2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Making Formal Property Verification Mainstream: An Intel® Graphics Experience

M, Achutha KiranKumar V, Intel Technology India Pvt Ltd, Bangalore (*achutha.kirankumar.v.m@intel.com*)

Bindumadhava S S, Intel Technology India Pvt Ltd, Bangalore (*bindumadhava.ss@intel.com*)

Abhijith A Bharadwaj, Intel Technology India Pvt Ltd, Bangalore (*abhijith.a.bharadwaj@intel.com*)

***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 the potential of application, 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, only few FV-experts work in small pockets and are able to cover only a small portion of the FV-able design space. This is a matter of conern, as 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:

FV is commonly considered an esoteric methodology by general masses. A naïve user is apprehensive about learning and implementing it.

There is no established metric to track and convey FV progress to the management which is well-accustomed to see standard progress reports by the simulation based verification flows.

FV user often struggles to translate his/her efforts to ROI in various non-standard reporting forms. If these efforts are not appropriately understood and acknowledged, the user/management may eventually discontinue the FV usage.

In this paper, a sincere effort has been made in order to bridge the FV adoption gap by addressing the above problems. To address the first issue where new users were impeded by the presumed complexity of FV methodology, conscious efforts were spent to lower the entry barrier. This resulted in creating a Graphical User Interface (GUI) that automated environment setup for all FV applications, elaborative trainings, enough self-help materials in place with detailed walk-through examples, creating cookie-cutter assertion libraries, etc

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 and needs to be

addressed by aligning with the project progress metrics. This equips the management to follow the FV progression more closely and appraise it better.

In order to overcome the third hindrance of effectually articulating FV ROI to the management, a new methodology that standardizes the planning, tracking, reporting and assessing the FV efforts was proposed. This methodology helps users to communicate FV progress and ROI in a more linear fashion 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 projections interested the designers. With proper motivation and realistic figures, the Management join the bandwagon of promoting FV on their 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 and published during the test-planning phase, which could 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 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.

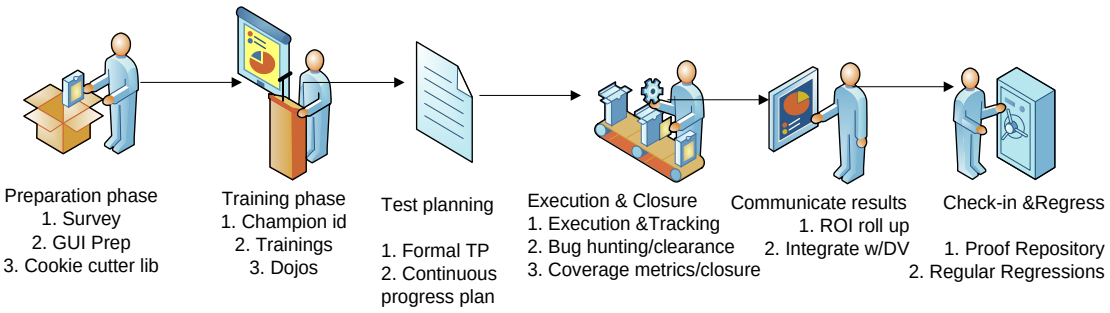


Figure 1: Deployment Methodology

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 any DV.
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.

