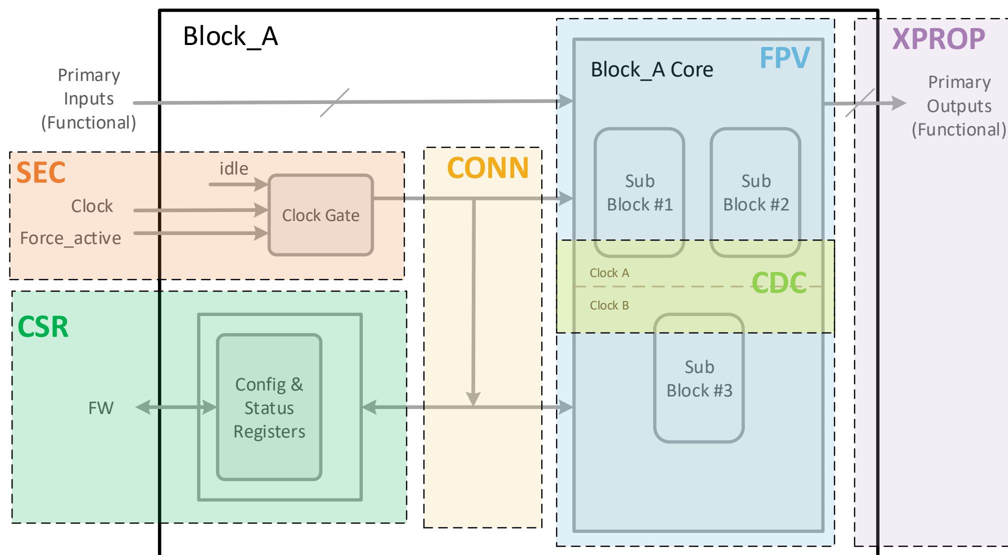

Figure 8 Using Formal Apps to Completely Verify A Design

IV. RESULTS

Developing a new design IP with over 20M gates from the ground up is very time-consuming and challenging. Therefore, to make a major impact, we would like to leverage FV to achieve two major goals: First, FV should be used in the early development phase on all new RTL to catch as many bugs as possible. Second, FV should also be used to verify every high-risk feature at the cluster or IP level to reduce the risk of critical bug escapes. Our current results show excellent ROI in both aspects.

1) Design RTL exploration shift-left
Since we employed the formal property-driven RTL development flow, we were able to start RTL bring-up and exploration as soon as RTL was ready. In some cases, we even built the FV tests before the RTL was coded. As a result, we expedited our debug process which produced high-quality block-level RTL at a very early stage.

