Making Formal Property Verification Mainstream: An Intel® Graphics Experience

M Achutha KiranKumar V[1] (achutha.kirankumar.v.m@intel.com)
Erik Seligman[2] (erik.seligman@intel.com)
Aarti Gupta[1] (aarti.gupta@intel.com)
Ss Bindumadhava[1] (bindumadhava.ss@intel.com)
Abhijith Bharadwaj[1] (abhijith.bharadwaj@intel.com)
[1]L3167, BGA, Intel Technologies Ind Pvt Ltd, Old Airport Road, Bangalore, India. +91-80-25076816
[2] RA4-4-J6, Ronler Acres, Intel Technologies, Portland, USA +1-503-613-7041

***Abstract*- Formal Verification (FV) has been widely accepted as a powerful approach to catch corner case issues and to reduce design risks [1]. Despite the recognized advantages, FV, being conceptually different from the traditional simulation approaches, has often been limited to central FV teams. Widespread formal deployment is challenged by the perceived methodology complexity and presumed high effort. However, Formal Property verification (FPV) EDA tools have made major leaps in usability over the past decade, and now can be easily deployed to non-FV-specialist design engineers [2]. In addition to the usability questions, articulating the Return on the Investments (ROI) of FV efforts has tended to be difficult. Therefore, integration of FV into mainstream activities needs tracking methodologies and metrics which are also analogous to existing verification practice. This paper regales the story of successful FV deployment carried out in the graphics design team at Intel, while streamlining the processes and debunking common myths. 85 engineers were rigorously trained on FPV in a span of two weeks and were supported with onsite consultancy for another 4 weeks. Despite starting after a major Dynamic Validation (DV) milestone completion, the deployment succeeded in finding 82 interesting bugs in a span of 12 weeks that accounted for a weighted ROI [4] of 220x against similarly supported simulation efforts. The results achieved in this short span have galvanized many designers to adopt FPV as a Plan of Record (PoR) for succeeding projects.**

Keywords- formal verification; formal ROI, formal coverage

I. INTRODUCTION

Formal Verification needs no introduction, as it is a well-established technique known to many designers. Its power to exhaustively and quickly verify the design-under-test (DUT) gives it a definite edge over the existing simulation-based verification counterparts. With this advantage, one would assume that FV could be the tool of choice for all DUTs where it can be applied. Quite the contrary, when a close analysis was done on FV usage patterns at Intel, this assumption was refuted. It was found that instead of the expected widespread FV usage, FV experts work in small targeted areas and are able to cover only a small portion of the FV-able design space. This is a matter of concern, as the potential of FV is not getting fully tapped to give better and faster verification confidence on the DUT. On further exploration w.r.t. the gap between FV adoption expectation and usage, some of the general arguments were:

1. **Fear.** FV is commonly considered an esoteric methodology by the designers. A naïve user is apprehensive about learning and implementing it.
2. **Progress metrics.** Most of the FV engineers are not conversant with the FV progress tracking methodology to report to the management, which is well-accustomed to see standard progress reports by the simulation based verification flows.
3. **ROI.** FV users often struggle to translate their efforts to ROI in various non-standard reporting forms. If these efforts are not appropriately understood and acknowledged, management will not see the effort as productive.

In this paper, we bridge the FV adoption gap by addressing the above problems. To address the first issue, fear, we made a strong effort to lower the entry barrier. This prompted us to create a Graphical User Interface (GUI) that automated environment setup, enable effective training, provide self-help materials with detailed walk-through examples, and create cookie-cutter assertion libraries.

The next two challenges, reporting progress and understanding the ROI, were closely related. Formal verification progress communication always used to be a step function, either it was not done or completed. Communicating random progress wouldn't be appreciated by the management unless aligned with the project progress metrics. In order to effectually articulate FV progress and ROI to management, a new methodology that standardizes the planning, tracking, reporting and assessing the FV efforts was deployed. This methodology helps users to communicate FV progress and ROI more lucidly as compared to the previous modus operandi where stakeholders got to be notified only at the start and completion of the task.

Apart from resolving these prominent challenges, certain steps made the FV adoption process even smoother. The success of any FPV activity has a major dependence on the designer's involvement and manager's support. FV motivational sessions were worked with the designers. While working through their tight schedules, handling the FPV activity in parallel often seems to be a clear overload to the design engineers. Methodology enhancements to ease the usage and ROI discussions interested the designers. Most importantly, the designers needed to clearly understand that by putting in some extra effort at the design stage, they were significantly pulling in bug finding and helping to quicken the overall project timeline.

