International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(Review Article)

Firmware based Bug Finding Mechanisms in Pre-Silicon Environment

Ankit Chandankhede *

Senior Member of Technical Staff, Advanced Micro Devices, Inc, Texas, USA

International Journal of Science and Research Archive, 2020, 01(01), 133-140

Publication history: Received on 10 October 2020; revised on 23 November 2020; accepted on 26 November 2020

Article DOI: https://doi.org/10.30574/ijsra.2020.1.1.0030

Abstract

In continuous progression of Moore's Law, modern architectures in CPU , GPU and custom chips have introduced different features based on the modern application of datacenters, gaming consoles and edge computing. This has significantly increased the complexity of design space and exponentially increased the verification space. With increasing competition, it is pivotal to reduce the verification cycles as well as meet the bug finding techniques tap-out milestone. Quality of the verification can be easily achieved with the bug finding techniques proposed in this paper. These techniques not only facilitate to finding bugs at early stage of design but also provide the quality metric to sign off the closing milestones. Further proposed method provides the evaluation and confidence in health of design based on the market centric workloads.

Keywords: Firmware; Bug Finding Mechanisms; Pre-Silicon Environment; Bug finding techniques; Bug finding techniques

1 Introduction

With increasing complexity of architecture and design and competitive silicon industry, challenges in verification processes are equally increased. Modern architectures are catering different application needs with multi-core , multi-die, chiplet , low power and custom chips and thus same verification process also needs to evolve based on the architectures as designs includes complex data paths, state machines, concurrency, complex mathematic instructions and branching which poses increased risk of bus, making traditional testing approaches inadequate.

It is crucial to deploy robust verification methodology to detect and fix bugs at early stage of development cycle. Traditionally presilicon verification includes formal verification, simulation based verification, emulation[9] and gate level simulation with ultimate goal of achieving a zero buy A0 tapeout and releasing product to the market at earliest[1][2].

Although Traditional verification techniques are robust enough but often lacks the real world application and user specific workload and hence leaves a gap for potential bugs.

This paper underscores different traditional methods being deployed and their advantages and further showcases the importance of firmware-based verification to find design bugs through real world workloads.

* Corresponding author: Ankit Chandankhede

2 Traditional Techniques Deployed

2.1 Alignment of Teams Across Organizations

Effective alignment across the teams is critical in order to meet different design milestones. The alignment involves staging of design features across units, verification milestones based on these design milestones through checklist item. In complex designs, particularly those that introduce new features or architecture, there may be cases where the existing DUTs are not sufficient to cover the entire design space. In such cases, additional DUTs should be created to ensure that new features IP specific features are verified and tested thoroughly at unit or cluster level as features are well contained and localized within certain part of the design and architecture. Whereas megafeatures are spread across the architecture and involves interoperability of the protocol interactions and algorithm and hence it is essential to test such features at higher levels of DUTs. For instance, compute or shader specific computation rely on the result of previous stage of the pipeline or shaders. [13]

2.1.1 Divide and Conquer

IP-Centric Feature Stress Testing

The architectural expansion of GPUs, CPUs and custom chip design for AI accelerators has grown significantly and dividing such mammoth of design space into smaller IP blocks, cluster and multisubsystm is essential from design and verification point of view.

Majority of the features are concentrated within specific region of design and hence verification of such design changes at unit or cluster level can regressively verify the functionality and performance. Such design often includes branch modelling, arithmetic computation, caching of the data or instruction, memory controllers[3]. Such part of the design can be localized within units.

Formal verification has been instrumental to thoroughly verify such design space as it uses mathematical models to exhaustively check for design errors. Robustness of formal verification technique allow to find the latent bugs early in design cycle and thus deploying formal verification complements the traditional verification environments based on UVM, C++ or OVM as well as reduces dependency. Further assertions from formal are reusable in the traditional verification environment which further allows design checks in sanity of submission as well as regressions. However, formal verification techniques is not scalable to bigger design space and recent trend and research to fit design in formal verification environments has made good progress[1].

