

Contributed Article

[Print](#) [Email](#) [Bookmark](#)

Understanding coverage with multiple verification methods

By Rajeev K. Ranjan, Jasper Design Automation; Jay Littlefield, Jasper Design Automation; and Brian Bailey, Brian Bailey Consulting
With contributions from David Lacey, Hewlett-Packard; Mercedes Tan, Sun Microsystems; and Somdipta Roy, Texas Instruments

11/28/07

In the world of functional verification, the term "coverage" is omnipresent, having seen decades of use for a variety of purposes. However, within the hardware design community, the usage and interpretation of this term has been confusingly inconsistent. The lack of an industry accepted definition for coverage extends well beyond the simple issue of terminology to concerns regarding the interoperability of coverage data from different vendors. The terminology and interoperability problem is further exacerbated when multiple verification technologies (such as testbench driven simulation and formal verification) are introduced into the verification flow.

In this article, we present a high-level view of different usages of the term "coverage." Furthermore, we discuss how this term should be interpreted in the world of mixed simulation and formal verification methods, and then we provide some examples for leveraging these two methods in an overall verification flow.

Introduction

In today's functional verification flows, coverage is consistently the metric of choice for determining when a design is ready for tape-out. Traditional simulation-based verification long ago surpassed the era in which directed testing coupled with code coverage was sufficient for fabrication sign-off. Introducing constrained-random testing (CRT) into traditional simulation flows required new metrics to determine the quality of such tests. From this need grew functional coverage, designed specifically to measure the effectiveness of a test suite at identifying potential bugs within the design.

As design architectures have evolved, so have verification technologies. In particular, formal verification is seeing much wider adoption among design and verification teams today. This is because it has simply become far too time consuming to verify all paths through mission critical code. Though many different forms of formal verification are available, the most common use model is complementary to traditional simulation-driven verification. Leveraging each technology where it best fits in the flow would seem to provide greater verification confidence, but collecting quantifiable data to this effect has been challenging.

Given the high complexity of today's designs, we are now seeing renewed interest in the area of coverage analysis. In October 2006, at Jasper Design Automation's initiative, 17 EDA users and vendor companies got together to attend a Coverage Interoperability Forum (CIF) to discuss the possibility of creating a long-term initiative to deal with the interoperability issues associated with different sources of verification coverage data. The CIF effort subsequently led to the formation of the Unified Coverage Interoperability Standard (UCIS) committee within Accellera. This committee hopes to solve some of the definitional problems mentioned and to provide an API that will allow tools to share coverage data, thus solving some of the interoperability issues.

In the next section we will explore exactly what coverage is and the important attributes of a universal coverage metric.

Three Types of Coverage

The process of functional verification involves stimulating the design, propagating the effects to an observable point, and checking the response for the correct behavior. The completeness of this process is dependent on the completeness with which the design is stimulated (all possible behaviors have been exercised), the ability to ensure that the effects (both good and bad) of exercising the behaviors are propagated to an observable point, and on the completeness of the checking mechanism (all relevant design behaviors have been checked against the specification).

We use three different coverage terms to capture the "completeness" of these three components:

1. Stimulus coverage

1. Stimulus coverage

This provides a measure of completeness of stimulation the design has been put through, and is a direct measure of controllability of the design. This is the most common interpretation of the term coverage in the world of functional verification today. More specifically, the term "coverage" is widely used to denote the stimulus coverage in a simulation-based framework and a number of metrics have been proposed and used. Examples of stimulus coverage are code coverage and functional coverage.

2. Observed coverage

This is the classic production test situation, where the objective is to ensure that all fabricated parts behave identically to the reference. To do this, every point in the design must be both controllable and observable, and thus these metrics combine both stimulus and observability. Any differences on the observed outputs are indicative of a failed part. There have been some developments in this area such as OCCOM [Fallah2001] but none are available in commercial products today.

