International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Acceleration of test sequences for graphics architecture testing at subsystem and SoC Level

Ankit Chandankhede *

Senior Member of Technical Staff, Advanced Micro Device Inc., USA.

International Journal of Science and Research Archive, 2020, 01(01), 155-164

Publication history: Received on 28 September 2020; revised on 23 December 2020; accepted on 26 December 2020

Article DOI: https://doi.org/10.30574/ijsra.2020.1.1.0023

Abstract

Graphics Processing Units are known for parallel computing and rendering process and have complex architecture. Compute and rendering process involves complex functions and shaders which introduces challenges in design as well as verification through the design development cycle. Test suite used for verifying such complex design runs longer and hence consumes magnitudes of turnaround time to run a test, retest any bug fixes, as well as running regression. Most of these test suites uses similar configuration and initialization of the design which often contributes to 30-40% of test simulation cycles which usually are stable after certain stage of design. This paper proposes methodology to save the state of design after the configuration and initialization phase which can be restored before the run phase for different workloads, thus allows to start the testcase from rendering or compute workload. This improves the efficiency of simulation test as the proposed method skips through the initialization and configuration cycles and saves about 30-40% simulation time at Multsubsytem and SOC.

Keywords: Graphics Processing Unit (GPU); Simulation Time; Functional Simulation Database (FSDB); Pre-Silicon Verification; Universal Verification Methodology (UVM)

1 Introduction

Graphics Processing Units (GPUs) are used for parallel computing and rendering processes to provide high performance, either gaming consoles or data centres. For rendering and computing operations, GPUs have different shaders, like vertex, geometry, hull, tessellation, and pixel shaders. The second is that GPUs are used to perform complex functions like z scaling (masking) and colour rendering, for example's sake, but GPUs depend on sophisticated execution units capable of processing mathematical instructions. The verification of GPUs is a very complex model; when designers start considering designs under tests, such as SoC or multi-subsystem environments, they can become even more complex. At the SoC or subsystem level, the simulation of the basic test cases can take several hours in pre-silicon simulation platforms. The total cycles for the initialization phase take up approximately 50-70% of the total simulation time for small workloads. On the other hand, initialization takes 30—40% of the total time for larger workloads.

Over time, the design cycle matures, and the initialization process becomes more and more stable. Experts can reuse initialization between different workloads. Reusing initialization enables dramatically lower redundant simulation cycles, improving the overall testing efficiency, utilizing compute and engineering resources, and reducing pre-silicon verification cycles. Testers and write test cases for each pipeline and shader as a single unit, similarly to simplify test cases. The resulting increase in the number of test cases requires more computational resources. For this reason, the time taken to run such test cases needs to be reduced to make the testing process quicker. In pre-silicon verification, this study proposes a methodology based on the Universal verification methodology (UVM), a Perl testing environment for SoC, and a multi-subsystem DUT environment. Additionally, this methodology is made more efficient for retesting bug-fix cases within a test sequence via new bug forecasting.

* Corresponding author: Ankit Chandankhede.

The contributions of this study include an efficient method for reducing simulation time by reusing the stable initialization processes of multiple workloads. Since the method removes redundant initialization cycles, the pre-silicon verification testing can be done faster and with less resource consumption. In this approach, the stable design initialization execution is used as it matures to provide the resulting initialization usable in subsequent tests. It reduces the testing cycle by reducing its cycle and saves resources and time. This study aims to demonstrate that the use of this method at the Design Under Test (DUT) level, particularly in the case of System Chip (SoC) and Multi-Subsystem environments, will significantly reduce the time needed for verification and, therefore, better use of computational resources.

2 Traditional Methods to Improve the Efficiency

2.1 Reducing the Checkers and Monitors

Checkers and monitors are often used parallel with the design verification during pre-silicon simulation tests. However, working in parallel does not eliminate the overhead they incur, and the costs in terms of wall clock time can significantly impact the simulation's speed and overall efficiency. Reducing the number of checkers and monitors will cause the compile time and the runtime of tests to increase directly. The checkers and monitors that facilitate the ability of the design to function as intended help reduce their number, as far as is practical, to minimize the test overhead and increase the speed of the simulation cycles. Improvements in simulation efficiency are possible during the verification process by optimizing these elements (Ray & Hunt, 2009).

2.2 Disabling the Coverage Modules

Verification modules also have an important role in coverage because they cover all parts of the design. These modules measure code or functional coverage to show that the test process has been completed. However, collecting coverage during simulation may result in extra overhead and extend the overall simulation time. By disabling coverage collection for some tests, coverage collection can be turned off when running tests. This can be very useful to improve the overall test run test, especially if the coverage data is not directly needed for a given test run. Coverage is still considered an important metric for verification quality. However, it can be minimized when the simulation time is not a concern of the current testing phase by making the coverage optional. This method aids in the simplification of the simulation whilst ensuring adequate test integrity.

