2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Formal for Adjacencies

Expanding the Scope of Formal Verification

M Achutha KiranKumar V, Formal Validation Principal Engineer, Intel, Bangalore, India (*achutha.kirankumar.v.m@intel.com*)

Ss, Bindumadhava, Graphics Hardware Engineer, Intel, Bangalore, India (*bindumadhava.ss@intel.com*)

Vichal Verma, Graphics Hardware Engineer, Intel, Bangalore, India (*vichal.verma@intel.com*)

Savitha Manojna, Graphics Hardware Engineer, Intel, Bangalore, India (*savitha.manojna@intel.com*)

***Abstract*— Pre-silicon simulation/emulation based dynamic validation (DV) has advanced by leaps and bounds to verify the latest complex hardware designs with decent confidence. However, the union of all these technologies would still leave some holes that get exposed as bugs on the silicon that motivates the designers to explore alternate methodologies to gain higher confidence. Formal Verification (FV) is one powerful validation technique that mathematically proves design correctness, rather than simulating specific test cases, and has been growing in popularity across the industry. Traditionally, FV has been found useful applications in the areas such as algorithmic circuit verification, complex control path verification, connectivity checks etc. There are many other areas where application of FV has been limited or completely non-existent since these are considered not suited for FV. In our paper, we present our experience in application of FV in some of these non-traditional areas.**

Keywords—formal verification, hierarchical sequencial equivalence, post-silicon formal, connectivity

I. INTRODUCTION

Increasing design complexity driven by feature and performance requirements and the Time to Market (TTM) constraints force a faster design and validation closure. Studies [6] indicate more than 50% of the project time is consumed by verification, and verification engineers spends 44% of overall verification time in debug, which is the maximum when compared to other tasks like test planning, test writing simulation and regression. This compels a verification engineer to think of novel ways of identifying and debugging behavioral inconsistencies. Despite extensive verification done using multiple tools and technologies, some bugs escape pre-silicon and are found at post-silicon validation stage, which brings in immediate management attention and management wants those bugs to be solved as quickly as possible to cause a minimum impact on the project schedule. The designers and managers at this stage not only want to fix the bug early but also wants to verify the bug fix. Post-silicon debug and bug fix verification using traditional simulation has been very challenging and time consuming. We proposed a methodology using formal verification (FV) which we have successfully applied to post-silicon debugs and reduced debug time, as compared to traditional simulation approach, and achieved higher confidence in verification of bug fix using FV.

RTL-RTL sequential equivalence checking using formal is already being used for verification of incremental features like timing fixes, clock gating, chicken bit[4], but these applications are mostly confined to unit level boundaries. Any incremental change at the chip level becomes challenging as RTL2RTL sequential equivalence tool hit complexity issues leaving the verification engineer with two options, to verify logical equivalence RTL-RTL and use traditional simulation to verify the functional correctness. Chip level simulation regression takes multiple days, consuming lot of time and resources which is not always suitable for the project timelines. We proposed an automation to establish the sequential equivalence, hierarchically from leaf level to chip level saving verification time and resources as compared to traditional approaches. We also present an innovative application, where we have utilized the power of sequential equivalence to establish transaction equivalence between two RTLs and found corner case control logic bugs which are very hard to find using traditional approaches. This paper also discusses how smartly applying connectivity verification can save lot of debug time and resources.

II. POST-SILICON FV

In the current design environment, even after running many cycles of verification both at simulation and emulation, there are critical bug escapes which are found post-silicon. These bugs are often hard to root-cause at the chip level since not all RTL signals are available for debug. An RTL bug which is found post-silicon is required to be root caused and fixed correctly at the RTL level. The current techniques which are being used, although help in localizing a problem, takes time to arrive at the root cause to a particular unit or a functional block (FUB). Some of the critical bugs could delay the PRQ of the product and hence causing a huge financial loss to the company. The key requirement for curtailing the compute resources would be exhaustive verification methods and expeditious debug time.

Formal property Verification (FPV) exhaustively verifies a given design against a specification coded as properties. The FPV environment uses mathematical techniques to verify the assertions, given a set of constraints (assumptions). A passing proof indicates that all possible behaviours were exhaustively verified, and the design adheres to the given specifications. In the Figure1, we describe the FPV methodology which was incorporated to resolve the post-silicon bugs on multiple units therewith not only reducing the time for debug but also proving that the RTL fix provided was correct. Although application of FV in post-silicon debugs is nothing new [5], we are presenting an optimised methodology which would help in catching bugs faster.

