ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

Review Article

Open Access

A Comprehensive Automated Generation of Functional Coverage and Structured High Verification Plan for Complex Architecture of GPUs and AI Accelerator

Ankit Chandankhede

USA

ABSTRACT

Functional coverage is the most important metric in design verification and writing the functional coverage accurately needs skills and is manual process. Moreover, hitting different complex cross functional cover points requires directed complex testcase. Deploying AI to write the cover points and writing sequences to hit complex scenarios allows easier test writing and converging on functional coverage for complex architectures such as graphics-purpose unit GPU and AI accelerators. Script also allows to create a high verification plan and analyze effectively. This HVP plan can also be used across DUTs that allows to measure effective coverage across different levels of verifications abstractions.

***Corresponding author**

Ankit Chandankhede, USA.

Received: December 01, 2023; **Accepted:** December 05, 2023; **Published:** December 14, 2023

Introduction

Functional verification has paramount importance in ensuring the completeness of the testing of today's advanced semiconductor designs, particularly in cutting-edge technologies like Graphics Processing Units (GPUs) and AI accelerators. Maintaining and delivering high quality function design by meeting performance standards for such systems which are powering modern computational tasks on phones, servers, gaming for rendering and recently inference and training of AI models.

Labor intensive testing platforms for such complex architecture has been a bottleneck and is subject to human errors on either interpreting the specification of architecture or missing out on critical corner case scenarios. Test planners in such an environment must define each functional cover point across different features, tabulating a series of scenarios and potential security scenarios from hacker’s perspective. This is followed by coding the functional coverage meticulously through painstaking manual effort often needs expertise in coding such complex cover points and further craft the test cases. This demands immense engineering effort and prolonged development and testing cycles. Often such processes may hit pitfalls in execution, resulting in delayed product launch and missing market opportunity.

Complexity of modern computer architectures such as GPUs and AI accelerators, which encompasses numerous new features, data paths, newer set of instructions in kernel and new commands for processing a workload. Verification plays an important role in identifying design bugs and reverifying the bug fixes through comprehensive testing. Functional coverage provides a direct correlation between testing environment, test plan and design health.

Considering aforementioned challenges, deploying artificial intelligence (AI) and automation through scripting to efficiently converge functional coverage offers ease on complex testing requirements and minimizes human error.

This paper underscores the importance of AI-driven and automation methodologies that addresses complexities of functional verification in complex architectures. It presents a framework for creating a structured test plan including cover points, converting these cover points from a table to system verilog functional coverage code, mapping coverpoint in structured high verification plan (HVP) in terms of feature, priority, and unit coverpoints. This automation streamlines the most pivotal metric of the verification process, thus optimizing coverage writing and analyses of functional coverage for quicker convergence to ensure high quality design.

Details in subsequent sections of this paper provides in depth understanding of automation of structured coverpoint scenarios, creating coverpoints, reanalyzing results using AI model from synopsys, HVP mapping and real time feedback to current test constraints. Additionally, the paper highlights other bottleneck facets of verification methodology, their impact and potential solutions.

Automated Functional Coverage Generation

Test planners are provided with specific format of tabulating the scenarios from testplan with specific priorities, features and sampling conditions. Scripts are fed with this specific tabulated format to create cover points such as explicit bins, transitional bins, reputation bins, wildcard bins, ignore bins, illegal bins and cross coverage bins. Moreover, the script is scaled to parse the protocol and packet information to generate automated coverages and hence can be hooked to interfaces which follow a particular protocol. Thus, illuminating the burden of the exhaustive manual process of creating the cover points and enhancing the overall efficiency in verification.

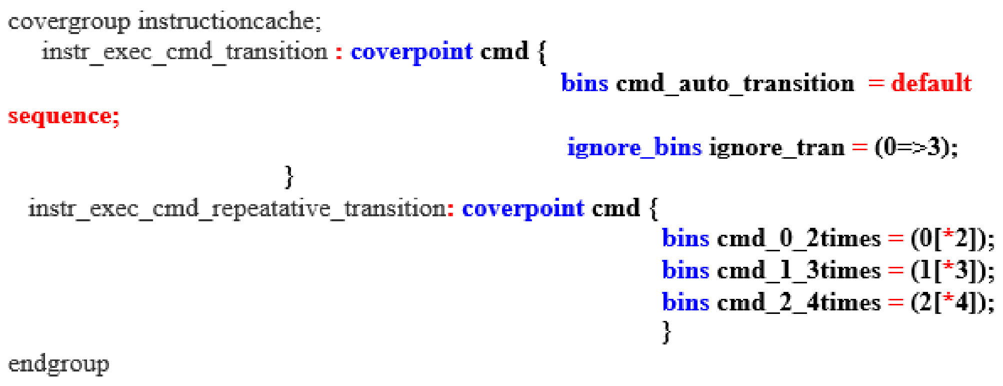
Following is the Structured Tabulated Format for Test Planners:

Legacy Feature	**Instruction Caching**									
Scenario	Type (Cover/ Assert/ Check)	SOC level	Priority	Bin info	Type of coverpoint	Signal or register to Code cps	Expression needed by cp	Sampling condition	Project	More info
All Execution Commands	Cover	N	P0	0:3 ignore: 0=>3	Transition	instr. exec_ cmd	instructoncache	instruction_ cache_ validq	A	Testplanner providing more info for better understanding of the bins
	Cover	N	P1	0:3 Repeat0: 2 times 1: 3 times 2: 4 times	Repetitive_ transition	instr. exec_ cmd	instructoncache	instruction_ cache_ validq	A	Checks repeation of command
Interface	Dispatch_ interface									
Scenario	Type (Cover/ Assert/ Check)	SOC level	Milestone	Bin info		Signal or register to Code cps	Expression needed by cp	Sampling condition	Project	More info

Column of type of coverpoint is used by the coverage coding script to specific create cover points. Script parses bins info for the values for creating the bins over the signals from the column of signals or register code cps. Following is example of coverpoint created.

Name of the covergroup is taken from the scenario column, coverpoint is created based off of type of coverpoint and name of signal and bin name is created based off the type of bins

Transition Coverpoint : Transition coverpoints are often used to check the transitions of finite state machines (FSM), change of virtual channels, transition of instruction or data type command, high priority to low priority cycles, different sources going into arbiters [1]. Such transition coverpoints exposes the corner case scenarios and hard to code. Hence the approach deploys parsing of signal with specific values mentioned in bin_info and creates a default *_auto_transition bins and creates different bins to be ignored, this way all transitions are covered except the ignored or illegal transitions . Repetitive cover points are also defined based off of number repetation expected from a particular value on a signal [2].



```
covergroup instructioncache;
   instr_exec_cmd_transition : coverpoint cmd {
                                              bins cmd_auto_transition  = default
sequence;
                                               ignore_bins ignore_tran = (0=>3);
                         }
  instr_exec_cmd_repeatative_transition: coverpoint cmd {
                                                      bins cmd_0_2times = (0[*2]);
                                                      bins cmd_1_3times = (1[*3]);
                                                      bins cmd_2_4times = (2[*4]);
                                                      }
endgroup
```

Implicit Coverpoint: Script determines the value of the implicit bins are taken from the bin info and signal to create cover point is taken from the Signal or register to code column.

Note: Illustrated example shows the implicit coverpoint created on a signal from an interface however implicit coverpoint can be created on any signal [2].

```
covergroup length_of_data;
cmd.dw_len_explicit  : coverpoint cmd.dw_len {
                            bins lens[]         = {[0:3]};
                            illegal_bins illegal_lens[] = {[10:15]};
                                                     }
endgroup
```

Likewise different type of cover points are generated through this script based on the table filled by the test planner and hence eliminating the coding cycle of such functional coverage and multiple compile process as these cover points are guaranteed to compile successfully.

High Verification Plan (HVP) Framework
High Verification plan is used for analyzing coverpoints in more structured way and is often mismanaged due to nature of coverpoints are not categorized properly and hence prioritizing coverpoints to be analyzed and focus on becomes challenging. Structured HVP plan is created by this automation script approach which relies on the table 1 as mentioned above. HVP plan add all the scenarios mentioned in the table based on the type of feature, units that it belongs to and priority. This allows the functional coverage analyzing engineer to effectively manage and focus based on the highest priority, feature and complex scenario such as cross feature and deadlock scenarios.

Script is also filters out the scenarios which are not applicable to a project based on the column project. If the entry related to this column is empty, script considers this scenario to be included by default to all projects.