2.1.2 Creating Proper Boundaries for DUTs (Device Under Test)

Dividing the architecture into multiple verification Device Under Test (DUT) allows faster and efficient verification but it pivotal to form clean boundaries of the DUTs which not only allows faster simulation but also contain features effectively, reduces chance of interoperability of design at higher level of verification abstractions, and effective reuse of the verification components. Further it essential to consider the scalability of DUTs for different configurations and Stock Keeping Units (SKUs)[4].

Interoperability

Complex architectures involve different protocols and pipelined decision making. Such pipeline decision making often relies on the result of previous pipeline stage. This result in numerous interoperability scenarios among different blocks of the design units, subsystem and pipelines.

Such cross IP block and pipeline interactions must be thoroughly tested to insure the integration of design. Stress scenario of interoperability includes following

- Stress testing of credit-hold conditions

This can be achieved through following techniques

  - Pushing more traffic in the design
  - Synthetically holding one end of the design to not allow design to process the workload and hence accumulating the traffic in FIFOs, cache/storage design components which creates back pressuring scenarios on the ingress of the design due to limited credits.

  - Introducing synthetic holds between design – Interaction among the design often are controlled by credit, acknowledgment, hold and ready signals. It is easier to either use force and release mechanism to hold or ready signals of the interfaces of the design and hence allowing back pressuring in very few cycles.
- Dependency of a thread with another
  - In GPUs, a workload is divided into multiple threads and most of the thread poses dependency on each other which may result in live lock or deadlock scenario is the dependencies are not handled properly and hence creating dependencies among threads in testing would help to find such bugs.
- Barrier and synchronization of threads
  - Among threads often barrier and synchronization are required to compute next stage of the workload and thus intervening among such threads while synchronization allows to find the bugs
- Protocol interaction between units such as AXI, AMBA protocols
  - Protocols compliant designs functionality can be tested using negative scenario as well as credit exhaustions.
- Throttling of the design performance configurations
  - For performance improvement, design operates different type of workload/messages to prioritize processing specific workloads and hence throttling the workload with different values may result in back pressuring of specific workload and helps to find the bugs.
- Design state for different workloads
  - Specific type of the workload can be processed differently which is configured through register programming or state cacheline updates. For instance, GPU shaders can be processed to higher level of details such as tessellation levels, depth of scaling, pixel shading and color shading. Crossing different states at different shading can help to find cross shaders architectural bugs.
- Flooding the pipeline with traffic on emulation platform:
  - Often stress case scenarios are manifested through bigger workload which can be run easily on emulation platform as simulation platform may not be able to run such traffic within short period of time and consumes lot of verification cycles. [9]

Different Configurations

Speed of the simulation is definitely a major factor in pre silicon verification stage as it not only affects the compute resources required during the test runs and regression but also feature bring up, debug and bug fix testing[10]. This can be achieved by defining the DUT boundaries. Often the architecture or System on chip (SOC) caters to different applications and requirements which is achieved through SKUs and configurations. While presilicon verification is performed at different levels of abstractions and DUTs, DUTs boundaries should be scalable to such configurations and SKUs, thus allowing the pre silicon verification of different configurations for different products as these configurations and handling feature enablement can be easily controlled through testing environments hooks.

Feature Confinement within DUTs:

Feature Confinement within certain part of architecture helps to localize the design changes required and also verification of such features as it does not spread across different Units or subsystems. Thereby reducing the verification scope, risk of design bugs due to interoperability and simplifies the integration of features from firmware and design point of view. For example, in a GPU, the rendering pipeline might be isolated as a separate DUT from the memory subsystem, allowing tests to be conducted independently on each pipeline and ensuring bugs are isolated.