With proper motivation and realistic figures, the management joined the bandwagon of promoting FV on the designs. It was the responsibility of the central FV team to convince the graphics team management and garner their support, by the results of early pioneering. Nevertheless, unless FV is integrated into mainstream deliverables, it would not be exercised, despite the benefits. A set of strategic check-points analogous and incidental to the DV milestones were introduced in the methodology, which convinced the management. The progress of the FV task completion is articulated starting from the test-planning phase, enabling a continuous tie to the project timelines. Project milestone completions were made dependent on FV task completions, thus ensuring FV momentum and process streamlining.

This deployment expedition is described in detail in Section II. The results of this exercise are discussed in Section III. The demonstrated results warranted the continuation of this methodology on future Intel Graphics projects. Authors believe that the proposed methodology can be incorporated in any ASIC verification flow to ensure effective usage of FV, and thus better verification confidence.

II. FV MAINSTREAM DEPLOYMENT DRIVE

The mission of integrating FV in the mainstream verification flow was accomplished in a phased manner. Each phase was targeted to tackle a set of FV adoption obstacles. Figure 1 provides a short summary of the different phases.

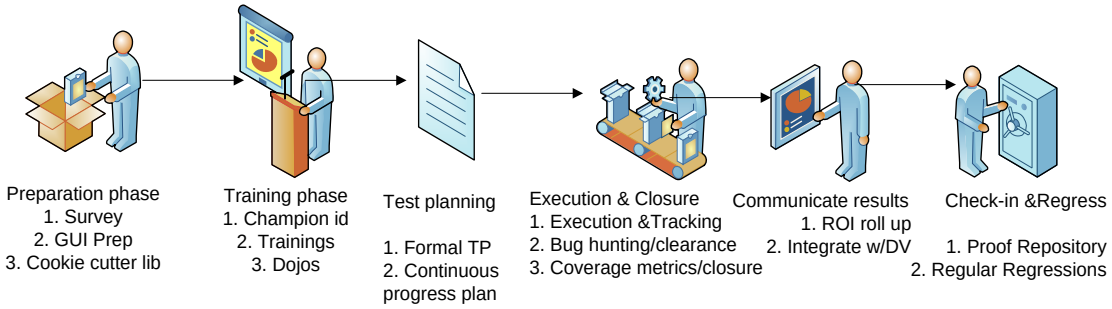

Figure 1: Deployment Methodology

A. Preparation Phase

This phase of deployment concentrated primarily on lowering the entry barrier for FV. Getting the requirements right is crucial to the success of any solution. So this phase started with identifying the user needs and establishing the requirements. A survey was conducted with users of varied FV expertise, to gather data on what they consider as roadblocks for FV adoption. Some interesting feedback was received that helped the team in shaping the methodology:

1. FV Setup was a bit alien to the designers and they wished for a seamless methodology as DV would have.
2. System Verilog Assertions (SVA) and formal friendly property coding was another challenge called out.
3. Some users articulated their concerns with immediate support.
4. Training was a common ask from most designers.

A GUI (Figure 2) was developed to meet the first user requirement, which automated most of the processes under the hood, using specialized knowledge of our graphics design conventions to streamline the formal setup. This GUI

is agnostic to the project and uses certain classes' specific to the GT design, which can be easily scaled to any other designs.