2.3 Reducing the BFMs Required

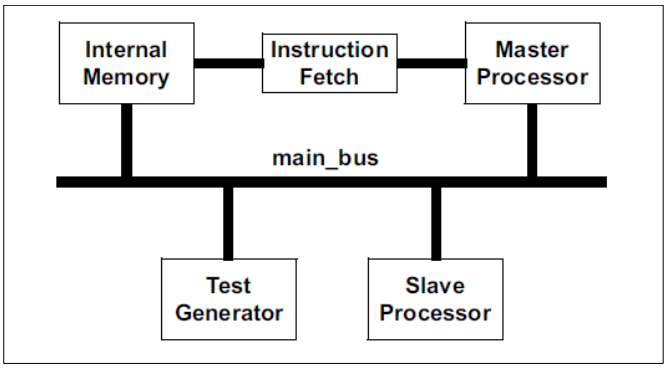

Figure 1 System Verilog for Design

The more complicated the simulation, the more Bus Functional Models (BFMs) are needed in simulation environments. To drive a design under test (DUT) and monitor its behavior, the design must operate with more BFMs, increasing the compile and simulation time for each. As a result, it is crucial to manage and minimize the number of BFMs in a simulation environment to optimize test cycles. Traditionally, techniques such as SystemVerilog-based interface merging can decrease the test environment size, reducing the compilation and simulation time. Through effective interface management, this simulation process becomes more streamlined, thus leading to faster overall performance by lessening resource consumption. These interface management techniques are vital to enable scalability for GPU architecture testing at subsystem and SoC levels (Kim et al., 2014).

2.4 Creating the Reduced or Skewed Configurations

Usually, the end product of the architecture instantiates the same tiles multiple times. Since they have similar functionality, these design tiles can be reduced logically to the least number of tiles - as few as a single tile or less. Simulation and compile time can be controlled as these smaller configurations are used. This method works because, in the case of design tiles, most of the tiles perform the same type of work or perform the same operation, so mobility of testing only a smaller configuration, for example, and one or a few tiles, can verify the entire system functionality. As a result, this reduces the configuration size, and the compile times take less time while the simulation overhead is equivalent or less. Thus, the testing process stays faster without losing coverage of the critical parts of the architecture.

3 The proposed acceleration methodology

3.1 Saving the State of Design after Configuration

The proposed method of speeding up the GPU test sequence relies on storing the design state after the first configuration and initialization phases, which account for a large percentage of simulation time. For complex designs such as GPUs, initialization sequences can significantly account for 50–70% of the total simulation time and are especially effective at the beginning of the design process. The system can circumvent redundant initialization steps by saving the design starting at a stable point and jumping right into TestingTesting at an initial phase, such as rendering or computing shaders (Moya et al., 2005). This permits efficient reduction of stable configurations without redundant simulations.

Once in this configuration phase, in which many of the components of the GPU design (shaders, execution units, and memory layout) are set up, it is possible to take a snapshot of the design state. This snapshot eliminates the need to run the initialization code in every new test case and consists of the configuration at that moment. It provides them the starting point to run subsequent tests concerning workloads without any kitchen sink. In addition to reducing the total simulation time, this methodology contributes to the conservation of engineering and computer resources utilized to repeat the configuration phase for both test cases.

3.2 Use of FSDB to Optimize TestingTesting

Within this methodology, the Functional Simulation Database (FSDB) is key to optimizing the testing process. It permits the restoration of the design state in various stages of the simulation, especially after the configuration stage. With the configuration and initialization information stored in FSDB, engineers can load this saved state and bypass configuration and initialization steps, taking advantage of what would usually be wasted time. It enables the engineer to start the main workload phase testing (shading shader or compute task) immediately without any repeated setup.

The design state is stored at FSDB, including various parameters, registers, and configurations that do not require re-execution. When saved, it can be restored at the start of the test phase for other workloads. This process drastically reduces the test cycle time and significantly decreases simulation times by reducing the reconfigurations needed during simulation. FSDB allows a more straightforward state restoration in scenarios where different workloads need the same initialization settings, speeding up and using the simulations more efficiently. This utilizes FSDB so that the verification process is cleaned up, and the testing cycle time is reduced as a result, helping both engineers and the overall tightening of the schedule.