Creating Additional DUTs

Architectures across different generations of product changes dramatically due to application requirements of product as well as performance improvements in existing architecture. Often such changes involves newer features and may introduce new units/IPs and thus expanding the verification space. To effectively cover such changes in architecture, existing DUT boundaries can be expanded or changed or introduce the new DUTs. For example, in GPU, newer shading feature was introduced called “Ray Tracing” which necessitates the introduction of newer pipeline and design Ips and thus necessitated to not only update boundaries of existing DUTs but also introduce new DUT to effectively cover Ray tracing features as features associated is spread across the GPU pipeline [14].

2.2 Checkers and Scoreboard Scalability

Bug hunting heavily rely on not only the quality of the test but also the checkers and scoreboards which are pivotal in checking the design functional correctness. Often these behavior checking components are designed based on the requirements of the IP or cluster and hence scalability of such component while integrating at higher level of verification abstraction or DUT makes it hard to reusable. Recent verification process has taken significant steps for scalability and reusability of such components including the assertion from formal verification as well[12]. Thus allows better checking process has elevated the verification quality at higher DUTs and reduced redundant work to implement high quality checks,

In a multi-IP, multi-subsystem environment, scalability of checkers and scoreboards is essential. Checkers are used to verify functional correctness, while scoreboards track the status of different elements within the design to ensure that the correct data is being processed. As designs grow in complexity, verification teams must design checkers and scoreboards that can be reused across various DUTs and are capable of scaling to higher levels of abstraction[6].

By leveraging checkers and scoreboards from individual IP blocks, verification engineers can avoid redundant work and ensure consistency across the verification process. The key benefit of importing checkers and scoreboards is that they enable easier convergence of test quality across the design hierarchy. This ensures that as the design transitions from individual IP verification to full-system verification, the quality of the tests does not degrade.

2.3 Test Sequence Scalability

Tests and its associated sequences are integral and centered part of the DUTs and test planning. Traditionally nature of DUTs and structure of testbench has been a cause of creating testcases specifically for a particular DUTs for testing same feature. This offers an opportunity of creating a cohesive environment of the DUTs and likewise the test sequence which can be reused across the DUTs[11].

Such scalability of Test sequence provides opportunities for reducing the engineering power required for test creation, check the status of such testcases at different level of verification abstractions which can help to pinpoint the roadblocks of design for health of particular feature and configuration further allows to verify different SKUs[7].

2.4 Test Sequence Complexity

Complexity of architecture poses unique challenges to stress the design space and cover the case scenarios. In order to create stress case scenarios , verification engineers can cross different pipelines (shaders in GPUs), features, configurations and size of the workload (that will ensure multiple threads and thread dependencies while processing the shaders). Often with such complex test sequences, stress inducing techniques are introduced on top of such testcases to find corner case and back pressuring scenarios[8]. In GPUs, live lock and deadlocks scenarios are spread across due to dependencies of threads, shaders and pipelines. Dependencies of the threads can be managed through the instructions and thus introducing the levels of complexity to every testcase.

2.5 Functional and Code Coverage

Coverage is a quality metric of the verification which provides the progress of the design milestones, test plan progression and final metric while signing off the design. Functional and code coverages are two types of coverages. While functional coverage ensures test planning is covered and design features are exercised , code coverage provides insights in testing of different part of design through toggle, fsm , branch, and line coverages[5].

Regressing different testcases allows to provide the final coverage through passing testcases. Analyzing coverage provides analysis of missing testcases or holes in testplans or intent of testcases and thus provides in depth understanding of modification required to test sequences or testplanning.

3 Necessity of current proposal

Although the above-mentioned techniques are covering the space of the design, it still lacks the real workload from users perspective, involves creating different testcases and scalability of testcases which involves curated engineering efforts. Real workload on any design is exercised and submitted through firmware. Running firmware usually has been considered a post silicon technique to test the design quality and integration of different internal and external IPs. For instance, running a game on GPUs creates different frames. Directly running such workload in post silicon poses a definite risk of expensive design issues, bug fixes and firmware based fixes for overcoming the design issues which can degrade the performance of the products or turning off specific features. Methodology proposes in this paper provides a definite solution to use firmware based workload to be used in pre silicon verification platform.