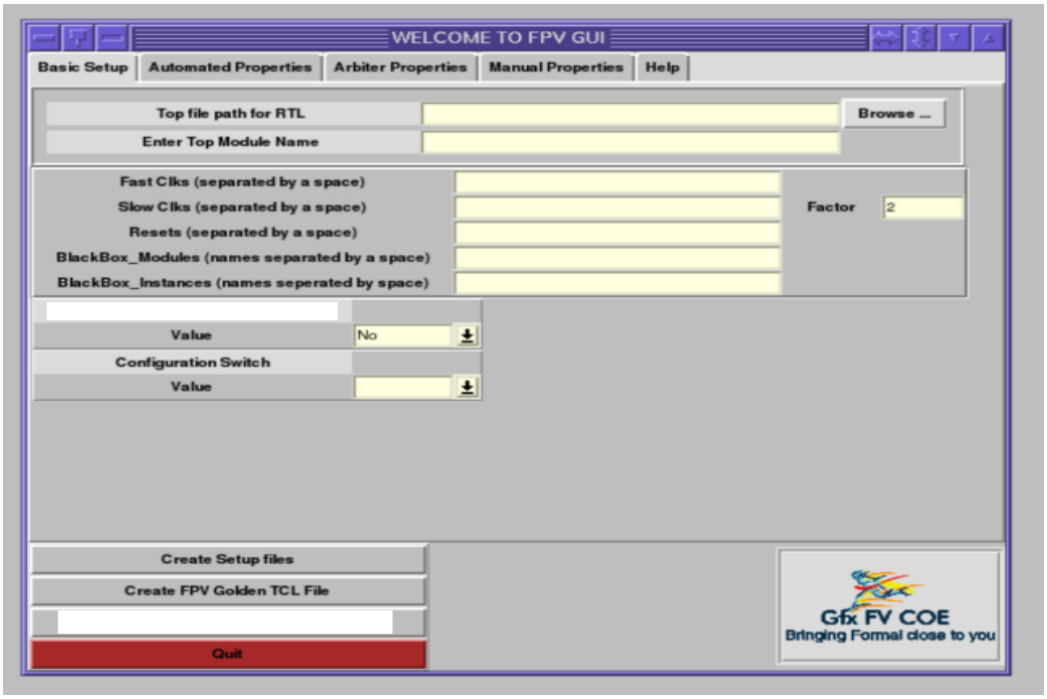

Figure 2: GUI Design Select Page

To help users in property creation, the Intel Graphics (GT) RTL was reviewed and cookie-cutter property templates were created for frequently used modules such as Arbiters, FSMs, and Standard Interfaces [3]. These templates were then integrated in the FV GUI as a one stop solution.

B. Training Phase

While analyzing the question of how to support new users in their day to day activities, it was agreed upon that the existing FV Centre of Expertise (FVCOE) team might not be able to cater to all the support requests efficiently due to bandwidth/time-zone restrictions. The FV Team succeeded in garnering support from the management to identify an FV champion for each cluster, who would be the first point of contact and the local expert for all FV queries. When a query was too complex for the local champion, it would be resolved by the FVCOE team. The cluster FV champions were trained rigorously on FV methodologies. The feedback from the FV champions and the unit owners has been very motivating and some coarse corrections were made to make the process easier. These FV champions would continue to own the activities across projects.

Additionally, sufficient self-help materials were created and made available in accessible SharePoint sites, so that users could overcome the teething troubles with online information. This comprised of Frequently Answered Questions (FAQs), cheat-sheets for different FV applications, working exercises (commonly called Dojos) with sufficient step-by-step instructions to help novice users, and links to useful papers and presentations.

A 2-week dedicated FV training was conducted for all FV champions and interested designers. The training catered to meet the expectations of varied expertise level users. The training included topics such as FV basics, FV applications for specific verification problems, and working demos. The identified FV champions were given additional trainings on tackling convergence issues and on more complex areas. The response seen in trainings was very encouraging, with almost 85 engineers becoming formally literate. A round-the-clock formal assistance network from various resources was available after the training.

C. Test Planning

FPV activity is always considered as a step function, the critical sampling events being only the start and end of the task. DV activity can be easily tracked by management by monitoring the regression pass/fail percentage status, but

there are typically not such obvious checkpoints for FV. This is due to the following limitations that FV users face regularly:

1. FV goals were not clearly defined.
2. Sometimes designs selected for FV were not suitable for the activity, either due to non-suitability of the design or the design size is beyond what the tool can churn.
3. There is no standard way to decompose the FV task in smaller sub-goals.
4. Usually, the FPV activity needs some design knowledge during "wiggling", the iterative debug stage during early constraint development, due to which the completion may depend on designer's bandwidth.
5. One of the most common question for an FV user is: "When to call the activity done?", "How good is the quality of the assertions & checks?" In the absence of an established way to answer these questions, it was quite difficult to mark the closure for the FV activities.

We implemented these techniques to deal with the above complications:

1) *Clear Goal definitions:* Before starting on any FV activity, the design was evaluated for FV feasibility depending on the design style, gate-count and logical clarity of the specification. If required, the design scope was pruned or limited to a sub-module of the larger design. The goals for execution were defined for each module to be either pure bug hunting or bug elimination depending on the complexity.