4 Implementation

4.1 Quality Checks during Design Bring-Up

Early on in GPU system design, the quality and reliability of the system is paramount. Rigorous design testing verifies many of these criteria at the configuration and initialization phases. The first is stable bootup or configuration, meaning the system boots properly, and the configuration process finishes without errors. This is vital because any instability in the bootup or the configuration stage can adversely affect the testing function and the product.

They also need fewer changes in design during later stages of development. After the system is cooked to a more mature status, changes should be frequent to a degree of being minimized to maintain system integrity and validity of the test result. The later stages of the system are more stable and consistent, meaning the testing will be more efficient and effective.

Different skews in the booting or configuration process should be tested. However, the robustness requires proper handling of configuration variations, for example, under different loading conditions or hardware variations. Since the design state changes with different configurations, test engineers do not create the configuration sequences and save the design state in the Functional Simulation Database (FSDB). The system is then run with workloads that guarantee that the system is in the adequately booted or configured state, available to start processing, rendering, or computing tasks. The advantage of this method is that it makes the test process consistent, thus not wasting too much time for repeating initializations.

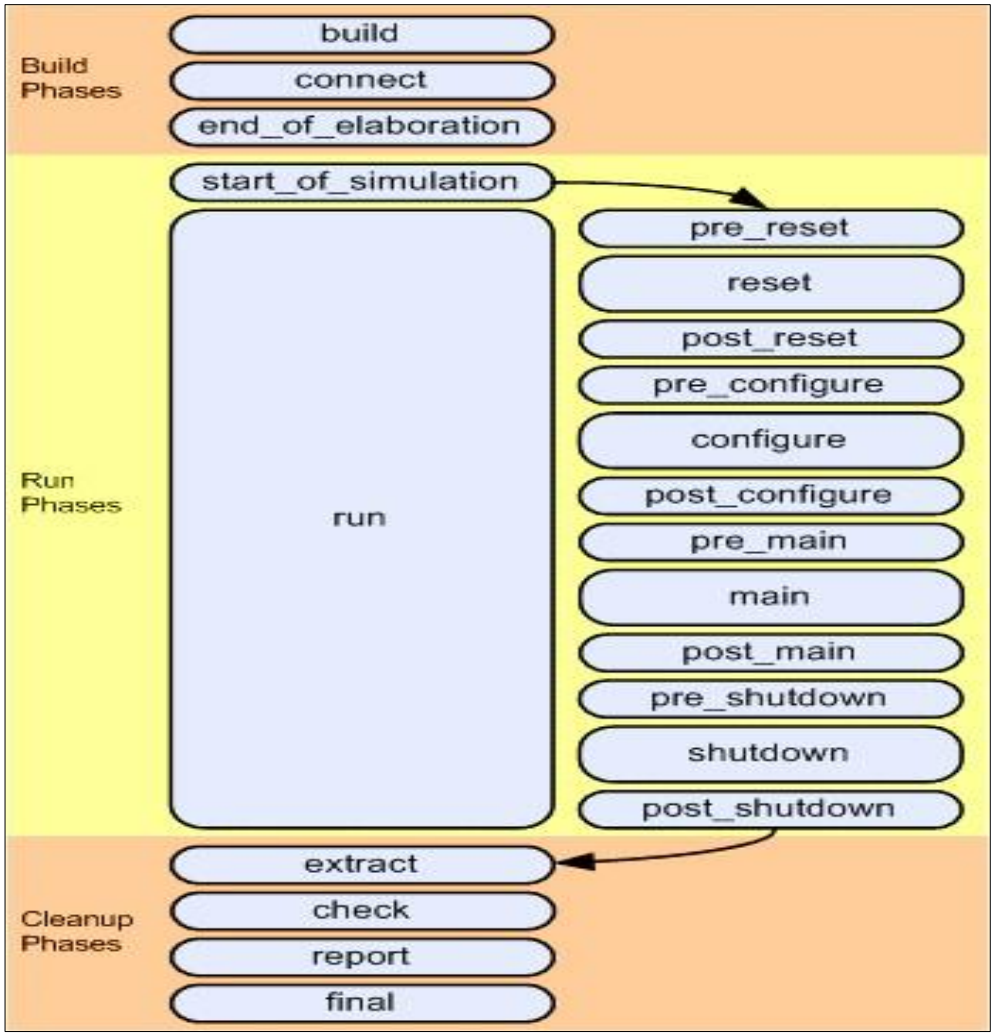

Figure 2 Overview of the UVM Phases

4.2 Using UVM for Efficient Test Execution

UVM serves as the key to improving GPU design verification efficiency. Running a test in the traditional verification methods requires starting from an initialization phase, which involves configuration sequences that have length and resource-expensive steps. The UVM environment has a separate configuration and main phase in the test flow (Moursi et al., 2018). With this methodology, the design state is borne in the FSDB file appropriate to the FSDB, and pre-main skips the bother of repeating the configuration. This leads to the test execution commencing directly from the primary phase and dramatically increases the speed of the test execution.