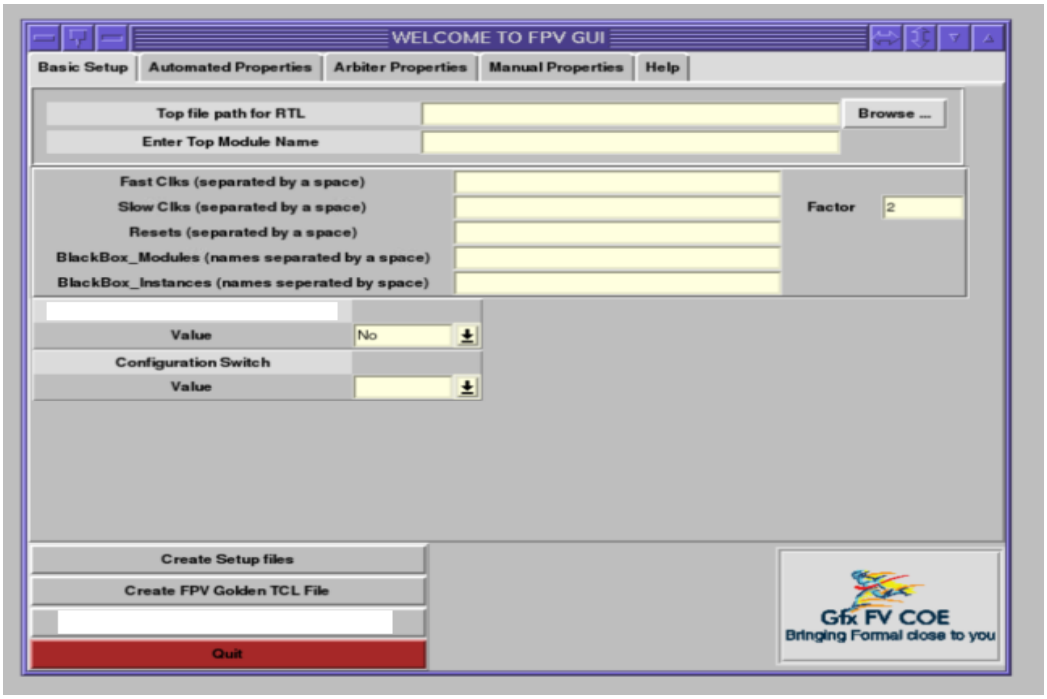


Figure 2: GUI Design Select Page

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. FV Team succeeded in garnering support from the management to lay modus operandi, where an FV champion was identified for each cluster who would be the first point of contact and the local expert for all FV queries from the cluster and complex queries to be resolved by the FVCOE team. The cluster FV champions were trained rigorously on FV methodologies.

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 etc.

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 ranged from FV basics, designs suitable for FV to FV applications for specific verification problems to 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 commissioned post 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. While DV activity can be easily tracked by management by monitoring the regression pass/fail percentage status, but there are no such available 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 scalable.
3. There is no standard way to decompose the FV task in smaller sub-goals.
4. Usually, the FPV activity needs some design knowledge for coding assumptions during wiggling 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 [5]. 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 in 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 set 2	Checker set 3	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

3) *Activity Ownership:* To guarantee designer's support, the designer had onus of monitoring the FV activity completion. FV completion was included in check-list 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.

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.

D. Execution & Closure

During this phase, each of the 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 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.

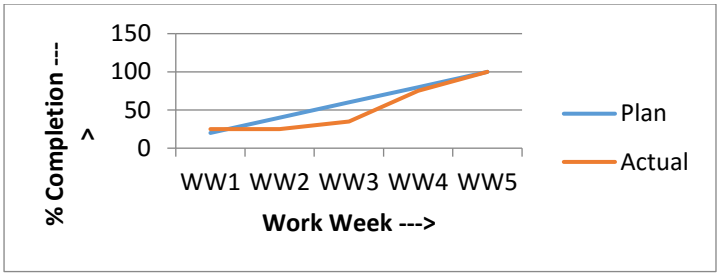


Figure 4: FV Progress Tracking of an Activity

2) ***Bug Hunting:*** Bounded proof, an ugly step sister of the full proof is common in FPV. With the bounded proof, the user is always on his toes worrying if he has missed any bugs. 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 the user in the design he/she has verified.

3) ***Coverage and Closure:*** Any DV is incomplete without collecting the 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.

Throughout this phase, an FVCOE expert was nominated and provided support to the users in their FV related queries.

E. 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 the need of the hour. 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 5 40 hours/week.*
[b]*Machine runs are not limited to a working day but for the whole week 5 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}