3. Response checking coverage

This provides a measure of completeness of the response checking mechanism that is part of the verification framework. An example of this would be a completeness measure of a set of properties written in some executable form. This should not be confused with assertion coverage, since that term is used inconsistently today, or with metrics that attempt to combine response coverage with stimulus and observability coverage.

The ultimate goal is for a metric that provides a single measure of completeness of verification overall, taking into account design stimulation, propagation and response checking. In an ideal world, with well defined notions of this coverage and well identified heuristics to approximate it, the verification completeness should be measured by this alone.

In the next few sections we will look at various forms of coverage, first in a simulation flow and then in a formal flow before suggesting ways in which the two can work together to improve the overall efficiency of the verification process.

Simulation-Based Verification Method

Simulation-based verification is a combination of both directed and constrained random techniques. Directed testing is used to target specific behavior explicitly described in the specification. This ensures "known behavior" is tested at a minimum level. Constrained random testing complements this by using volume vector testing to flood the design with large numbers of pseudo-random vectors in order to measure more abstract elements of the design such as corner case behavior, maximum data rate handling, and unforeseen conditions meant to more accurately depict real world data flows. Three primary methods are used to check the response of the circuit to these applied stimuli:

1. Embedded reference

Directed testing has been used for decades and is still the preferred functional verification method for some people. With directed tests, the expected results are usually embedded into the same file as the stimulus. That means an engineer has worked out how they expect the circuit to behave and created the appropriate response data to be verified. The big disadvantage with this technique is that it may change every time the design changes because of timing differences. The tendency to limit the response checking to just a few indicators of success rather than checking all of the results that are obtained leads to incomplete verification.

2. Reference model comparison

This is the traditional two-model comparison form of verification. A separate reference model describing the specified behavior is crafted independently of the design-under-verification (DUV). Identical stimuli are driven through both models and reference points, typically outputs, of each design and measured at meaningful points in time. A mismatch between the measured points indicates a problem with one model or the other. Often, the reference model will be written at a higher level of abstraction than the implementation-level RTL code. This can lead to differences in behaviors that are not errors, since the abstract model will define multiple acceptable behaviors rather than just the one exhibited by the implementation. This often means that the comparison process is not trivial.

3. Monitor/observer/assertion

This class of checking describes everything from testbench monitors designed to track signal changes to Assertion-Based Verification (ABV) monitors noted in much of the present day verification literature. The intention is to break down the verification environment into miniscule pieces and scatter them throughout the design in order to achieve a high level of debugging locality when a failure is identified. While used successfully for years in software development, assertions have found mixed acceptance among designers today, who are primarily responsible for entering them into the RTL during coding.

We will now look at some of the coverage techniques in use with simulation-based verification.

Stimuli Coverage in Simulation

Simulation-based verification is a sampling process of all of the possible input sequences that could

be applied to the design. Given that this is a heuristic process, it is very important to measure the completeness of the input sequences that are checked. Several coverage metrics are used in practice to measure the stimuli completeness, including:

- Code coverage metrics (line, conditional, expression, block)
- Finite state machine metrics (state coverage, transition coverage)
- Transaction-based metrics
- Functional coverage points based metrics
- Assertion coverage metrics

Considering that response checking coverage has not been adopted in the simulation world, stimulus coverage is synonymously perceived (albeit erroneously) as overall verification coverage. This will result in an over-optimistic evaluation of the completeness of the verification process.

Observability and Response Checking Coverage in Simulation

Today, due to the almost total focus on stimulus coverage metrics, there is an implied acceptance that full response checking is being performed, although in most cases this is not shown to be true. In the software world, mutation analysis [DeMillo1978] has been proposed to deal with this, but has seen little adoption. Mutation analysis makes small changes in the system and looks for a difference to be observed on one or more of the outputs. Even this technique does not deal with the checking of the result, as it only looks for a difference on an output rather than checking for correct operation.

Recently, a commercial product has become available from Certess [Bailey2007], [Certess2007], which addresses the complete coverage metric in a simulation world. Specifically, a fault model for the design is applied and the completeness of the stimulus set, and the set of response checking mechanisms is measured based on the ability of the verification framework to catch errors resulting from those faults.