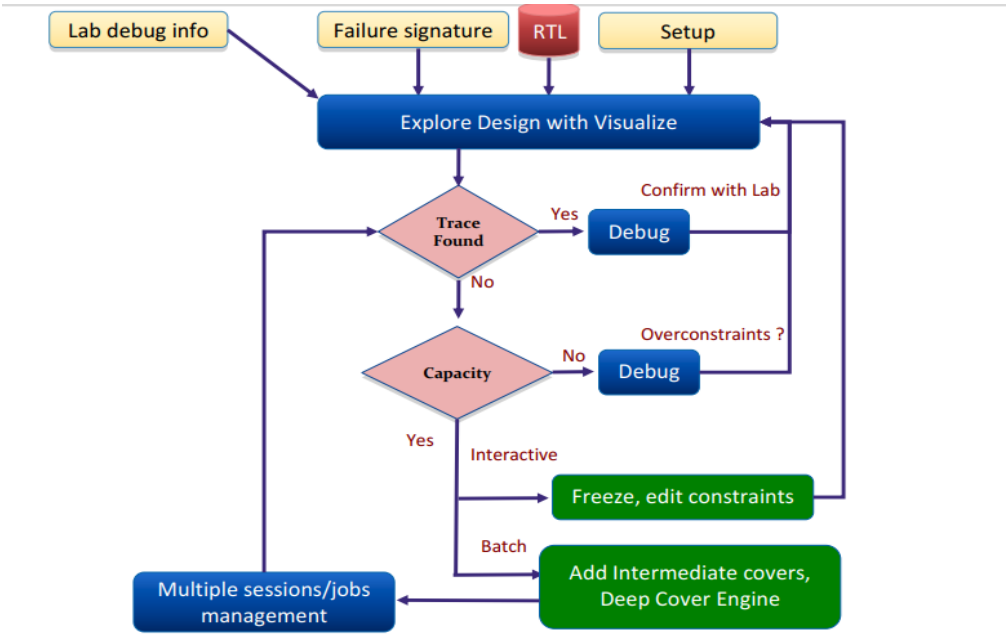

Figure 1 Flowchart for Post-silicon Debug.

III. NON-CONVENTIONAL APPLICATIONS USING FORMAL

With the increasing use of sequential optimizations during logic synthesis, sequential equivalence checking (SEC) has become an important practical verification problem. SEC might employ symbolic algorithms, based on binary decision diagrams (BDD) to traverse the state space or any optimized methodology for the specific state space traversal to check for the equivalence of two circuits. On the other hand, the equivalence problem could also be mapped to a model checking problem, where a set of properties define the equivalence between the two circuits. The basic theory behind the RTL-RTL is that when we inject two functionally equivalent RTL models with same inputs, we expect the outputs to remain the same. The tool requires us to give two RTL models. If the input names are same, then it drives the same values on both the models and an assertion is added at the output to check for

* Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

equivalence. This methodology has found widespread applications in areas such as clock gating verification, timing fixes [4] etc. We employed this methodology to attune to our validation environment which lead us to the application of sequential equivalence checking in new areas.

A. Hierarchical RTL-RTL for SoC designs and derivative products

When there are multiple derivative projects forked off from a parent project, there are certain features which are coded both in the parent project and the derivative project requires the need to validate if the feature is not affecting the unintended partitions. To ensure the functional correctness of the change, many tests are run which has an added impact on the compute cost on each test. This effort could be greatly reduced if we have equivalence checked from the top level to the design till the leaf level of two projects. Conventional RTL –RTL would hit convergence issues if the design size is more than 5 million. We worked with vendor to enhance the equivalence check capabilities at chip level to automatically partition the design depending on the hierarchy defined at the chip level till the leaf level units. This compares the two designs till the leaf level and equivalence is proven. We have enhanced the automation further to compare the partition if there is a change in the name of the partition with the lower level units being the same. This has helped us to achieve the results nearly 50X faster to current validation environment.

B. SCHMOO RTL-RTL

Intel GPU (GT) has media capabilities which handle video encoding, decoding and transcoding process. The traditional way of validating media hardware algorithms is using a golden C model as reference and control logic in these blocks is validated using testbench techniques to create different timing scenarios. In any VLSI design it is hard to expose some of the control logic bugs using traditional techniques as we are limited by either test content, testbench infrastructure or coverage and these bugs end up showing in silicon, which is very expensive. We use the terminology schmoo to refer to different interface delays and backpressures created in testbench to the DUT.