F. 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 intial 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.

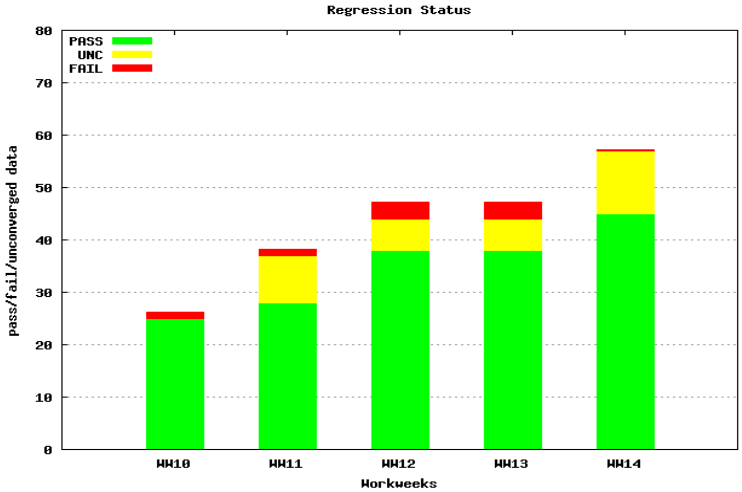


Figure 5: Regression Monitoring

A formal GIT repository was created which would house all the FPV related files. Once the user completes his FPV, he turns in his 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 which 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 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.

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 scenarios required directed assertions to be written, to verify a certain property or a behavior. The example considered here is an arbiter which gives grants to three kinds of requests – read requests, write requests and read returns, depending on credit availability. There are four read/write clients and one read returns (highest priority). The fub has two command and one data output ports. Other than arbitration, the module should also intelligently pack the requests. That is –

- If 2 read requests are present in priority, both requests are given grants.
- If 2 writes are requesting in priority, only one write is granted as the module has only one data port.
- If a read and a write/memory return request, then both can be granted as reads go on command port and write/memory returns go on data port.

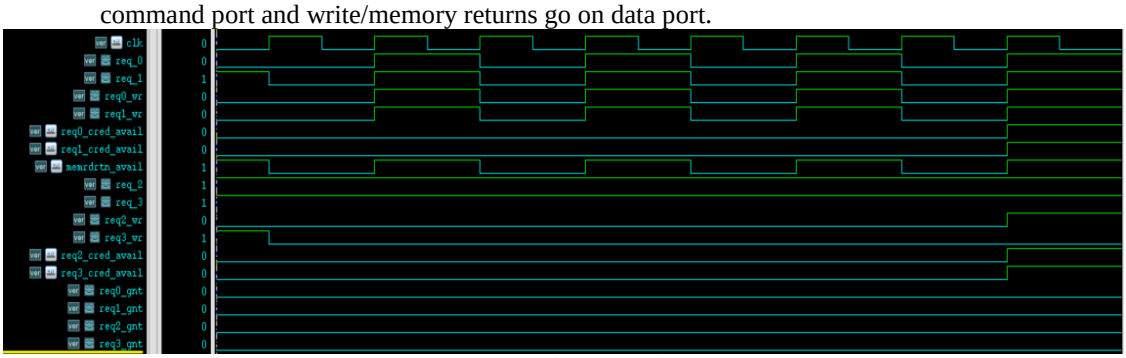
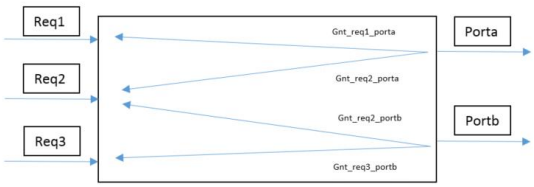


Figure 6. Meta-packing failure waveform

The scenario caught was, when a memory read return requests with client 3 read, and only these two clients are requesting, then only the read return was being granted. The arbiter was not intelligent in this case and the error happened only with client 3.

b. *Medium Difficulty Bugs:*

TThese scenarios require one to delve much deeper into the design. This is a proper definition for a 'corner case scenario', which is next to impossible for DV to catch**.**



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 scenario was as follows:

- 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)
- 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.