Formal Verification Method

In formal verification, the correctness of a design is verified with respect to a desired behavior by checking whether the mathematical state of the design (typically a labeled state-transition graph) satisfies the specification of this behavior, expressed in terms of a temporal logic formula (specified by a set of properties) or a finite automaton.

Assuming that the implementation of the formal techniques is accurate, the verification completeness of a given design is then dependent on:

1. The completeness of the property set (specification) against which the design is formally verified,
2. The accuracy of the constraints (the set of legal stimulus under which the design gets analyzed), and
3. Whether formal analysis was complete or partial. A partial analysis means that the computation was not able to complete due to resource constraints such as time or memory.

Stimuli Coverage in Formal Verification

Formal analysis attempts to analyze the conformance of the design against the specification under all legal stimulus, specified through the constraints. When this analysis is completed, the stimuli coverage for the design and specification pair is 100%, meaning the design has been sensitized for all possible legal input scenarios.

Due to the mathematically complex nature of this analysis, it is possible that the computation may hit time and memory limits. In this case, the stimuli coverage is not 100%. Often an incomplete formal analysis is measured in terms of the number of cycles, with reference to some specific clock, of analysis the design has gone through. For example, "k" cycles of formal analysis establishes that the design has been analyzed for all input sequences of length "k" (this statement is true for most common type of formal analysis, which traverses the state space of the design in a breadth-first manner).

In another scenario, the formal analysis can be done under some restricted set of inputs (a subset of inputs could be tied to a constant). This typically happens when a design's validity against the specification has been formally analyzed for a particular mode setting. Another scenario of partial formal analysis is when an arbitrary starting state is used. This precludes the guarantee that the property has been analyzed for all states that the design could get into.

The most accurate way to measure the stimuli coverage for partial formal analysis would be to determine the fraction of all possibly reachable state space that has been covered. However, establishing this measure is not practical since determining the exact reachable state-space itself is a hard problem.

One important aspect of formal analysis that should be kept in mind is that the analysis pertains to a specific property or to a group of properties. This is in contrast to simulation method, where it is possible that the design is stimulated while all the properties are actively checking for the right

behavior.

For the reasons above, there is no practical and technically sound way to create a "simulation equivalent" stimulus coverage metric for these incomplete formal analysis scenarios.

Response Checking Coverage in Formal Verification

Measuring the exhaustiveness of a specification in formal verification ("do more properties need to be written?") has a similar flavor to measuring the exhaustiveness of the input sequences in simulation-based verification ("do more input sequences need to be created?"). Nevertheless, while for simulation-based verification it is clear that coverage corresponds to activation during the execution on the input sequence, it is less clear what coverage should correspond to in formal verification, as in model checking all reachable parts of the design are visited [CKV2003].

There are some sanity checks that are used to assess the modeling of the design and the specification, resulting in vacuous satisfaction of the specification. In vacuous satisfaction of a specification, the error checking is never enabled (because the pre-condition is never true), thereby making parts of the specification irrelevant to its satisfaction. For example, the specification "every request must get a grant within 6 cycles" is vacuously satisfied in a design where no requests are sent. A similar direction is to check the validity of the specification (a specification is valid if it holds for all designs).

Research in the area of coverage for formal verification has been focused solely on state-based coverage. This metric is based on mutations (small errors or mutants injected into the design) applied to the finite state machine (FSM). Essentially, a state "s" in the FSM is covered by the specification if modifying the value of a variable in the state renders the specification untrue.

In [CKV2003], Chockler et al adapted the work done on coverage in simulation-based verification to the formal-verification world in order to obtain new coverage metrics. For a number of metrics used in simulation-based verification, the authors presented a corresponding metric that is suitable for the setting of formal verification and described an algorithmic way to check it. Examples given in the paper were code coverage, circuit coverage, FSM coverage and mutation coverage. In fact, they claim that almost every heuristic regarding stimuli coverage in the simulation world has a corresponding metric to determine the response checking coverage in the formal world.