2) *Progress Continuum:* A novel method of progress continuum was implemented [4]. The progress of a FV activity was assessed through a detailed FV testplan and graded over time. A sample testplan is shown in Figure 3 and its progress graph is shown in Figure 4. The task was divided into various smaller sub-goals, where each goal comprised completing a set of pre-determined number checks/assertions under a constraint/assumption set. This allows user to start with an over-constraint environment thus wiggling only few interface signals, and thereby slowly relaxing the constraints, adding required assumptions and additional checks. The execution commitments were also staggered with respect to the applied assumption set.

Over Constraint	Checker set 1	Checker set2	Checker set3	Checker set 4	Checker set 5
Phase 1	1 week				
Phase 2	1.5week	2 weeks			
Phase 3	2 weeks	2.5weeks	3 weeks		
Valid Constraints	3 weeks	3.5 weeks	4 weeks	5 weeks	6 weeks

Figure 3: Sample Execution Testplan

For example, on one arbiter design, Phase1 consisted on only allowing 2 agents and shutting off scan and other 2 agents, Phase2 comprised of allowing 3 agents, Phase 3 allowing all 4 agents, while the DFX (Design for Test/observability/verification) and Scan (scan important registers) activities are disabled. The final phase had all the valid constraints. The Checker set 1 constitutes only basic wiggling checks, Checker set 2 had all checks related to safety properties for 2 contenders, Checker set 3 had checks for all contenders, Checker set 4 had checks that include the liveness properties and the set 5 had all checks including scan, DFX and other properties.

3) *Activity Ownership:* To guarantee designer support, the designer had onus of monitoring the FV activity completion. FV completion was included in the check-lists for all project deadlines and every designer was held responsible. Also, if the FV user was new to the design, sufficient time was accounted in test-plan for understanding the design through signal wiggling experiments that can be done using the FV tool. The central FV team was actively involved in all activities and supported the FV users' right from planning till closure.

4) *Closure:* The closure of FV activity was also tracked through the test-plan to ensure the user has completed the necessary actions before calling the activity done, to ensure the quality. The various steps to measure the activity coverage and to mark completion based on this data are detailed in next sub-section.

C. *Execution & Closure*

During this phase, each identified FPV owner implements the properties identified in the Test Planning phase using the tools and techniques he/she learnt in the Training Phase. This phase can be broadly classified into three sub phases:

1) *Execution and Tracking:* The user would code the properties using the FV GUI and also use the cookie-cutter property set for the arbiters, FSM's etc. The progress was tracked by the FVCOE along with the needed support. Weekly tracking data was published and the management was updated about the progress of the overall FPV activity. One such graph for one of the unit is as shown in Figure 4. Additionally, the bugs found by the users using FPV was also published which acted as a force multiplier.

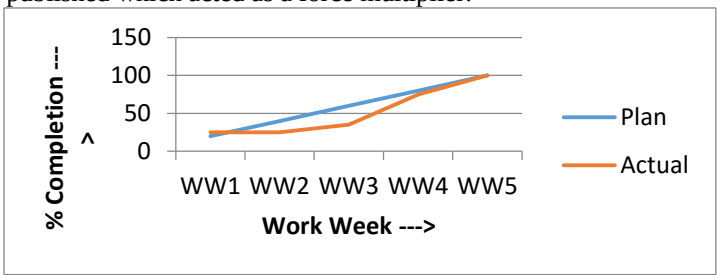

Figure 4: FV Progress Tracking of an Activity

2) *Bug Hunting:* "Bounded proof", a weaker result than "full proof" is common in FPV. In modern FPV usage models, bounded proofs are often accepted for signoff, but do require some care; we aimed for full proofs when possible. Different abstraction techniques were used in some designs which helped to get a full proof on the property desired. If the property still couldn't converge, an extensive bug hunting exercise was carried out on such properties using advanced engine settings. This, along with the proven cover properties, provided high confidence to enable bounded signoff.

3) *Coverage and Closure:* Any DV is incomplete without collecting coverage metrics. A similar exercise is needed at the end of the FPV exercise to say if the user has indeed covered all of the RTL. A line/statement and branch coverage was mandated at the end of the FPV exercise. The users were able to identify some interesting coverage holes during the exercise.