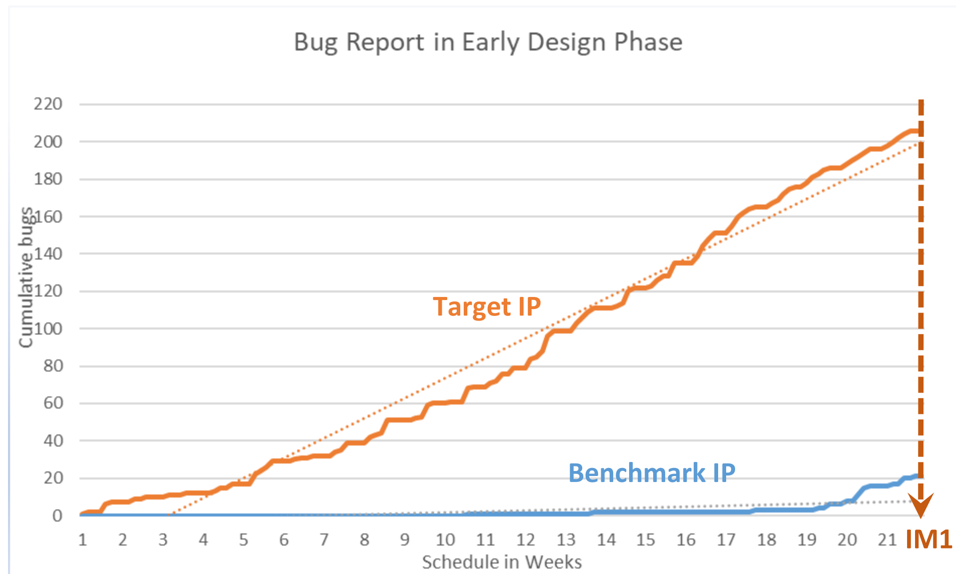

Figure 9 RTL Debug Shift-left

Figure 9 demonstrates the timeline of design bugs found before the IM1 milestone. In this chart, we compare the bug-finding rate of our target IP with another comparable IP. The comparable IP is chosen because it was also a brand-new IP in that previous project, it has similar design complexity to our target new IP, and the allocated verification resources are roughly the same. The only major difference is that the benchmark IP was completely verified with dynamic simulation, from block to IP level. As shown in Figure 9, by the time of IM1, only 21 RTL bugs were found in the benchmark IP whereas we found over 200 in the new target IP. It took us several more months to find the same number of bugs in the benchmark design, at that time it was already close to IM2. Of those 200 bugs, most of them were found by our block-level FV environments, including corner-case bugs which typically can only be found at a much later stage.

2) High quality sign-off
Besides RTL exploration shift-left, we also tried to verify critical design requirements at an early stage. Some of these requirements are at the IP level. Figure 10 shows the block diagram of the IP cluster verified in FV. It is a highly concurrent system interfacing between two different protocols. One major goal is to prove the absence of a deadlock in this system. The original design RTL contains hundreds of thousands of flops inside the cone of influence (COI). Running FV proof directly on the design will likely run into a state-space explosion. To overcome the complexity challenge, we followed the architectural FPV flow to create architectural models for each block with highly abstracted logic. As a result, we were able to reduce the COI size by over 15x! With the model, we were able to prove the absence of a deadlock considering error cases in the IM1 timeframe.

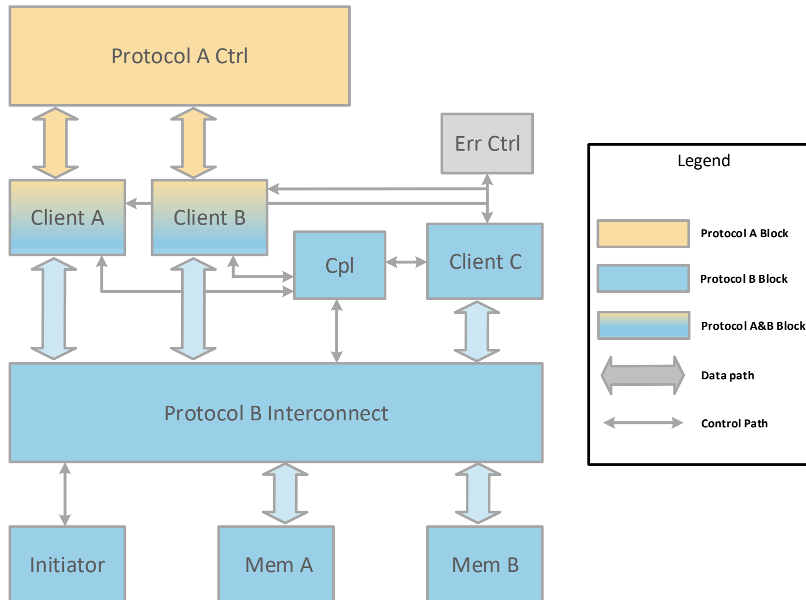

Figure 10 IP Cluster Verified in FPV

Another common source of bug escape comes from 3rd party IPs. Although 3rd party IPs are generally pre-verified, when we plug them into our IP and have our logic interacting with them, we often detect bugs at the integration level. With the traditional approach, we usually need to wait for a long time before simulation tests become available to verify 3rd party IP interactions. If these bugs are found late in the design cycle, we have no control over when they will get fixed. This often results in costly delays in schedule. With FV testbench built on the IP or cluster level, we can enable such tests early on. For example, we tried to formally verify cycle-accurate interface handshake behaviors using FBMs (Commercial Assertion-based VIPs are used for standard interfaces). This effort helped us clarify ambiguities in the 3rd party IP design specification. It also helped us find corner-case bugs violating the handshake protocol. Meanwhile, we also used end-to-end checkers to ensure the selected 3rd party IPs fulfill our design requirements. We found a few interesting bugs due to misinterpretation of the 3rd party IP's output behaviors. Overall, we were able to verify several high-priority requirements against 3rd party IP before the IM1 milestone in a couple of weeks. This provided us with greater confidence in our IP quality and helped us ensure a predictable schedule.

