2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE OCTOBER 26-27, 2021

An Analysis of Stimulus Techniques for Efficient Functional Coverage Closure

Caglayan Yalcin, Senior Design Verification Engineer, QT Technologies Ireland Limited
(*cyalcin@qti.qualcomm.com*)

Aileen McCabe, Design Verification Engineer, QT Technologies Ireland Limited
(*amccabe@qti.qualcomm.com*)

***Abstract*—This paper describes different approaches to closing coverage. As SoC designs become more complex, achieving 100% coverage using the industry standard constrained-random testing is becoming exponentially more difficult. In this paper, different approaches such as graph-based stimulus and autogenerated stimulus are proposed as alternative strategies to constrained random testing. These alternative strategies are automatable and take up less resources, memory, and verification time by an order of magnitude.**

Keywords— Design Verification, Constrained Randomization, Graph Based Stimulus, Coverage Closure

I. INTRODUCTION

Verification teams have been deploying constrained-random stimulus techniques to find bugs at an early stage in projects [1]. Even though constrained-random stimulus uncovers bugs early and contributes to functional coverage, which is one of the widely adopted dynamic techniques, creating sufficient tests to verify the design and closing the coverage is still one of the biggest challenges in verification [2]. One of the main reasons is the difficulty of exercising every feature with constrained-random approach for a design with a very large stimuli space when the problem is adequately modelled [3]. Sooner or later, the verification engineer needs to enhance either the test plan or the stimulus techniques to be able to close the coverage. This paper describes and deploys transition coverage closure with constrained-random stimulus and graph-based techniques and analyzes the cost in terms of resources and time spent during coverage closure. Additionally analyzed, is an approach that enables automatic generation of stimuli from the coverage model.

II. CONSTRAINED-RANDOM STIMULUS

The desired variables are randomized each time the sequence is called. The randomize function has no knowledge of previous values. The values selected by the randomize function are chosen from the entire valid stimuli space. It results in duplication of coverage in some bins and some bins not being covered even after multiple runs.

To target the last few uncovered bins without running the simulation excessively, verification engineers often edit the test to steer the stimulus. This manual intervention wastes time and is not scalable. It needs to be completed each time coverage is analyzed. If a bug is found at a time when coverage closure is nearly complete, coverage analysis needs to be done from the beginning on the new RTL.

III. GRAPH-BASED STIMULUS

As designs have become more complex, the use of randomly generated stimulus to achieve full coverage has made the process more cumbersome and tedious. The graph-based approach aims to solve the difficulties associated with constrained-random stimulus described above.

Functional coverage is usually considered as an output of running regressions which shows how effective the stimulus used has been in covering design. In the graph-based approach the target coverage is determined before a test is run. This coverage goal can then be used as an input. The coverage goals are typically defined using SystemVerilog covergroups, coverpoints and constraints. This allows full reuse of the existing testbench.

First, the tool identifies all solutions to the constraints in the stimulus space and then targets each possible path. The tool used for graph-based stimulus can remember what paths have previously been targeted. This prevents any duplication of tests as each path/bin is covered only once. The constraints and covergroups provided to the graph-based tool must cover all the scenarios that are required to verify the design. Since graph-based coverage only covers the exact stimulus that is defined, if the stimulus space is defined incorrectly it will result in missing parts of the design. It relies on the verification engineer to understand the stimulus space and create the functional model that perfectly defines the cases that need to be covered. In the below figure, the valid stimulus space is defined in yellow. If the constraints have been incorrectly given to the tool and there is a valid set of stimuli within the blue circles, it will not be covered.

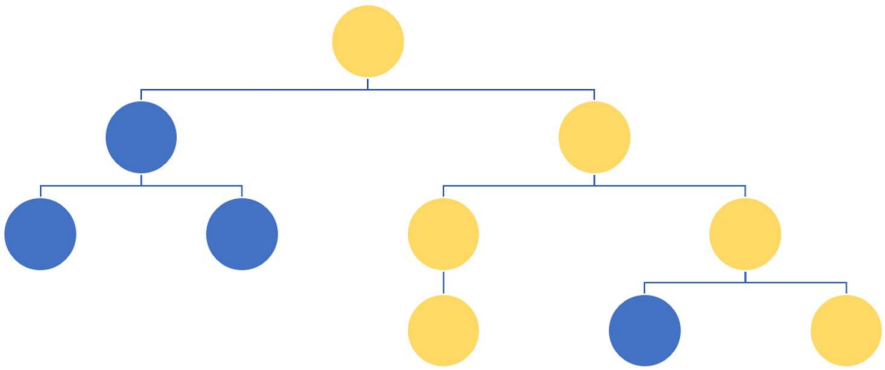

Figure 1 – Graph-based stimulus space. Valid stimulus space in yellow. Invalid stimulus space is in blue defined by constraints

Graph-based stimulus also relies on the path being decided before runtime. This means that the graph-based solution would not work well if some of the stimuli need to be constrained based on the design behavior during runtime.