D. *Communicate Results*

If applied effectively, FV provides verification confidence that cannot be achieved otherwise. Articulation of the formal results in a common speaking technical jargon was important. The most common factor considered while deciding on continuation of any methodology is the ROI that it provides. Thus, it becomes mandatory to keep updating the management with ROI information. The challenge is to procure realistic ROI numbers, to be compared against DV, and the elements to be included for ROI assessment.

To address the above-mentioned challenge, a new metric for ROI calculation, which was first proposed by authors in [1] was use to estimate ROI on a given activity. The parameters used are summarized in Table 1.

Table 1: ROI Parameter Definitions

Parameter	Depiction
Number of engineers worked on the problem	N_{E_F}
Average bandwidth/engineer	B_{W_F}
Number of weeks spent on effort	W_{w_F}
Total time spent in hours	$T_{s\_F} = B_{w\_F}{}^{a} W_{w\_Fa}{}^{a} 40^{b}$
Project schedule in weeks	P_{w_F}
Scaled engineering costs	$C_{E\_F} = T_{s\_F}/(P_{w\_F}{}^{a} 40)$
Machines used for computations	M_{c_F}
Runs/week	R_{w_F}
Number of days of run	D_{w_F}
Total machine-days/week	$T_{m\_F} = M_{c\_F}{}^{a} R_{w\_F}{}^{a} D_{w\_F}$
Project machine-days/week	P_{m_F}
Scaled machine costs	$C_{m\_F} = T_{m\_F}/P_{m\_F}$
Coverage (% cover points hit)	Cov_{tot_F}
Bugs found by the method	N_{B_F}

[a] *Assuming eight hours/day for a five-day work week = 40 hours/week.*
[b] *Machine runs are not limited to a working day but for the whole week = seven days/week.*

The methodology followed is outlined in Table 2. The notation uses suffix "_D" to indicate DV and "_F" for formal methodology. The ROI numbers can give a decent picture of the comparative effectiveness of the methodologies in discussion. During the FV deployment drive at Intel Graphics, all the users were requested to tabulate all the details required for these calculations from the inception till the logical conclusion, that helps the calculation of ROI of each activity towards the end of the activity.

Table 2: Basic Comparisons of FV and DV ROI

ROI Parameter	Formal	Dynamic	Relative
Bugs% found by FV	$N_{B\_F}/(N_{B\_F}+N_{B\_D})$	$N_{B\_D}/(N_{B\_F}+N_{B\_D})$	N_{B_F}/N_{B_D}
Total coverage achieved	Cov_{tot_F}	Cov_{tot_D}	
Bug - engineering cost ROI	$B_{EC\_F} = N_{B\_F}/C_{E\_F}$	$B_{EC\_D} = N_{B\_D}/C_{E\_D}$	B_{EC_F}/B_{EC_D}
Bug - machine cost ROI	$B_{MC\_F} = N_{B\_F}/C_{m\_F}$	$B_{MC\_D} = N_{B\_D}/C_{m\_D}$	B_{MC_F}/B_{MC_D}
Coverage - engineering cost ROI	$C_{EC\_F} = Cov_{tot\_F}/C_{E\_F}$	$C_{EC\_D} = Cov_{tot\_D}/C_{E\_D}$	C_{EC_F}/C_{EC_D}
Coverage - machine cost ROI	$C_{mC\_F} = Cov_{tot\_F}/C_{m\_F}$	$C_{mC\_D} = Cov_{tot\_D}/C_{m\_D}$	C_{mC_F}/C_{mC_D}

Some practical values of the above values for one of the unit we tried on our designs gave us a number of 130x as Bug-Engineering costs and 1096x on Bug-machine ROI.

E. Check-In & Regress

In the DV world, regression plays an important role in catching bugs. A similar methodology is required in FPV towards the completion of the initial FPV activity. The RTL would undergo change due to addition of late features or due to feedback from the Structural Design (SD) team. If the FV owner is unaware of these changes, then his entire FPV effort comes to a naught. Hence, this phase is equally important.