In order to enable the requirement for control logic verification, we needed to create an environment where we do a transactional equivalence i.e., irrespective of the time the data is injected into the RTL, we expect the RTL to generate the same output. This transactional verification is for the data and the associated valid signal only. Since the tool maps all the signals, we had to specifically unmap just the data and its associated valid signals and add FIFOs both at the input (INFIFO) and at the output (OUT FIFO) as shown in figure 2. The advantage of un-mapping the valid signals is that, the tool is free to drive them independently which is needed to create delays.

The tool is programmed to drive back-back transactions into spec RTL and essentially has no back pressure or delay. Using INFIFO, we have introduced delays at the input of the IMPL model. This is done to keep the input data between SPEC and IMP the same, but data arrives at two different time-stamps in the SPEC and IMP models. The write and read equations for the INFIFO are as follows:

INFIFO.WRITE= SPEC.INCOMING_DATA_VALID && ! INFIFO.FULL

INFIFO.READ = IMPL.INCOMING_DATA_VALID && !INFIFO.EMPTY

We added an OUT FIFO since the SPEC RTL will generate outputs faster than the IMPL RTL and we have to hold the spec data until the corresponding transaction from IMPL model comes out. The OUT FIFO is written when there is valid data at the SPEC RTL output and this data is compared against the IMPL RTL output when a corresponding data is present.

The write and read equations for the OUT FIFO are as follows:

OUT FIFO.WRITE= SPEC.OUTGOING_DATA_VALID && ! OUTFIFO.FULL

OUT FIFO.READ = IMPL.OUTGOING_DATA_VALID && !OUTFIFO.EMPTY

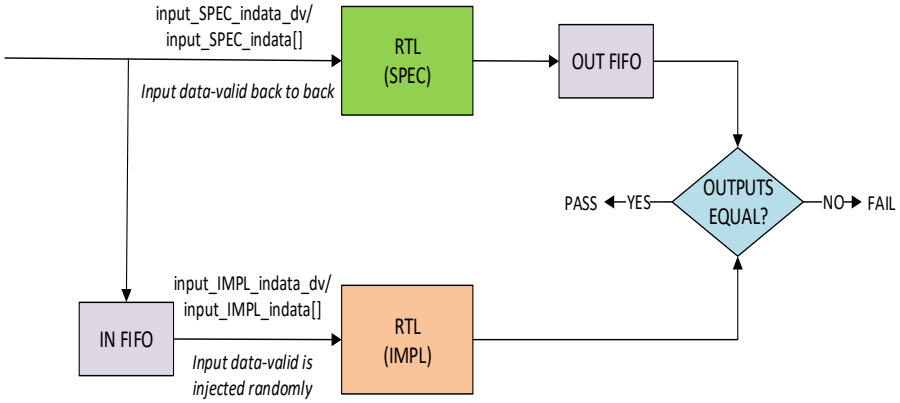

Figure 2. Schmoo RTL-RTL.

C. Connectivity Verification

The Formal Connectivity verifies RTL connections at the block, unit, and chip level for IP integration. It takes connectivity specification as input, automatically generate properties for each connection specified in the specification and exhaustively verifies connectivity. Formal Connectivity tools traverse the connectivity specification and automatically black-boxes the modules which are not required, thereby reducing the complexity for formal. Connectivity tools also have the capability to extract connectivity specification from the design and is termed as reverse connectivity flow.

Using normal SoC connectivity verification flow at top chip level, a lot of the debug time is wasted in debugging the failures related to errors in connection misses. Formal Connectivity is helpful in exhaustive verification of connections, but creating connectivity specification is challenge, which we smartly solved by extracting connectivity specification using reverse connectivity from monitors connected at leaf levels and translated these connections to top level and verified connectivity at chip level, thereby reducing down the connectivity debug time to less than 1% of debug time, proving a typical example of shift left in validation cycle. By using this automation, we were able to quickly find out critical issues in chip level connectivity much faster than traditional flow.

IV. RESULTS

Post-silicon debug using formal reduced the debug time by 60% as compared to simulation approach. The properties used in reproducing the issue, were also utilized in verifying the bug-fix, provided higher confidence due to exhaustive nature of formal.

In the course of project, there were multiple derivatives which were forked off from the main project. We applied the hierarchical RTL-RTL between the main model Vs forked off model. In addition to the models, the tool was provided with high level partition information so that it could effectively do the equivalence. Taking these as the inputs, the tool was able to do a hierarchical equivalence between the two models by internally partitioning the 700 million gate design into close to 7000+ partitions**.** The equivalence was completed in 8hrs compared to the DV regression time of 125 hours. The tool was able to point out many errors such as signal mismatches, incorrect porting of features etc.

