2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Enabling high quality design sign-off with Jasper structural and auto formal checks

Vishnu Haridas*, Mansi Rastogi‡ and Guruprasad Timmapur†
Intel Technologies India Pvt Ltd
Bangalore, India
*vishnu.haridas@intel.com, ‡ mansi.rastogi@intel.com, †guruprasad.timmapur@intel.com

***Abstract* —Ever-shrinking project schedule and the need to deliver high quality IPs make effective RTL signoff, an important part of the overall RTL development process. Deploying a robust and comprehensive verification tool can reduce wasted debug cycles in later phases of the project and eliminate expensive iterations of RTL clean-up. In this paper, we describe a new formal based Jasper Superlint application to achieve robust RTL delivery, by enabling the designers to find bugs upfront through structural and auto-formal checks, thus accelerating the RTL-validation convergence. The paper also discusses the challenges faced while tuning the tool to produce accurate results.**

I. INTRODUCTION

There is an increased need of finding bugs at the early stages of IP development cycle and to reduce escapes to sub-system and SoC (pre-Si, post-Si). The cost of fixing the bugs becomes exponentially higher for later stages of the development cycle. Hence, it is inevitable for IP teams to build high quality RTL and the onus is on designers to hand-off quality RTL with all checks and balances in place, e.g., Unit level tests, auto-formal checks and various TFM checks. Deploying Jasper Superlint as part of the design process helps in achieving the aforementioned objective. In addition to comprehensive set of structural lint checks e.g., naming, coding style, simulation-synthesis mismatch, arithmetic overflow and out-of-bounds index that helped to provide the level of assurance necessary for signing off today's designs, the tool also provides best-in-class auto formal checks e.g., dead-code, FSM reachability/deadlock/livelock checks for functional verification of many aspects of the design, using properties derived automatically from the RTL.

The key component of this technology is development of a robust formal environment/setup that is independent of the features or the assertions that are being validated. Jasper Superlint requires no new work in terms of defining new stimulus to test new features. Since the tool assumes all possible inputs combinations and its values possible, we need to add constraints/restrictions to exclude behaviors that can result in huge runtime, hangs and false counterexamples. Our experience has shown that this can be a faster task to complete as writing assumptions is a one-time effort and can be extended to verify any new features of the IP.

II. METHODOLOGY

In this section we will describe the overall flow for doing RTL signoff to pre-Si validation team using Jasper Superlint tool. The Fig.1 below provides a roadmap for the methodology flow. The RTL files of the respective module/design under test are given as input to the Superlint tool. The required rules/checks are enabled as part of setup creation. Structural violations are reported in the compilation stage itself. Once compilation is successfully completed, auto-formal properties are automatically extracted by the tool from the DUT [1]-[2]. A series of constraints or assumptions are added on the input to the DUT to prevent illegal behaviors and eliminate false failures. Sometimes, to deal with complex issues, we need to abstract the behaviors of the surrounding logic. The development tends to be an iterative process that goes through iterative loops of "compile -> proof-run -> debug -> add restrictions/modelling" as shown in Fig.1. We start with an environment that provides unrestricted freedom of inputs and through discovery of false failures, gradually constrict it to valid microarchitecture behaviors.

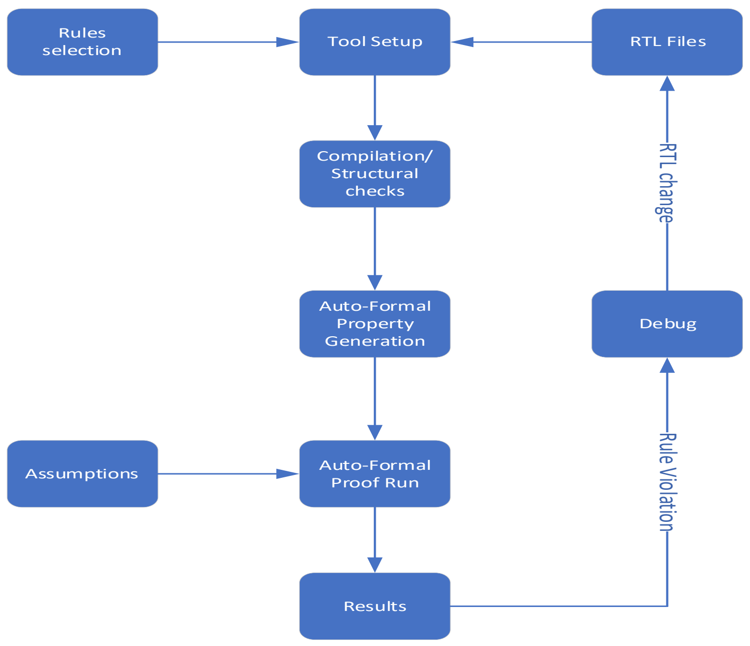

Figure 1. Jasper Superlint tool flow

III. RELATED WORK

Given below are some case studies where we applied the above-mentioned methodology in various IPs.