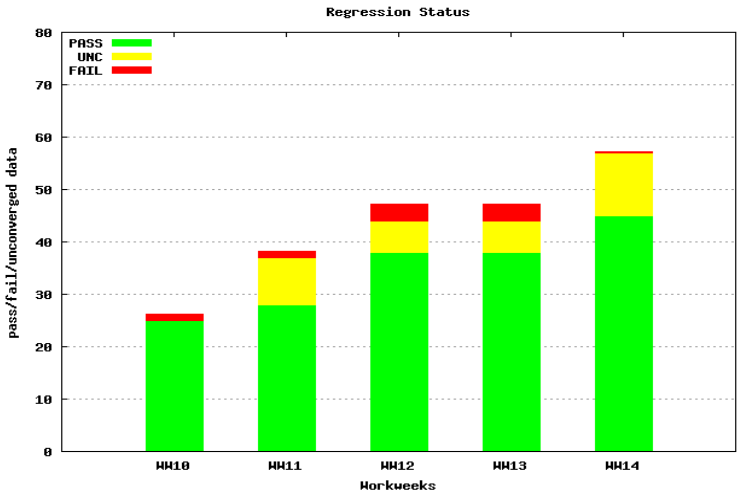

Figure 5: Regression Monitoring

A formal GIT repository (Database to hold all proofs and scripts) was created which would house all the FPV related files. Once the user completes his FPV, he turns in the FPV collaterals to this repository. An array of scripts were created which would help the user to enable regression on his recently concluded FPV unit. These scripts would leverage the FPV collaterals in the repo and launch a regression on the user specified unit at the prescribed interval, say, once every two weeks. With the help of these scripts, regression would be enabled on the unit and periodic reports sent to the user/manager. This graphical report would let the user/manager know how the unit has tracked over time. One such snapshot is published in Figure 5.

III. RESULTS

Determining the success of the deployment depends on three factors:

1) *Quality of Reception:* In a span of three weeks, 85+ engineers were trained on FPV. As a testament to the standards of the activity, around 15 bugs were caught across designs within the first two weeks of the activity, and this increased to 84 within 8 weeks. Though the activity started at a much later stage of the design cycle with a stable DV environment, the sheer number of the bugs caught by FV serves as a witness to the success of the activity. During the execution stage, the designers could, for most of the time, function independently in bringing up and exercising the FV environment. In intricate cases when the designer fell short of exercising his design completely or when the design faced convergence issues, the central team stepped in with expert solutions such as selective abstractions, pruning down the design, determining FV coverage or bug hunting strategies. Most of the bugs caught turned out to be of very high quality, with some proving to be impossible for any number of DV tests to catch. Nevertheless there are some cases where still some designers had a mental block in catching up with formal but have changed their perception after looking at regular formal success updates.

2) *Quality of Bugs Caught:* All of the bugs caught were classified by the FV effort required. The categories were - easy to catch issues (Typographical errors, for example), medium difficulty (Counter under and overflows, etc) and hard to catch bugs (starvation and bubbles). With respect to the DV effort required, even the easiest of the bugs caught would have been difficult for DV as the environment was already running and stable. Here are some examples for each of the category.

   a. *Easy to Catch Bugs:*
   These are the kind of bugs that are likely to be revealed in first couple of runs in any kind of verification methodology. Figure 6 shows a simple Finite State Machine (FSM) multi-path scenario which was quickly found by FV using the following tool generated automatic static property:

```
// State must never attempt multiple arcs
STACK_STREAM_multi: assert property (
        @(posedge clk) disable iff (!rst_b)
                (state == STACK_STREAM) |->
                $onehot0({raw_arc_STACK_STREAM_STACK_POP,
                raw_arc_STACK_STREAM_STACK_PUSH});
```

   This bug though easily caught by FV would have resulted in an X-propagation issue in DV which would have been quite difficult to troubleshoot. Here, from current state (STACK_STREAM), two transitions are active on the same clock: “STACK_STREAM -> STACK_POP” & “STACK_STREAM -> STACK_PUSH”. This will make the next state X which is a valid RTL failure.