Graph-based portable stimulus tools such as Mentor's inFact can be used. inFact enables reuse of the existing testbench structure including SystemVerilog covergroups, classes and drivers. inFact works by replacing the UVM randomize function with targeted random stimulus to focus on paths that have not yet been covered. It keeps track of covered and uncovered paths.

IV. AUTOGENERATED STIMULUS

In this approach, the stimuli space is autogenerated as per the coverage model. Bottom-to-top and top-to-bottom approaches are followed to feed the tests with the autogenerated stimulus.

Bottom to top: The autogenerated stimuli are picked up by a server. The server is then used to distribute the stimuli in tests as and when required. The connection between the server and the tests are done with socket programming. The tests are considered as clients in this topology. Client request from the test is raised via DPI-C.

Top to bottom: The regression manager is fed with the auto-generated stimuli and the regression manager runs the simulations by feeding the tests with a unique stimulus.

In both cases, each stimulus is sent to a unique test in the test suite. Hence, every test run simulates a unique stimulus and there is no repetition in the regression, like in a graph-based approach.

V. RESULTS

A. *Constrained random stimulus*

To compare each method, the focus would be on a particular covergroup with a large coverage space. Consider a coverpoint sleep mode which can take on 4 different values. The aim is to test a processor where a sequence of 4 sleep/wake cycles are run. In each cycle a randomized sleep mode is used.

This results in a transitional covergroup with 256 (4^4) different outcomes.

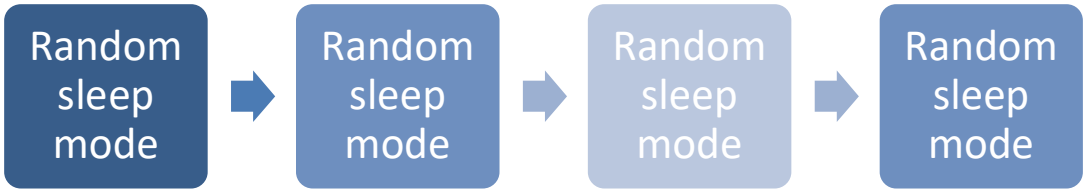

Figure 2 - Transition cover bin

Each test runs through 4 sleep/wake cycles and it is assumed that each sleep mode is equally likely and independent (0.25 chance of each). Therefore, geometric distribution can be used to estimate the number of tests required to run for full coverage.

$$n\sum_{k=1}^{n}\frac{1}{k} = 256\sum_{k=1}^{256}\frac{1}{k} \approx 1568 \text{ runs needed for full coverage}$$

Each test takes 23 mins to run so the total compute time needed is 601 hours (or approximately 25 full days). This is hugely wasteful of resources as there are many redundant runs to cover the 256 bins. The number of runs can be smaller or larger than the theoretical number. On one occasion, it was found that 1350 runs were needed to close coverage on the covergroup.

The total coverage data stored in memory by all these runs is 84.3 GB. The biggest drawback is that every change in design requires regeneration of coverage, which can consume the same or larger memory space for storage.

The other option with constrained random approach, is to run simulations until you hit most bins and then further constrain the UVM test to hit the last few uncovered bins. This manual effort is needed each time coverage closure is to be accomplished and is extremely wasteful of the engineer's time.

B. Graph based stimulus

Graph-based stimulus saves a large amount of compute resources as it targets each bin only once, preventing duplication of simulation runs. The first attempt at covering the covergroup in inFact resulted in running the test 607 times. This number is much larger than the theoretical minimum of 256, as the graph-based tool used is not best suited for transitional covergroups. This still resulted in substantial savings in memory and time as can be seen below:

	Constrained Random	**Graph-based stimulus (transitional)**	**% saving**
Compute time (hrs)	601.07	232.68	61.28%
Data stored in memory	84.3 G	32.7 G	61.2%

Once the covergroup was converted to a cross covergroup, inFact achieved full coverage in the lowest number of runs, which is also the theoretical minimum of 256. The transitional coverbin can be converted to a cross coverbin as shown below, where sleep_mode_e is an enumerated type with each of the sleep modes defined:

TRANSITIONAL COVERBIN:

```
rand sleep_mode_e sleep_mode;

bins trans[] = [SLEEP_MODE_0:SLEEP_MODE_3] => [SLEEP_MODE_0:SLEEP_MODE_3] =>
[SLEEP_MODE_0:SLEEP_MODE_3] => [SLEEP_MODE_0:SLEEP_MODE_3];
```