A. *Case Study 1: Using Jasper Superlint to deliver a complex clock management feature with ZERO bugs*

New clock management feature was added into the legacy IP to implement a low power requirement. This implementation affected many legacy modules along with the addition of the new module. With the introduction of this new module, a new FSM was implemented which works on a gated clock. The validation effort was decomposed into two parts, one for legacy behavior of IP and one for verifying the new module.

For the verification of new module, a logic was modelled in Superlint setup to generate a gated clock with one enable pin and the same was fed to leaf level module. All the comprehensive structural and auto-formal checks like reachability check, deadlock check, livelock check were performed on the FSM. The gated clock implementation helped to create all the complex corner case scenario which resulted in the correction of the architecture definition of the feature in a very early stage. When the RTL was delivered to the verification team after all the early bug fixes, there were ZERO bugs discovered by the verification team.

B. *Case Study 2: Finding a complex corner case scenario in the legacy FSM*

As part of the new requirement, an error handling mechanism was implemented in a legacy FSM which manages data streaming operation into the destination interface. As per the new feature, whenever an error is detected by the target module, the initiator drops all the data in the pipeline and start the transfer again. There was a corner case scenario where the hang would occur in the FSM whenever the pointer reaches a value greater than 5 on the exact clock where the data transfer is reinitiated. This issue had escaped the pre-Si test environment. However, the Jasper auto-formal check exposed the hang in the FSM with its auto generated potential deadlock check on the FSM.

Initially there were couple of challenges, like run time issue where the potential deadlock assertions were not getting converged. With continuous interaction with Cadence support team and internal Formal team multiple strategies were applied. The outcome was that in case of complex designs we need to convert these liveliness potential deadlock assertions to safety assertions as shown below [3]. It means that, instead of checking the FSM hang till the end of simulation, the tool is made to check if the state machine would get hung for 100 cycles, which ensures that the hang scenario is real:

Liveliness assertion**:** *assert -env {(i_top_wrapper.EN_MODULE.i_module_wrapper.i _datapath.dipsm_ps == 6'd8) |=>* ***##[0:$]*** *(i_top_wrapper.EN_MODULE.i_module_wrapper.i _datapath.dipsm_ps != 6'd8)}* [3]-[4]

Safety assertions: *assert -name {<embedded>::safety_assertion} {(i_top_wrapper.EN_MODULE.i_module_wrapper.i _datapath.dipsm_ps == 6'd8) |=>* ***##[0:100]*** *(i_top_wrapper.EN_MODULE.i_module_wrapper.i _datapath.dipsm_ps != 6'd8)}* [3]-[4]

The safety assertion converged faster and proved the issueto be real. When the RTL fix was done, the same safety assertion was applied to prove that the hang no longer exists to validate the fix and to deliver it to the customers.

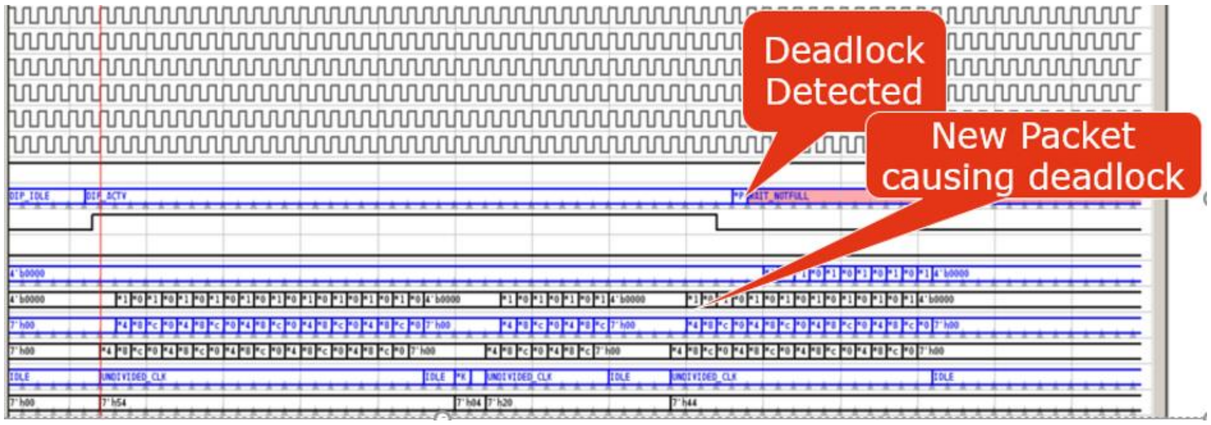

Figure 2. Waveform of the FSM Deadlock scenario created by the Superlint

IV. RESULTS

During the development stage of multiple IP's Jasper Superlint usage for an early validation has reduced the amount of late discovery of bugs, enabled robust design, and reduced the quantity of bugs at pre-silicon validation stage. The below chart shows the number of critical bugs observed across debug IPs during RTL and VAL signoff.