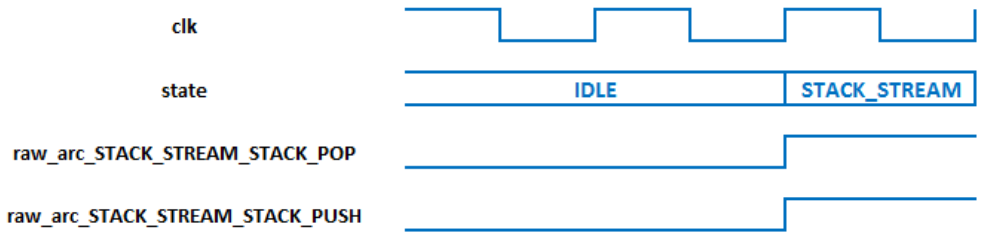

Figure 6. FSM failure

   b. *Medium Difficulty Bugs:*
   These scenarios require one to delve much deeper into the design. One such example is described in Figure 7, where an arbiter permanently starves one requesting agent.
   The arbiter has 3 requestors, Req1, Req2 and Req3. The requestors would be granted across 2 ports, only when the ports were available. Req1 would get a grant on port1 only, when p1 is available. Req3 would get a grant on port2 only, when port 2 is available. Req2 could get grant on either port1

or port2, on round robin basis. And on the top, there was a round robin scheme present in between Req1, Req2 and Req3.
The failing scenario was manifested only when following conditions were met:
• First few requests are serviced such that output FIFO gets full and both output ports become unavailable.
• After both ports become unavailable, all 3 clients assert request, and these requests are high continuously
• Output ports (port a & port b) start becoming available alternately (depending on output FIFO read conditions)
In Figure 7, after cycle 7 whenever porta becomes available, arbiter is giving grant to Req2 and whenever port b becomes available, arbiter gives grant to Req3. Req1 requests are totally ignored.

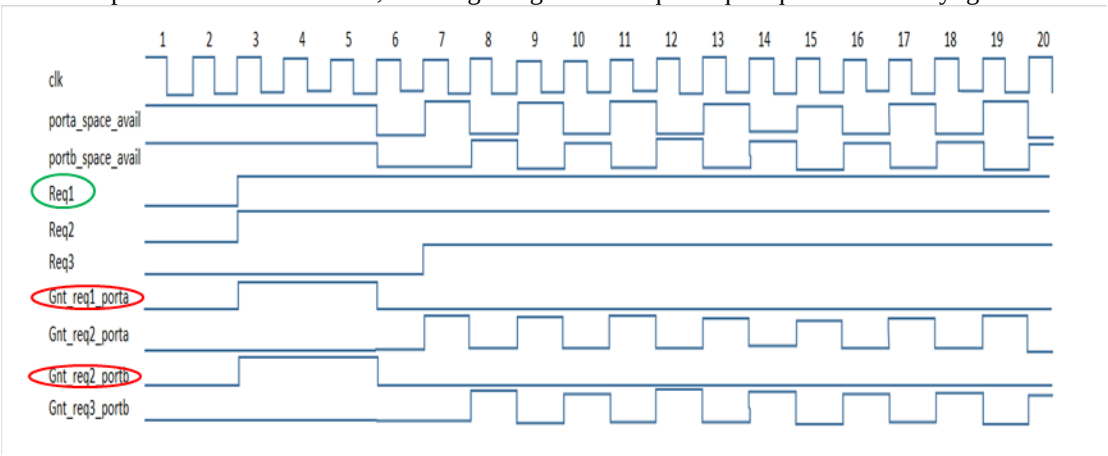

Figure 7: Arbitration Failure

The necessary input conditions for revelation of this bug are quite complex and next to impossible to be created using DV.

c. *Extremely Hard to Catch Bugs:*
This category of bugs comprises of complex state-control interaction bugs, where the bug is manifested only when the design is in a particular state and some particular input sequences are seen at input interface. One such example is shown in Figure 8. In this example, bug is exposed only when internal counter (CI) reaches a value of 32'h8001_0000 and at the input A is received followed by 3 consecutive Bs. This is a very corner case scenario and extremely hard to catch using DV.