CROSS COVERBIN:

```
rand sleep_mode_e sleep_mode;
rand sleep_mode_e sleep_mode_1;
rand sleep_mode_e sleep_mode_2;
rand sleep_mode_e sleep_mode_3;

cp_sleep_mode_0 : coverpoint sleep_mode;
cp_sleep_mode_1 : coverpoint sleep_mode_1;
cp_sleep_mode_2 : coverpoint sleep_mode_2;
cp_sleep_mode_3 : coverpoint sleep_mode_3;

cross_sleep_mode : cross sleep_mode,sleep_mode_1,sleep_mode_2,sleep_mode_3;
```

Each of the sleep modes which were randomized by inFact can then be fed to the test until all sleep modes are consumed. The behavior from the test perspective is the same. Below is a comparison of cross coverage results with those from constrained random method and the savings are evident:

	Constrained Random	Graph-based stimulus (cross)	% saving
Compute time (hrs)	601.07	93.13	84.51%
Data stored in memory	84.3 G	13.8 G	83.62%

C. *Auto-generated stimulus*

This approach is similarly structured as the graph-based stimulus technique as it targets each bin only once. Thus, when compared with constrained-random approach, similar savings in consumption of computing resources and time are observed.

	Constrained Random	Auto-generated stimulus	% saving
Compute time (hrs)	601.07	93.03	84.52%
Data stored in memory	84.3 G	13.8 G	83.62%

As the entire set of stimuli can be autogenerated with scripting and the socket programming can be done in C, tool independence becomes the biggest advantage of this approach. Client-side code can be called via DPI-C to get the stimulus that the server needs to provide. However, this requires basic knowledge of socket programming and server client topologies in networking, which a verification engineer may not have. This can be managed if the stimuli distribution is overseen by the regression manager. In which case, the verification engineer need not have the knowledge of networking and socket programming.

VI. PITFALLS

A. *Graph-based stimulus*

There are some pitfalls associated with using graph-base stimulus. The main downside is that it relies on the verification engineer to create a perfect model of covergroups and constraints to be given to the tool. Any cross covergroup that is missed would be left uncovered. As opposed to graph-based approach, there is no need of functional coverage model to generate the stimulus in the constrained random approach. Hence, the cross coverage of two variables can still be covered in the constrained random approach by simply defining valid constraints. To help with this there is a 'bug hunt functionality' in inFact, using which, a test can be run after all covergroups have been covered. This results in random stimulus being applied, which alleviates this concern.

Another downside with the tool inFact is that it works best with cross cover groups. As the covergroup was a transition case, the sequence had to beedited and a virtual cross coverage model had to be defined to increase the efficiency. Using inFact with transitional coverbins resulted in 351 duplicate runs. This number is still much smaller than what was observed in the constrained random approach. However, to reach the maximum efficiency, a significant code refactoring may be needed in a testbench that heavily relies on transitional cover bins.

The final downside of using graph-based stimulus is that it can be hard to replicate the failing scenario if one exists. If one of the testcases fails, it would be required to manually constrain the test to reproduce that exact waveform.

Even with these downsides noted, the graph-based approach still saves a large amount of time for the verification engineer.

B. Auto-generated stimulus

The main downside of the auto-generated stimulus approach is that the data or the generated stimulus sent to clients (tests) by the server is raw. Hence the client or test needs to process this raw stimulus received to drive the correct stimulus on the design. Proper decoding code needs to be developed for complex scenarios, just to interpret the data sent by the server for the simplest client code or stimuli. Another downside is reproducing the exact simulation, like graph-based approach. Graph-based approach requires manual constraints to be applied in the test whereas simply rerunning the simulation with the same seed is enough in constrained-random approach.

VII. CONCLUSION

It was observed that although constrained random approach doesn't require a setup beforehand and is able to generate functional coverage in earlier stages of the verification cycle, it lags when it comes to closing the coverage for verification sign-off. Although the two approaches (graph-based and auto-generated stimulus) initially require some setup effort, coverage closure with these methods is much faster and is less expensive when it comes to consumption of compute resources. Graph-based approach needs additional tools to be deployed in the verification environment, but once deployed it can leverage the full force of portable stimulus. On the other hand, the autogenerated stimulus approach doesn't require any special tool. It uses concepts like server-client interaction and regression management to make coverage closure as fast and economical as possible. Thus, both graph-based and auto-generated stimulus approaches help to overcome the challenge of closing coverage for complex models such as transition or cross coverage.

ACKNOWLEDGMENTS

Thanks to the Mentor team for assistance in getting Infact set-up in our testbench.

REFERENCES

[1] Nguyen Le, Mike Andrews, Efficient Bug-Hunting Techniques Using Graph-Based Stimulus Models, DVCon, 2016

[2] Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study

[3] Staffan Berg, Mike Andrews. A New Stimulus Model for CPU Instruction Sets. Verification Horizons, November 2015