The structural checks of the tool found 2 bugs. The tool was able to find 2 deep corner case deadlock scenarios which was difficult to detect with simulation runs. Certain deadlock and dead-code scenarios were also detected. This has reduced the amount of late discovery bugs.

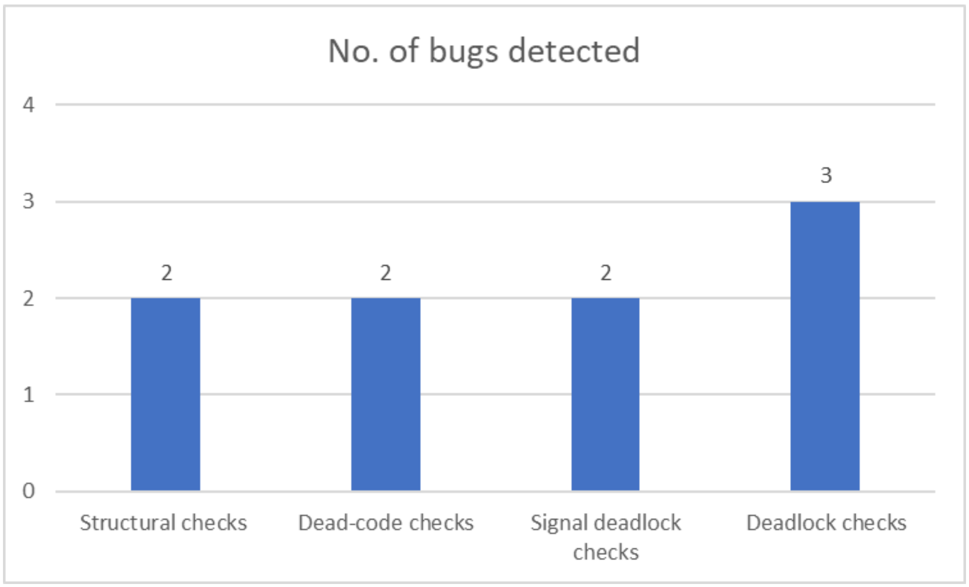

Figure 3. No of bugs detected by Superlint tool during each of the Structural and Auto-formal checks

IV. CONCLUSION

This paper talks about a comprehensive technology application that was successfully used in many projects to pull-in bug finding capability. The methodology presented in this paper provides a standardized flow that can be a powerful tool in the hands of the designers to efficiently complete early validation of new features and then handoff the pre verified RTL for pre-Si. Jasper Superlint was successfully deployed to validate the new features in the IP. This effort was completed while dynamic validation was coming in-progress – no logic bugs have been found by pre-Si validation team that resulted in ZERO bug RTL delivery of RTL. As the designers gained experience with the tool, the techniques were integrated more tightly into the design process and deployed to various new IPs to incrementally check partially coded RTL, preventing bugs from being released. The effort resulted in healthier and more robust RTL at a much earlier time which paved the way for successful release of the feature on schedule. This tool lent itself very well to exploit the strengths of the designer who has the intimate understanding of the DUT. In addition to number of bugs found, there are clear advantages of using Jasper Superlint:

1. Provides comprehensive structural analysis as compared to traditional linting.
2. Prevents Si escapes.
3. Significantly pull in of bugs with reduced time to market.
4. High confidence level while doing RTL signoff.
5. Automatically generated assertions based on the RTL.
6. Eliminate expensive iterations of RTL clean-up.

To conclude, designers can use this tool as an effective early validation vehicle and thereby dramatically improve RTL quality. Based on our experience in debug IPs, and the results achieved, we believe that Jasper Superlint gives a very high ROI and strongly recommend using this technology in future projects.

V. REFERENCES

[1] "JasperGold Superlint App User Guide", Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA. Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

[2] "JasperGold Superlint Checks Reference" Product Version 2021.09
September 2021, Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA. Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA

[3] Mark Eslinger, Jeremy Levitt and Joe Hupcey III, "Deadlock Verification For Dummies – The Easy Way Using SVA and Formal", Mentor, A Siemens Business, 46871 Bayside Pkwy, Freemont, CA 94538

[4] Ashok B. Mehta, "SystemVerilog Assertions and Functional Coverage", Third Edition, ISBN 978-3-030-24736-2