3.1 Proposed Methodology and Implementation

Firmware based methodology cover the gap between synthetic testcase and real workloads. Firmware is bridge between a software and hardware interactions and thus breakdown the task on hand in system understandable commands and instructions (including kernels). Thus provide a granular understanding to hardware for processing the workload. This provides an opportunity to use firmware's ability to bring the real workload to presilicon environment. In GPUs, while a game is played on gaming console, firmware breakdown the workload by deciding the task into smaller frames and frames into commands and instructions as shown in figure 1.

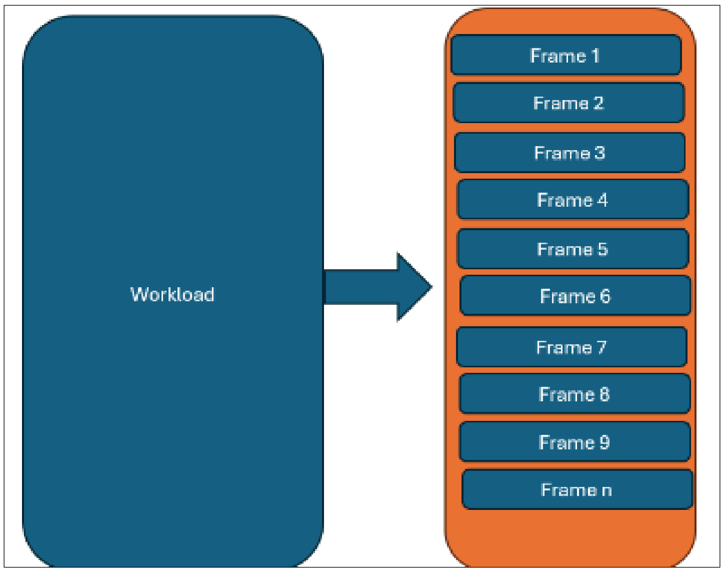

Figure 1 In GPUs, while a game is played on gaming console, firmware breakdown the workload by deciding the task into smaller frames and frames into commands and instructions

This frame can further be decided based on the commands and instructions which can be used by pre silicon simulation test sequences to drive the interface of DUTs as shown in below figure 2.

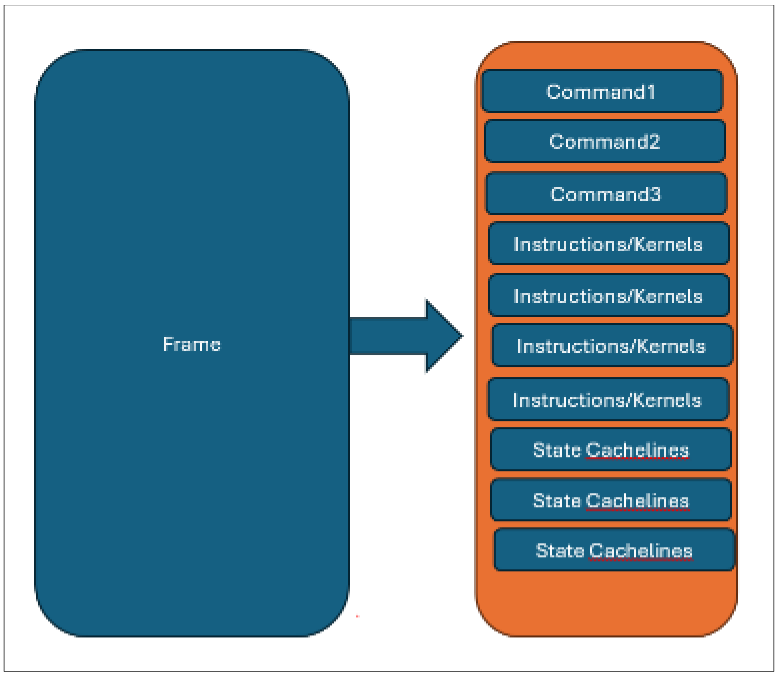

Figure 2 Based on the commands and instructions which can be used by pre silicon simulation test sequences to drive the interface of DUTs

Perl script is used to convert commands, instructions/kernels, state cachelines and final expected dataframes into the cacheline in bit format which can be loaded into the design. Doorbell are initiated using UVM sequences with pointers of the cacheline to load the configuration and workload (3D or Compute command) to process the workloads.