We enabled Schmoo RTL-RTL on DUTs which were already reasonably validated using traditional approaches and this new methodology exposed additional 3 bugs, one of which was artificially introduced. This proved the robustness of the methodology and is made POR in current project and planned to be used for other in upcoming projects. The amount of time needed to expose these bugs is much faster compared to traditional approach.

Using normal SoC connectivity verification flow at GT level, 8% of the debug time was wasted in debugging the failures related to missed connections. Using the connectivity specification extracted at leaf level we were able to automate the complete flow and reduce down the connectivity debug to less than 1% of debug time.

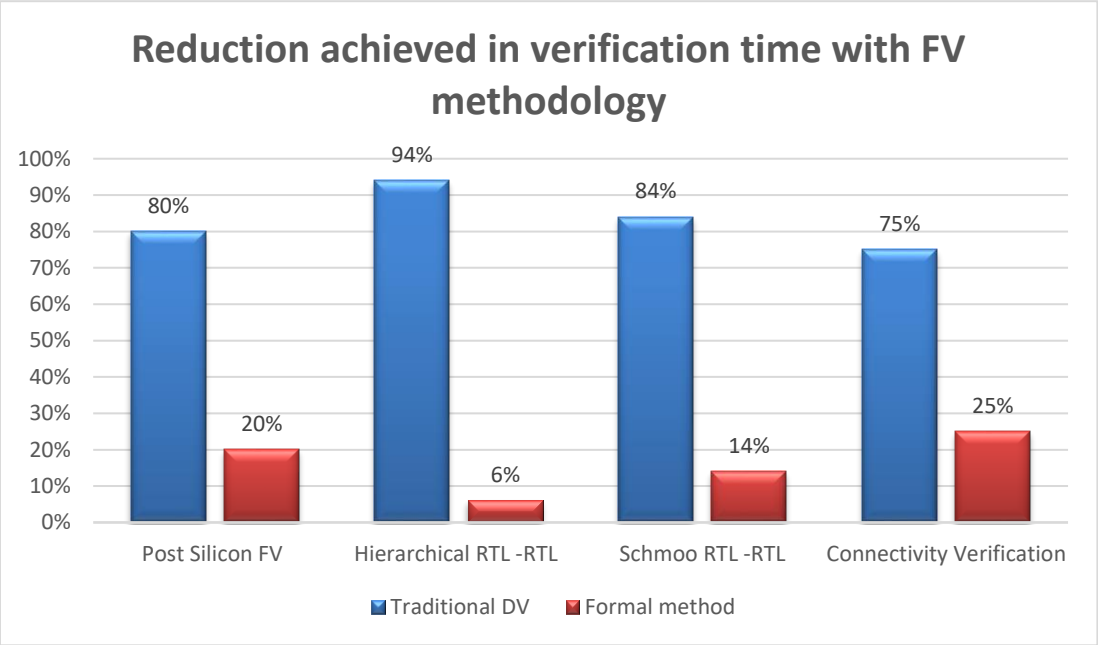

Figure 3 Results

ACKNOWLEDGMENT

We would like to thank our management Ashok Natarajan, Fred Steinberg, Ari Rauch for supporting us in driving the new methodologies towards our drive for best in class. We would also like to thank all the designers who worked with us closely and helped us succeed in achieving this. We are also thankful to the vendors who quickly reverted with working solutions on RTL-RTL. Last but not the least, we would like to thank Erik Seligman and the whole formal community for sharing their insights that helped in coarse correcting the methodologies.

REFERENCES

[1] [1] C. Pixley, "A theory and implementation of sequential hardware equivalence", IEEE Transactions on Computer-Aided Design, Dec. 1992, pp. 1469-1478.

[2] [2] Z. Khasidashvili, M. Skaba, D. Kaiss and Z. Hanna, "Theoretical Framework for Compositional Sequential Hardware Equivalence in Presence of Design Constraints", ICCAD 2004, pp. 58-65.

[3] [3] “Formal Verification: An Essential Toolkit for Modern VLSI Design” by Erik Seligman, Tom Schubert, M V Achutha Kiran Kumar, Elsevier Publications, 2016.

[4] [4] M AchuthaKiranKumar V, Aarti Gupta, Bindumadhava S S, "RTL2RTL Formal Equivalence: Boosting the Design Confidence" DVCon 2014.

[5] [5] Aditi, Laurent et al., In case of emergency, call 1-800-FORMAL, JUG 2018

[6] [6] https://blogs.mentor.com/verificationhorizons/?s=where+verification+engineer+spent+time