Additionally, if the test sequences are updated, the tester can start testing the updated sequences immediately without the need to recompile and re-run the complete initialization process. This is also achieved through Perl scripting the workloads (render or compute shaders), which enables flexibility in the test sequences without a full recompilation. Unlike conventional UVM methodologies, which require the test sequence to be changed, the proposed methodology ensures acceleration by skipping through these redundant steps, thus resulting in a significant decrease in total test time.

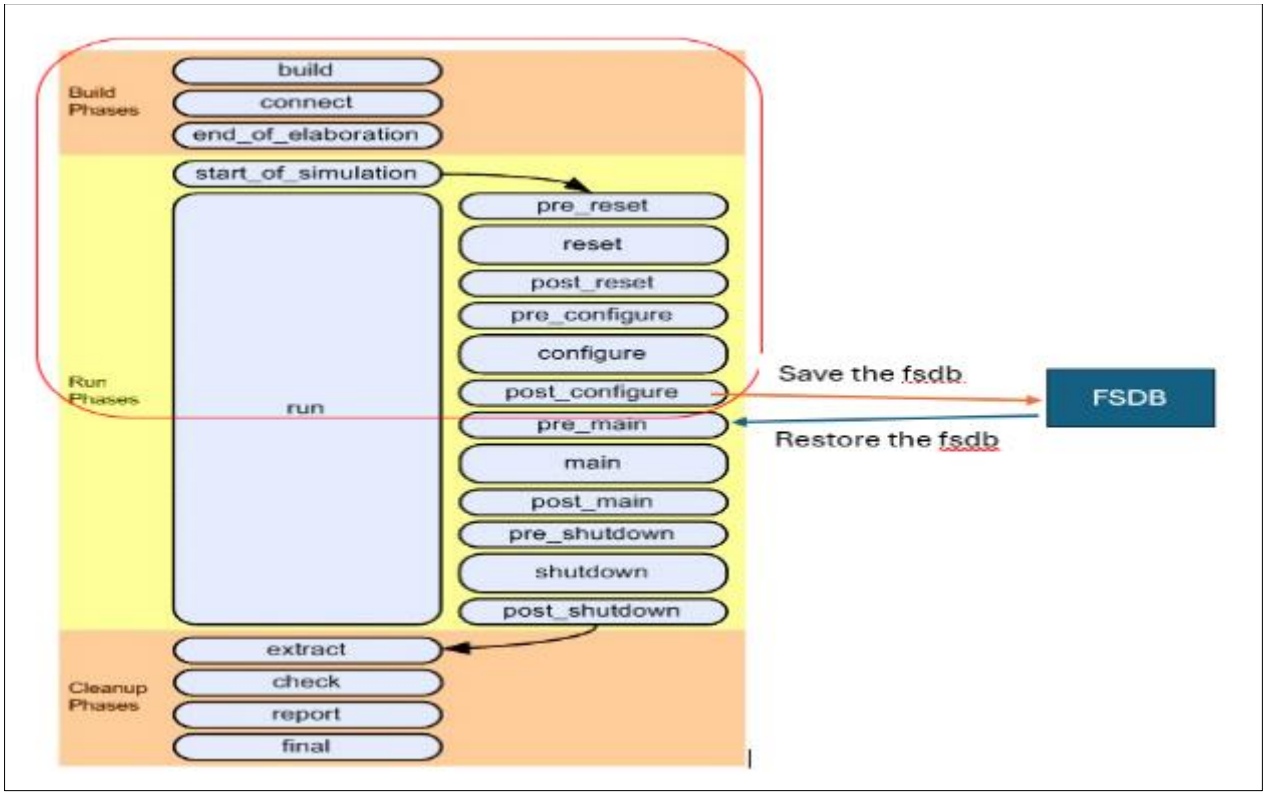

Figure 3 UVM Phases with FSDB Handling

However, this methodology has limitations. When there are changes in the BFMs, the environment, or the design code itself, it cannot load the FSDB. The FSDB would have to be recreated, prolonging the test cycle and requiring recompilation. To mitigate this, changes in the scoreboard and checker can be handled through C++ or Perl/Python scripting. Such scripting languages also provide checks without compilation in the System Verilog flow, improving the verification flow (Rieder et al., 2011).

5 Results and Discussion

5.1 Test Suite and Benchmarking

The proposed methodology was tested on several workloads that exploit different shaders and different combinations of shaders, both with and without the acceleration technique. Consequently, these workloads were scaled to support multiple rendering or compute commands, and the analysis was conducted on small and large-scale simulations.