3.1.1 Command

For confidentiality reasons following cacheline are not revealed however hypothetical command looks like below which is generated by perl.

<Rending configurations><Threads Group per Command ><Threads per ThreadGroup><Command type (Compute/3D)><flush type>

3.1.2 Instructions

<Arithmetic command ><Operands cacheline pointers ><Thread Dependency><Valid Instruction bit>

StateCacheline:

<Shader type><Pipeline stage><Levels of surface rendering><Surface state rendering type><Statecacheline valid bit>

----

4 Results

4.1 Bugs founds

Since this firmware-based methodology is able to close the gap between the realtime workload and synthetic content, this methodology was able to find bugs in designs and architectural definitions and its categorization is as follows.

4.2 Architectural bugs

Architectural definition was missing the pipeline threads per WGs with respect to the fifo depths while 3D and compute workload . Further, implementation of flushes and invalidation of across 3D and compute pipeline were flushing and invalidating the caches entirely although the other workload were still in progress which accounted to be performance issue.

4.3 Interoperability bugs

Interoperability bugs are defined as bug that are related to inter communication between two units. This methodology was able to stress the interfaces between different Units and pipelines and found bugs in the credit flow mechanisms as well as timing sensitive bugs in state machines due to time specific events such as preemption or change of priority ordering of workloads.

4.4 Gaps found in the testplan

Although the synthetic content was covering the planned functional and code coverage but was missing the scenarios of crossing different features, time critical events crossing with different features , thread dependencies and commands.

4.5 Readily available testcases

Since real world workload is divided up using firmware automatically , testcases are generated with minimal effort and are scalable to different level of abstraction of verification DUTs. This reduces redundant effort on creating and maintain new testcases. These testcases offers complementary to synthetic test content. Furthermore, it provides confidence on design for market readiness.

4.6 Drawbacks of Firmware based Methodology

This method does not scale well on newer features as firmware development cycle follows the pre silicon environment, thus framework is not ready for newer feature until later part of the project and hence relying on the firmware, does not allow this methodology to newer features.

4.7 Improvements

Current implementation is only able to scale down to certain extent which still runs for hours on pre silicon simulation and emulation environment. Further breaking down of the workload may not be possible as it needs to be broken down at clear boundary . Thus creating smaller standalone cases is not viable option with this methodology. Future improvement to this methodology can be scaled to replace the synthetic content for legacy features.

5 Conclusion

Pre-Silicon verification is critical in testing of hardware design and product to market cycles as it provides confidence in design health, however traditional verification techniques heavily rely on synthetic content and hence there can be a real gap between understanding of feature and feature implementations in real workload, allowing bugs to escape. Firmware based verification methodology in pre silicon verification covers this gap and provides testcases based on real workload which allows the stress the design as well as cover majority of the feature crossings. This methodology has been instrumental in finding interoperability and architectural bugs.

Compliance with ethical standards

Disclosure of conflict of interest

If two or more authors have contributed in the manuscript, the conflict of interest statement must be inserted here.

References

[1] D. Stoffel, "Formal Verification of Systems-on-Chip - Industrial Experiences and Scientific Perspectives," 2009 20th International Workshop on Database and Expert Systems Application, Linz, Austria, 2009, pp. 3-3, doi: 10.1109/DEXA.2009.83.