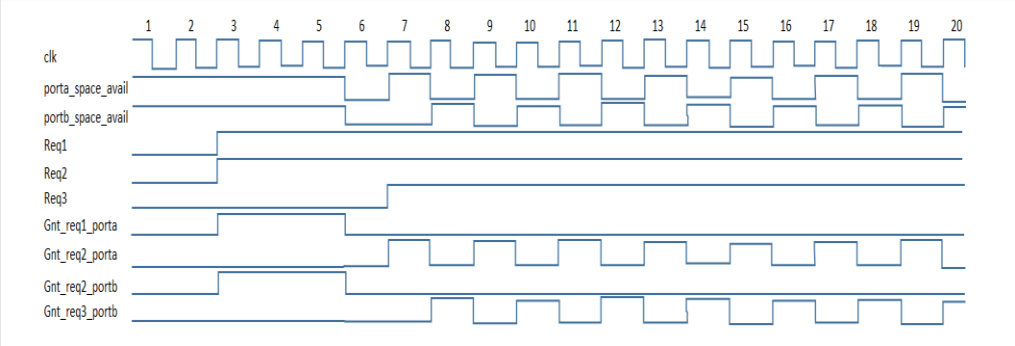


Figure 7. Starvation failure waveform

c. *Extremely Hard to Catch Bugs:*
This category of bugs is usually comprised of complex state-control interaction bugs, where bug is manifested only when design is in a particular state and some particular input sequences are seen at input interface. One such example is shown in Figure 7. In this example, bug is exposed only when internal counter (CI) reaches a value of 32'h8001_0000 and at input A is received followed by 3 consecutive Bs. This is a very corner case scenario and extremely hard to catch using DV.

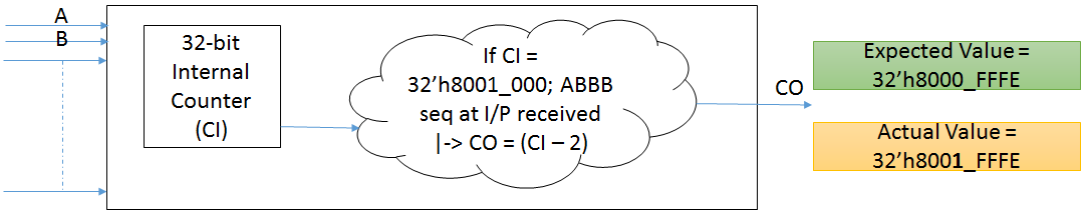


Figure 8: Example of Extremely Hard to Catch Bug

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, a weighted ROI of 220x was seen as compared to DV.

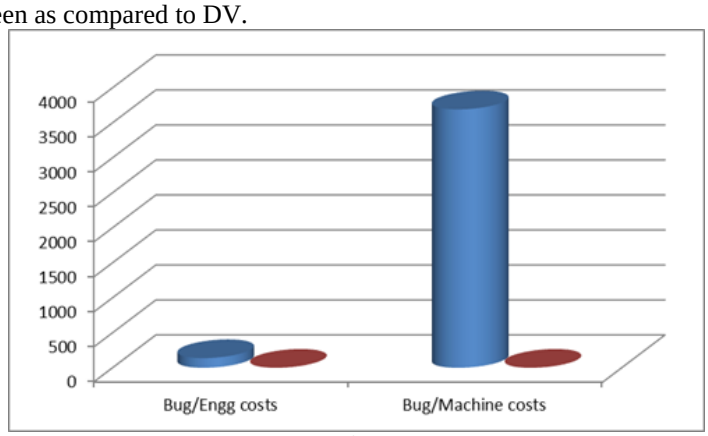


Figure 9: ROI comparison for a sample activity

ROI comparison data of one such activity is given in Figure 7. Here, FV was found to be 138x efficient w.r.t engineering cost and 3692x efficient w.r.t. machine costs. Note that some bugs found by FV had a very high weightage, resulting in the huge ROI advantages in Figure 7. This is because DEs found a surprising number of tricky corner-case bugs that would have been very unlikely to be found by traditional methods and would have required many millions of random cycles to hit through DV. Based on previous project bug data, such cases would have required at least 25x the resources to debug at late project stages vs the effort spend on FPV. We also collated the data of bug costs wrt the level of verification, which shows an exponential cost increase wrt the late bugs. Clearly communicating such finds played a major role in increasing designer's emphasis on this FPV effort. On a side note, everyone wants the prestige of being a unit owner with no late bugs!

REFERENCES

[1] Erik Seligman, M Achutha KiranKumar, and Tom Schubert, “Formal Verification- An Essential Tool kit for the modern VLSI Design,” Elsevier Publications.

[2] M, Achutha KiranKumar V, 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] Laurent Arditi, “Formal verification in ARM,” JUG 2014.

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