Simulation time results show a gain of 30% to 50% reduction as a function of workload complexity. Even in the case of smaller workloads, there were significant improvements or even up to 70% reduction in the wall clock time required for the simulation test run (Simakov et al., 2018). Skipping the initialization phase and jumping into the rendering or compute workload phase is beneficial, as the simulation efficiency seen in this case is much better than otherwise. Since the methodology shortens redundant steps, it dramatically accelerates the test sequence, resulting in faster iteration and substantial time savings to achieve better pre-silicon verification cycles.

5.2 Resource Optimization

Besides the speed improvements, the proposed methodology showed important resource optimization. The ability to reduce simulation time decreased the need for computational resources and engineering power. By reusing the design state during the test execution phase, new test sequences were brought up faster without reinitializing the whole system at the expense.

Engineers also had more control over how they spent their resources because the simulation times were shorter. This allowed them to focus on the more important things during the verification process, such as hunting bugs and improving

the system's design. It increased hardware resource efficiency and engineering time shrinkage, which directly translates to shrinking the verification process as a whole.

5.3 Scalability and Debugging Efficiency

An advantage of one of the proposed methodologies was its scalability. It works very well, is applied to various existing test suites without degrading the performance of test cases, and brings features up faster while introducing a cleaner debugging interface. This allowed pre-silicon verification engineers to spend more time enhancing test cases because of the increase in simulation time. The ability to reuse design states permitted them to retest bug fixes much faster without heavy delays.

The methodology further improved the efficiency of the debugging. Engineers did not need to wait long simulation cycles to see and fix problems. Such a decrease in turnaround time is important in contexts such as pre-silicon verification, where time is critical, and consequently, faster identification and resolution of problems is achieved (Henschel et al., 2013). This methodology has provided the ability to retest updates within test sequences while maintaining minimum speedup of the debugging process to speed up the overall development process and improve the quality of the end product.

5.4 Functional and Code Coverage Convergence

It also has a nice benefit compared to existing solutions in that the proposed methodology converges faster for functional and code coverage than the existing ones. Typical test plans have been wholly inadequate because they do not include toggling of interface signals or functional coverage. To close these gaps, test sequences have to be updated and rerun to fill all the coverage bins, which can be a resource- and time-intensive process.

However, the proposed technique requires minimal changes to the test sequences that must be run in parallel. For the main test phase, one can avoid initializing the code, which increases the function and code coverage convergence. This permits the engineers to minimize the coverage gap without having to rerun large parts of the test sequence to improve coverage convergence efficiency. Doing this will save us from repeating many iterations and cut down the time by reducing the number of iterations needed and ensuring that all the functional and code coverage is provided.

6 Potential Areas of Improvement

6.1 Extension of the Base Initialization for Curated Workloads

One of the challenges in deploying and testing GPU architecture is the disparate intensity requirements of different workloads, which has motivations to force more memory or register space than others. A more refined approach to base initialization that can be utilized to optimize simulation times and resource usage would be needed. Creating multiple initialization databases for every type of workload, specifically those needing extra resources. With these workloads curated, test engineers can work around the redundant initialization steps and return the previous base initialization for similar tasks, leading to significant time savings for the simulation. This approach would also relieve the issue of repeated initialization and thereby conserve the resources more efficiently while ensuring an accurate test process. For example, testing base initialization to handle varying workload demands will reduce the testing overhead and allow for more realistic simulation scenarios reflecting real-world application performance.

6.2 UVM Component Updates

The third area for improvement is handling the UVM (Universal Verification Methodology) component update. The UVM framework does not provide smooth updating of component contents without total recompilation and may be resource-intensive when only small changes are required. This limitation can be overcome using Perl scripting to modify the relevant packets or parts of the simulation without re-running the theatre function. Using Perl scripting, the test engineers can introduce dynamic changes in the test sequences, which gives them more flexibility to manage and modify the test cases. Automating the update process with Perl makes it possible to avoid the need for a complete recompilation, which will accelerate test execution and shorten verification cycles.

7 Best practices

7.1 Optimizing Test Environments

For GPU testing, optimizing test environments is important to have a simulation at a fast speed, consuming fewer resources. Moreover, one of the key practices is to minimize the useless checkers and monitors in the system. Checkers and monitors help catch functional wrongness while intervening during simulation but are expensive to run in parallel. This reduces the run time of the overall test by minimizing the number of active checkers and monitors. For instance, in the early stages of simulation, where the design is stable, nonessential checkers can be eliminated, and this directly leads to reducing the wall clock time for the simulation process.