[2] Debapriya Chatterjee, Andrew Deorio, and Valeria Bertacco. 2011. Gate-Level Simulation with GPU Computing. ACM Trans. Des. Autom. Electron. Syst. 16, 3, Article 30 (June 2011), 26 pages. https://doi.org/10.1145/1970353.1970363

[3] Ilya Wagner and Valeria Bertacco. 2008. MCjammer: adaptive verification for multi-core designs. In Proceedings of the conference on Design, automation and test in Europe (DATE '08). Association for Computing Machinery, New York, NY, USA, 670–675. https://doi.org/10.1145/1403375.1403539

[4] Karthik Ganesan1 , Florian Lonsing1 , Srinivasa Shashank Nuthakki1 , Eshan Singh1 , Mohammad Rahmani Fadiheh2 , Wolfgang Kunz2 , Dominik Stoffel2 , Clark Barrett1 , and Subhasish Mitra1 , " Effective Pre-Silicon Verification of Processor Cores by Breaking the Bounds of Symbolic Quick Error Detection," https://arxiv.org/pdf/2106.10392

[5] Z. Ying, W. Ning, K. Xiazhi and G. Fen, "A Coverage-Driven Verification Platform for Evaluating NoC Performance and Test Structure," 2012 26th International Conference on Advanced Information Networking and Applications Workshops, Fukuoka, Japan, 2012, pp. 261-265, doi: 10.1109/WAINA.2012.104.

[6] Y. -N. Yun, J. -B. Kim, N. -D. Kim and B. Min, "Beyond UVM for practical SoC verification," 2011 International SoC Design Conference, Jeju, Korea (South), 2011, pp. 158-162, doi: 10.1109/ISOCC.2011.6138671. keywords: {Verification;SoC;SystemVerilog;UVM;testbench},

[7] S. Banik, S. Roy and B. Sen, "Test Configuration Generation for Different FPGA Architectures for Application Independent Testing," 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, India, 2019, pp. 395-400, doi: 10.1109/VLSID.2019.00086.

[8] F. Plasencia-Balabarca, E. Mitacc-Meza, M. Raffo-Jara and C. Silva-Cárdenas, "A Flexible UVM-Based Verification Framework Reusable with Avalon, AHB, AXI and Wishbone Bus Interfaces for an AES Encryption Module," 2019 IEEE Latin American Test Symposium (LATS), Santiago, Chile, 2019, pp. 1-4, doi: 10.1109/LATW.2019.8704549.

[9] S. S. Shankar and J. S. Shankar, "Synthesizable verification IP to stress test system-on-chip emulation and prototyping platforms," 2011 International Symposium on Integrated Circuits, Singapore, 2011, pp. 609-612, doi: 10.1109/ISICir.2011.6131936.

[10] Keith Campbell, Leon He, Liwei Yang, Swathi Gurumani, Kyle Rupnow, Deming Chen, " Debugging and verifying SoC designs through effective cross-layer hardware-software co-simulation," DAC '16: Proceedings of the 53rd Annual Design Automation Conference Article No.: 7, Pages 1 – 6 https://doi.org/10.1145/2897937.289800

[11] J. R. Biddle, "Optimizing Bug Detection through Test Sequence Reusability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 9, pp. 1578–1586, 2017.

[12] Hazra, Aritra & Banerjee, Ansuman & Mitra, Srobona & Dasgupta, Pallab & Chakrabarti, Partha & Mohan, Chunduri. (2008). Cohesive Coverage Management for Simulation and Formal Property Verification. VLSI, IEEE Computer Society Annual Symposium on. 251-256. 10.1109/ISVLSI.2008.53.

[13] Pathak, S.K.. (2013). 3D Graphics Hardware and Role of Shaders. 7.07-7.07. 10.1049/cp.2013.2340.

[14] Clua, Esteban & Montenegro, Anselmo & Pagliosa, Paulo & Sabino, Thales & Andrade, Paulo & Lattari, Lucas. (2011). Efficient Use of In-Game Ray-Tracing Techniques.