The first entry in table defines the type of cover group/ cover point Interface:
Legacy Feature
- Legacy feature is defined as a feature which has existed in previous projects. These features usually are often tested over the generations of projects and hence defines the basic health of the design across projects and are covered through previous developed testcases. This is usually a first key metric in any product development cycle to maintain backward compatibility of newer architectures.

New Feature
- New feature is defined as a feature which has been introduced in current project. New feature is most prone to bugs as the implementation of architecture definition can be misinterpreted and often is prone to cross units discrepancies. Hence units which are impacted by such new features across architecture needs outmost attention and extensive coverage. New feature category helps to categorize unit coverages and hence helps to prioritize to verification of such bug prone design.

Cross Feature
- Cross feature cover points are defined to be a cross of multiple features. Cross feature combination often exposes the architectural definition or design implementation across features. In above case cross feature could be a combination of instruction cache crossing with interface [3].

Registers
- Register in designs are categorized under this category.

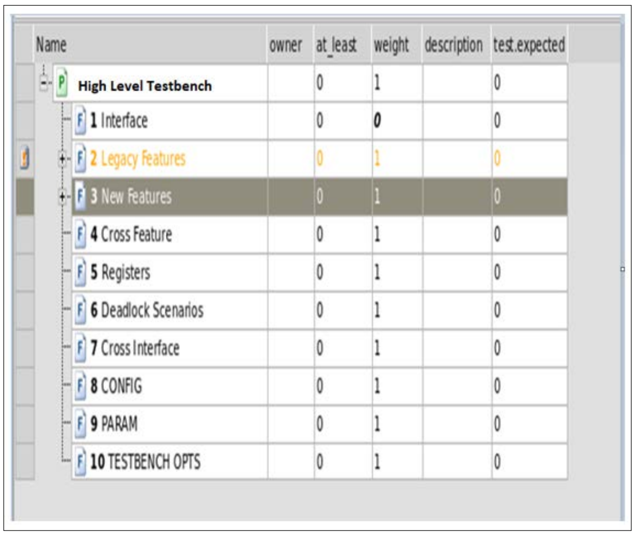
Deadlock Scenarios
- Architecture encompasses arbiters which usually takes in inputs from different sources and these sources competes for same part of the resource and hence creates chances of deadlocks due to back pressure of credits and holds on the outputs of arbiter interfaces [3].

CONFIG
- Modern architecture is defined to be scalable for multiple projects based on the platform for phones (low power design), servers, gaming processors and crypto processor. Such design is scalable based off different configurations which may or may not enable all features, multiple instancing of certain pipelines within the architectures based off of configuration and hence verifying such configuration is equally important.

PARAM
- Often design is scaled based on the parameters for different type of products which scales up or down the design such as depth of FIFO, instancing same pipelines multiple times, cores included, cache size. Hence randomizing these parameters across design constraints for different product is outmost important. Cover point for such parameters are included under this category.

Testbench_opts
- Often testbench enables different features or capabilities using SIMV arguments options which controls the test sequences and constraints. Combination and standalone of such simv arg option cover point is critical to not only test feature standalone but also cross different feature across architecture. Such parameters are cover through this category.

High level verification plan structured created by this automation used in synopsys tool is shown below:



Name	owner	at_least	weight	description	test.expected
High Level Testbench		0	1		0
1 Interface		0	*0*		0
2 Legacy Features		0	1		0
3 New Features		0	1		0
4 Cross Feature		0	1		0
5 Registers		0	1		0
6 Deadlock Scenarios		0	1		0
7 Cross Interface		0	1		0
8 CONFIG		0	1		0
9 PARAM		0	1		0
10 TESTBENCH OPTS		0	1		0

Automated Analysis and Optimization

Analyzing coverpoints and likewise identifying the under or over constraint are difficult and prolonged engineering process. Synopsys tool is deployed for such analysis by feeding back the coverage generated after the regression of test suites. This tool identifies different under constraints from stimulus and coverage report and hence exposing the test sequence gaps and optimizes the stimulus generation which in turn improve test suite quality and early convergence of functional coverage.