Disabling coverage modules is another effective strategy that not only achieves the goal of evaluating the test completeness but also eliminates additional delay in the simulation since they contribute much to the length of the simulation time. This effectively allows coverage collection to be turned off for some phases if coverage checking is not critical during these. However, it is necessary to ensure that this does not undermine the quality of the verification or the amount of testing being done. The number of BFMs necessary in the simulation environment can also be optimized for test execution (El-Ashry et al., 2019). The use of BFMs helps connect different components of the test environment and hence compensates for the simulation's increase in complexity. By reducing the number of BFMs without degrading the integrity of the test case, engineers can consolidate BFMs where possible and merge interfaces.

7.2 Leveraging FSDB for Design State Preservation

Design State Preservation gives one an essential technique to minimize simulation time using FSDB (Functional Simulation Database). Skipping the initial test run saves the design state after the initialization and configuration phase of testing so that engineers arrive at it for subsequent test runs. The advantage of using this technique is that it mainly helps with complex GPU architectures where the initialization phase accounts for a large part of the simulation time (Donaldson et al., 2017). Since engineer's store FSDBs during the computing or rendering workload phase, they can save the unnecessary time of starting from scratch.

In order to make good use of FSDB, the design must be stored in a stable state after its initial state. FSDB should be responsible for all the configurations and data points, and the design should be in the right state for executing the workload effectively. It quickens the testing process and decreases the redundant steps during initialization. Additionally, it allows for running the most recent test sequences without having to restart from the beginning of the tests, increasing the flexibility and efficiency of the tests and the resources.

8 A future consideration section

8.1 Extending the Methodology to Other Architectures

Although the proposed methodology on GPUs and the testing around them is described, the concept of this method can be expanded to other architectures such as System on Chips (SoCs) and AI architectures. This rule of reusing design states that clear redundant initializations can be applied to other hardware platforms with complicated configurations or intricate setup needs. For example, AI architectures, like most AI architectures, often contain multi-stage pipelines and complex initialization, sometimes involving as many as 100 distinct layers in each stage, which are ideal candidates for such an approach (Stephenson et al., 2015). Similarly, applying state preservation techniques for SoC or AI-based systems can give organizations similar gains in terms of time to simulation, consumed resources, and debugging cycles.

Future research and development might adapt this methodology for nearer hardware categories, and their corresponding verification engineers investigate ways of optimizing their workflows through an expanded variety of platforms. This methodology could also be expanded to build a more uniform testing framework to unify the verification processes of several architectures, aid in streamlining the testing process, and enhance the overall testing efficiency.