3) Team growth
Besides systematic methodology, another key factor to enable wide formal adoption is the team. Unlike simulation methodology such as UVM, FV is still a completely new field for many of our verification engineers. This was another limiting factor that prevented us from moving from the monolithic simulation-based verification strategy to more efficient hybrid strategies. However, according to our survey, over 90% of engineers who participated believe formal is useful and are willing to learn formal. With the management team's support, we were able to increase formal usage in this new IP. In this project, we built a formal team of seven members (two part-time members working on simulation in parallel). Among them, we only had two experienced FV engineers, and the rest of the team was learning formal from the beginning. With proper planning and support, our new FV engineers were able to successfully execute the assigned tasks. Overall, this project provided great learning opportunities for our team members, both designers and verification engineers. It is essential to keep our team competitive to deliver high-quality products given aggressive development schedules.

V. Conclusions and Future Work

FV is an extremely powerful methodology to help deliver high-quality products in a predictable way. When used properly, it provides a great competitive advantage. In the past, we used FV to sign off targeted blocks in several projects. Results indicated great ROI in quality of verification and productivity. However, due to the limited scope of application, the overall project impact was still relatively low. Two major challenges were observed as obstacles to wider FV adoption: 1) lack of confidence in FV to handle large/complex designs, 2) lack of FV expertise in our team.

To overcome these obstacles, our FV expert team had conducted several path-finding projects to improve our methodology, with a focus on complexity analysis and reduction. One great technique we learned is the architectural FPV flow. It helps to divide and conquer the design complexity barriers to verify IP-level requirements. Meanwhile, the expert team also conducted internal FV training to improve our team's formal expertise. Overall, we provided this training to over 80 design and verification engineers. All the preparation work made us confident to upscale formal usage to increase ROI.

With the management team's support, we started to deploy FV on a much larger scale to accelerate the verification sign-off of a brand-new IP. As predicted, with FV applied on most of the design blocks and one cluster, we were able to find most of the RTL bugs much earlier and thoroughly verify critical IP-level requirements at a very early design stage. Overall results demonstrate significant ROI in terms of IP quality and time to market. Moreover, this project helped our team to gain valuable hands-on FV experience. It enables us to further expand formal usage in the future.

ACKNOWLEDGMENT

We would like to first thank our IP manager, Jason K. Tan, for his full support to enable this work. We would also like to thank our micro-architect, Scott Nelson, and the entire design and verification team for their great collaboration during the development cycle.

REFERENCES

[1] A. Li, H. Chen, J. K. Yu, E.L. Teoh, I. P. Anand, "A Coverage-Driven Formal Methodology for Verification Sign-off", DVCon 2019

[2] H. Chen, Y. Sun, A. Li, D. Cao, "A Systematic Formal Reuse Methodology: From Blocks to SoC Systems", DVCon 2020

[3] "What is Test Driven Development (TDD)?", Agile Alliance

[4] Harry D. Foster, "2020 Wilson Research Group Functional Verification Study", Siemens EDA

[5] N. Kim, J. Park, H. Singh, V. Singhal, "Sign-off with Bounded Formal Verification Proofs", DVCon 2014.

[6] M. Munishwar, N. Zaman, A. Jain, H. Singh, V. Singhal, "Architectural Formal Verification of System-Level Deadlocks ", DVCon 2018

[7] M A. Kiran Kumar, E. Seligman, A. Gupta, S. Bindumadhava, A. Bharadwaj, "Making Formal Property Verification Mainstream: An Intel Graphics Experience", DVCon 2017.

[8] J. R. Maas, N. Regmi, A. Kulkarni, K. Palaniswami, "End to End Formal Verification Strategies for IP Verification", DVCon 2017.