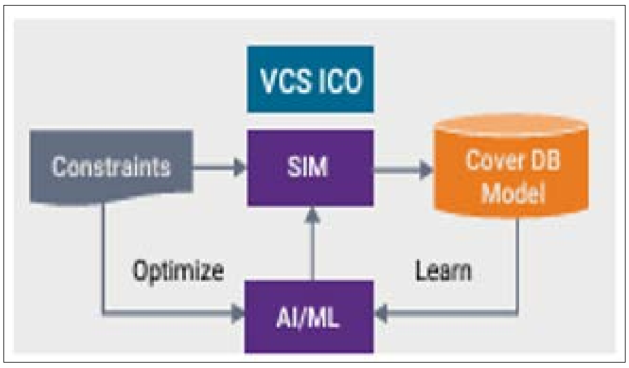


Figure 1 : Constrained Random Verification with ICO

Ease of Synchronizing across Test Team

Usually the verification team included multiple engineers. Often test plan and test writers or sequence writers are different and hence synchronization across team members in test development or functional convergence is warranted. This structured table provides in-depth understanding to test write from test planners perspective and hence fewer iteration of feedback and resulting in better communication between test planner and test writers.

Case Study: Application Across Different Projects and DUTs

Case study using this approach shows faster cycle of creating bug free coverpoints from test planning stage, saving about 85% of coding efforts on cover points. Very complex and cross IP coverpoints are bit challenging to be created through such automation and hence is manual process. However relieving effort on creating certain complex coverpoint through such automation allows verification engineer to focus on greater complexity of coverpoints.

Structured HVP has been key in efficient analysis of functional coverage and has helped verification engineer to focus on high priority coverpoint and hence effectively close on coverpoints quickly.

Further AI tool from Synopsys provides additional assistance to verification to engineers in identifying the under constraint and also automatically creating stimulus to hit certain coverpoints [4]. Automated stimulus from Synopsys has also helped to find corner case scenarios specifically in cache hit-miss , arbiters and command execution in execution scenarios in GPUs and AI accelerators.

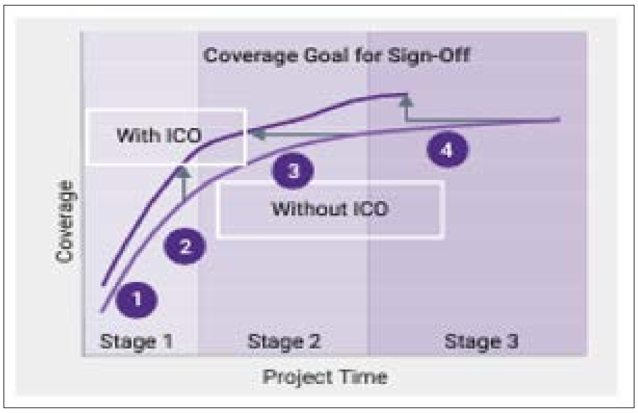


Figure 2 : Effects of ICO on Coverage Convergence

Conclusion

The integration of automation and AI tool from Synopsys in functional verification improved efficiency, significantly enhances the efficiency and accuracy of coverage-driven verification processes for complex semiconductor designs. By automating the creation of functional coverage and HVPs, this approach not only accelerates verification cycles but also improves overall design quality and reliability.

Future Directions

Future research directions include on expanding the capability to create sequences based on the planned coverpoints using AI which allow verification engineer to reduce the test developing cycle and focus more on the complex nature of testcases if there be any missing. Further optimizing current scripts to handle more complex coverpoint and better structure of HVP across different type of architectures seems a promising endeavor. Additionally, exploring AI's capability to debug the failure cases to assist verification engineers on gruesome debug cycle remains a potential to be explored.

References

1. (2014) Functional coverage. ASIC World https://www.asic-world.com/systemverilog/coverage17.html.
2. Jonathan Bromley, Mark Litterick (2016) Effective SystemVerilog Functional Coverage: design and coding recommendations. Snug https://assets.website-files.com/63f4bb21bd5303fe472ad00e/64957f836fbf7349210cf29c_bromley_coverage_paper.pdf.
3. ManiKumar Jammigumpula, Prashant K Shah (2020) A new mechanism in functional coverage to ensure end to end scenarios https://ieeexplore.ieee.org/document/9298222.
4. Accelerating Verification Shift Left with Intelligent Coverage Optimization. Synopsys https://www.synopsys.com/cgi-bin/verification/dsdla/pdfr1.cgi?file=ico-wp.pdf.