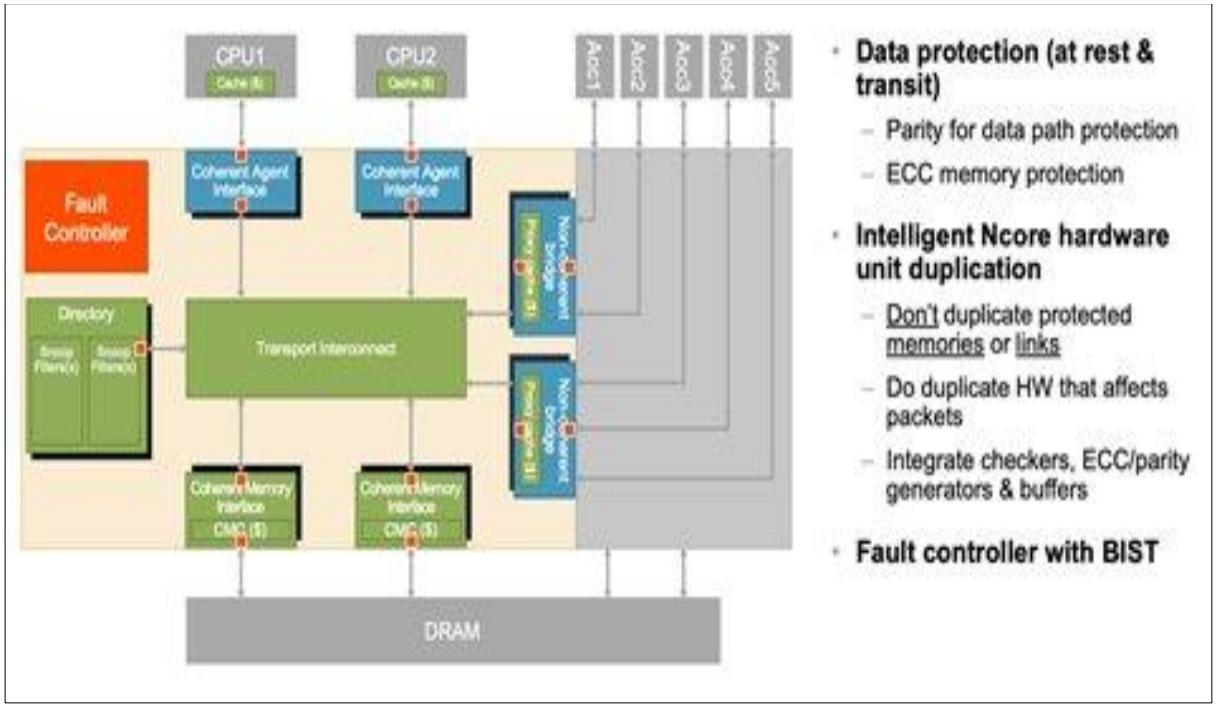

Figure 4 Re-Architecting SoCs for the AI Era

8.2 Dynamic Updates and Flexibility in Testing

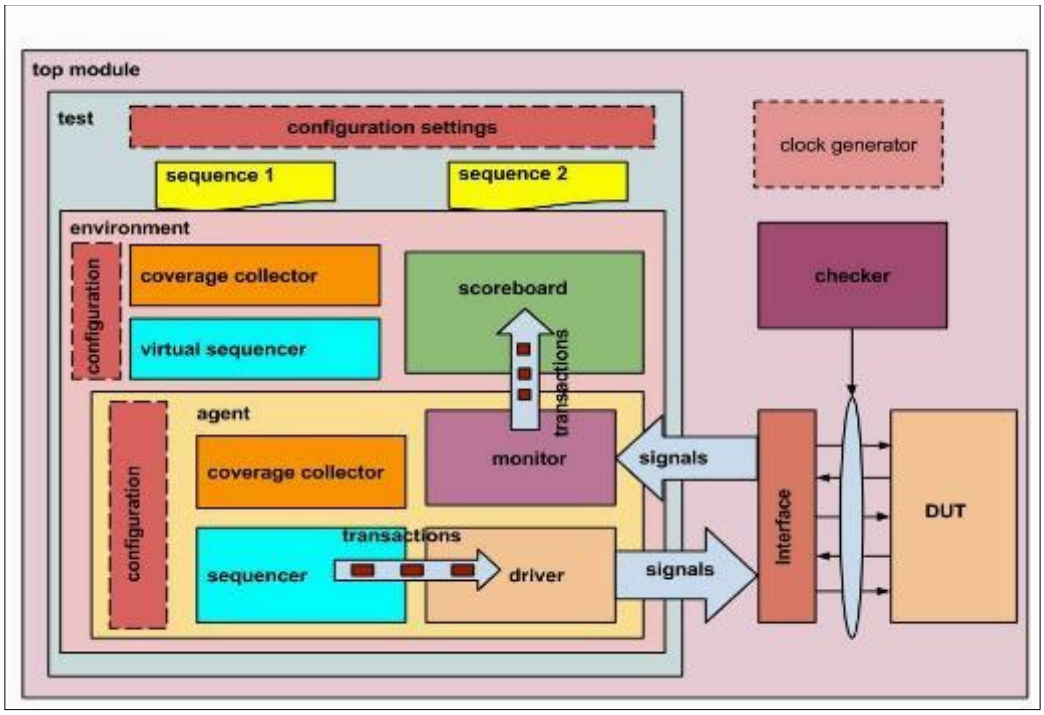

Figure 5 UVM Environment Components

Dynamic updating of test sequences and design components in a modern testing environment is one of the biggest roadblocks of modern testing, and it demands a complete recompilation of the test environment. With evolving designs, there is an evolution in test cases and sequences that validate them. Traditional UVM workflows have a current limitation where they become an issue whenever the design or the test sequence changes in some manner, and during those instances, it takes a full re-compile, which is not time efficient. To solve this, future improvements can be directed to providing more flexible testing by allowing the change of UVM components without a complete recompilation cycle.

This capability would allow engineers to quickly modify and update test cases without interrupting the testing process's totality. For example, test parameters could be dynamically adjusted with techniques such as automated scripting, or components could be modified automatically on the fly to speed up and decrease the response time of the testing workflow. Furthermore, these updates are bridged with scripting languages such as Perl, Python, or System Verilog. In that case, they become tools that integrate well and allow engineers to test sooner and better. Such improvement would significantly add to the flexibility and decrease the turnaround time of testing, significantly increasing the productivity of the whole verification process.