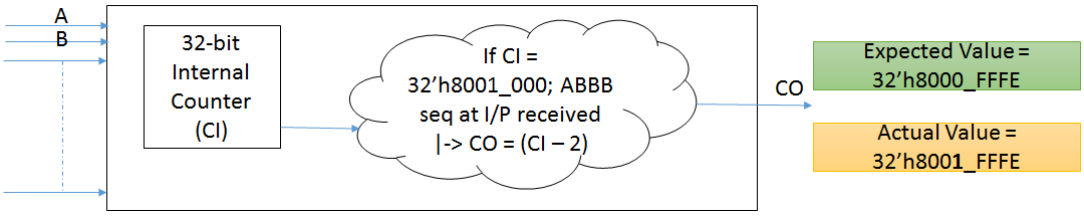

Figure 8: Example of Extremely Hard to Catch Bug

3) ROI:
All FV activities undertaken during the FV deployment drive on the ongoing Intel Graphics project were evaluated w.r.t. ROI as per the formulae given in Table 2. After consolidation of all the results, a weighted ROI of 220x was seen as compared to DV.
ROI comparison data of one such activity is given in Figure 9. Here, FV was found to be 138x efficient w.r.t engineering cost and 3692x efficient w.r.t. machine costs. Of course these numbers might seem extremely large (and FV skeptics might argue we are biased!), but weightings were generated based on expected time to find the bugs in simulation: as we saw in the previous situation, FV quickly found some corner cases that would have taken years to randomly simulate. Based on previous project bug data, such cases would have required many times the resources to debug at late project stages vs the effort spend on FPV. We also collated the data of bug costs w.r.t the level of verification, which shows an exponential cost increase due to late bugs.

Clearly communicating such finds played a major role in increasing designer's emphasis on this FPV effort. The designers were excited because everyone wants the prestige of being a unit owner with no late bugs.

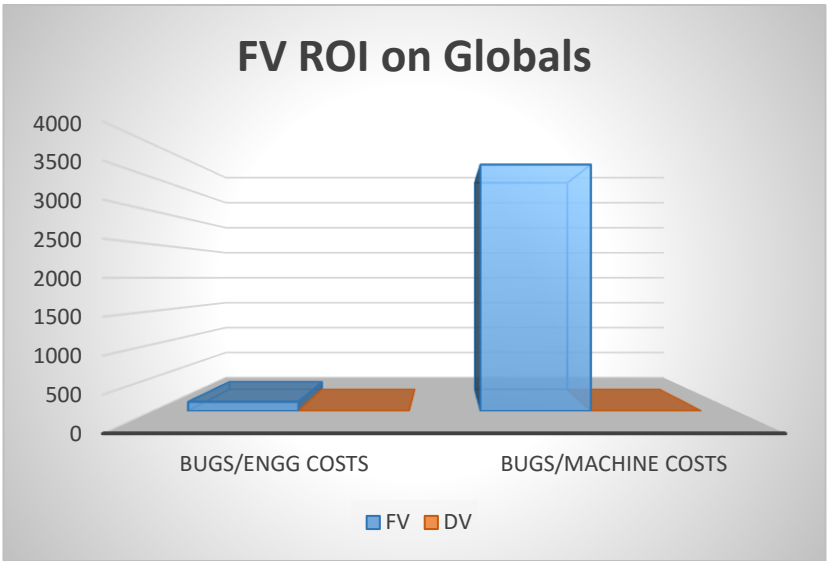

Figure 9: ROI Comparison for a Sample Activity

IV. Conclusion

Formal Verification, despite its benefits, has still not seen mass-scale deployment in ASIC verification world. This paper discussed the factors that has been limiting its adoption, and described a comprehensive approach that conquers these restrictions. This approach was demonstrated on a live Intel Graphics project. The process of this deployment was detailed and the results were recounted. In this deployment drive of 12 weeks, over 80+ engineers were made FV experts, 85+ bugs were found and designers'/management's confidence was won with convincing ROI numbers.

The deployment exercise discussed was commenced at a very late stage of design cycle. Authors believe that if the proposed methodology is exercised from the start of project, it can provide significant improvement in the quality and effectiveness of verification.

Acknowledgments

Sincere thanks to our managers Subeer Patel, Sajjad Zaidi and Pradeep Raghavendra in showing confidence in our vision and supporting us throughout in this journey.

References

[1] Erik Seligman, M V Achutha KiranKumar, and Tom Schubert, "Formal Verification- An Essential Toolkit For Modern VLSI Design," Elsevier Publications, 2015.

[2] M V Achutha KiranKumar, Aarti Gupta, and Ss Bindumadhava, "RTL2RTL Formal Verification," DAC 2015.

[3] Erik Seligman, Laurence S. Bisht, and Dmitry Korchemny, "SystemVerilog Assertion Linting: Closing Potentially Critical Verification Holes," DVCON 2012.

[4] Formal Verification Training, Vigyan Singhal, Oski Technologies, DAC 2015