Abbreviations

- GPU – Graphics Processing Uniits
- SOC – System on Chips
- DUT – Design under test

9 Conclusion

This study presents a GPU testing methodology that reuses the initialization and configuration phases, significantly reducing simulation time and resource consumption. By saving the design state after the configuration and initialization, the method skips redundant steps and starts workload testing directly. It significantly improves testability efficiency, shaving 30-50 percent of simulation time depending on the workload complexity and even more for less complex tests. With the Functional Simulation Database (FSDB) support, the framework restores the design state to speed up test execution by skipping the start-up process. Perl scripting, in addition, gives flexibility, permitting test workloads to be reworked without reworking the total configuration cycle, which moderately streamlines the verification course. Since the verification cycle restarts after fixing the bug, once the bug is fixed, the cycle is done much faster than the traditional method, so this speeds up verification cycles and makes it easier to debug. Furthermore, the approach is scalable and thus amenable to other fields of use, including, for example, System on Chips (SoC) and other AI architectures alongside GPU testing. This method is beneficial for pre-silicon verification at the system and SoC levels and can efficiently handle test cases and eliminate redundant processes, imposing the use of optimal resources. The proposed methodology efficiently reduces the verification cycle and reduces the overall cost by 40 – 60%. Since the flexibility and Scalability of the Framework can reduce Verification Times and all those overall, in Hardware testing on various architectures, the framework seems to be a good candidate.

References

[1] Donaldson, A. F., Evrard, H., Lascu, A., & Thomson, P. (2017). Automated testing of graphics shader compilers. Proceedings of the ACM on Programming Languages, 1(OOPSLA), 1-29. https://dl.acm.org/doi/pdf/10.1145/3133917

[2] El-Ashry, S., Khamis, M., Ibrahim, H., Shalaby, A., Abdelsalam, M., & El-Kharashi, M. W. (2019). On error injection for NoC platforms: a UVM-based generic verification environment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(5), 1137-1150. https://ieeexplore.ieee.org/abstract/document/8680662/

[3] Henschel, O. P., & dos Santos, L. C. (2013, December). Pre-silicon verification of multiprocessor SoCs: The case for on-the-fly coherence/consistency checking. In 2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS) (pp. 843-846). IEEE. https://ieeexplore.ieee.org/abstract/document/6815546/

[4] Kim, Y., L ee, J., Kim, D., & Kim, J. (2013). ScaleGPU: GPU architecture for memory-unaware GPU programming. IEEE Computer Architecture Letters, 13(2), 101-104. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=16173a35a83b8d260c1175ece0f22fcf8dd89b09

[5] Moursi, A., Samhoud, R., Kamal, Y., Magdy, M., El-Ashry, S., & Shalaby, A. (2018, December). Different reference models for uvm environment to speed up the verification time. In 2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV) (pp. 67-72). IEEE. https://www.researchgate.net/profile/Sameh-El-Ashry/publication/334072659_Different_Reference_Models_for_UVM_Environment_to_Speed_Up_the_Verification_Time/links/5d184ee1a6fdcc2462b0dcbf/Different-Reference-Models-for-UVM-Environment-to-Speed-Up-the-Verification-Time.pdf

[6] Moya, V., Gonzalez, C., Roca, J., Fernandez, A., & Espasa, R. (2005, November). Shader performance analysis on a modern GPU architecture. In 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05) (pp. 10-pp). IEEE. http://www.csh.rit.edu/~oguns/ps3/vmoya.pdf

[7] Ray, S., & Hunt, W. A. (2009, November). Connecting pre-silicon and post-silicon verification. In 2009 Formal Methods in Computer-Aided Design (pp. 160-163). IEEE. https://repositories.lib.utexas.edu/bitstreams/c211346d-9861-4f1b-b938-6994877d529a/download#page=179

[8] Rieder, C., Palmer, S., Link, F., & Hahn, H. K. (2011, June). A Shader Framework for Rapid Prototyping of GPU-Based Volume Rendering. In Computer Graphics Forum (Vol. 30, No. 3, pp. 1031-1040). Oxford, UK: Blackwell Publishing Ltd. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.01952.x

[9] Simakov, N. A., Innus, M. D., Jones, M. D., DeLeon, R. L., White, J. P., Gallo, S. M., ... & Furlani, T. R. (2018). A slurm simulator: Implementation and parametric analysis. In High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017, Proceedings 8 (pp. 197-217). Springer International Publishing. https://par.nsf.gov/servlets/purl/10404197

[10] Stephenson, M., Sastry Hari, S. K., Lee, Y., Ebrahimi, E., Johnson, D. R., Nellans, D., ... & Keckler, S. W. (2015, June). Flexible software profiling of gpu architectures. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (pp. 185-197). https://www.researchgate.net/profile/David_Nellans/publication/289500717_Flexible_software_profiling_of_GPU_architectures/links/577e734408aeaa6988b0c8ba/Flexible-software-profiling-of-GPU-architectures.pdf