In [CLA2006] Koen provides a way of approximating an answer to the question – "Have we specified enough properties?" Given the interface of a design under verification, plus a property list, the technique identifies cases where some outputs of the design are not constrained at all by the properties. In practice, under some scenarios (the design states) the outputs are allowed to be under-constrained. The technique allows for easy specification of such "don't care" exceptions.

Vacuity detection and coverage estimation are complementary techniques. The former is concerned with strengthening individual passing properties or weakening failing ones, so as to improve their effectiveness as specifications, while the latter is concerned with augmenting a set of properties, without modifying the existing ones, so that, collectively, they represent a better specification. It is therefore natural to consider the combination of the two techniques [CKV2003].

Combining Coverage Metrics from Multiple Verification Techniques

With the wider adoption of formal techniques in the verification flow, the question often asked is - "How does one combine the results from formal verification with those obtained from simulation?" The intent behind this question is to eliminate any duplicate verification effort between these technologies and improve overall verification productivity. An effective means of measuring overall verification coverage which combines both formal and simulation metrics remains elusive.

As discussed in the analysis of stimuli coverage for formal verification, the traditional metrics used in simulation-based approach are not quite applicable. Similarly, the standard "partial" results from formal (such as bounded analysis, restricted mode analysis) have no counterpart in the simulation world. For example, if a line in an RTL description falls in the cone of influence of a property/assertion, then the formal analysis guarantees that the line has been exercised for all possible combinations of states that the design could get into. The line coverage approximation in simulation only guarantees that the line has been activated one or more times. On the other hand, in formal analysis, the activation and sensitization of an RTL line is observed by only the property being currently analyzed, whereas, the line coverage in simulation methods typically pertains to all the properties in the system. In essence, it is not technically sound to arbitrarily take coverage metric numbers from the two methods and combine them.

There are two methodologies in which the two verification methods (simulation and formal verification) can be leveraged to improve overall verification productivity – top down and bottom up. An example of each will be provided, but there are many more ways in which these two tools can be used together co-operatively.

In the top down view, simulation and formal can be leveraged in a systematic way as dictated by a verification plan. Effective verification planning adds predictability into the verification flow by specifying exactly what needs to be tested. Planning also provides the necessary structure to identify key areas where complementary verification technologies can be applied effectively.

Verification planning tools aid engineers in the verification plan creation and tracking process by simplifying the planning process and enabling easy reporting of progress against the plan.

Planning tools can enable the creation of a verification plan, broken down along the hierarchy of features provided in the specification. By making sure that the verification tasks are tied to the specification and assigning appropriate verification method (simulation or formal analysis) to each such task, one can significantly reduce, if not eliminate, any redundant verification effort. For example, for a design block which has a huge number of possible scenarios (resulting in a large number of coverage targets for simulation), it would be more prudent to attempt to verify it via formal analysis. The formal analysis, with its exhaustive analysis capability, can effectively cover this large set of targets, eliminating the need for handling them via simulation methods.

In the bottom-up view, formal analysis and simulation methods can co-operate to help improve overall coverage. For instance, formal analysis can establish that certain coverage targets (line, expression, functional) are not reachable for any legal input sequence. This would eliminate unnecessary simulation effort targeting those areas. In addition, formal analysis can create specific scenarios automatically which when simulated will fulfill specific coverage targets.

Another example where formal analysis can reduce the simulation effort is where a design block is fully verified for a specific mode (perhaps the most complex one). The simulation effort can then be targeted to provide coverage in untested modes. On the other hand, simulation can provide a sequentially deep "seed state" (a set of states which would serve as the starting point for formal analysis). The idea is that it would be practically difficult, if not infeasible, to reach these seed states using formal analysis. It's important that the tool used for formal analysis has the capability to handle verification tasks that are at the block specification level.

The analysis above is not limited to just simulation and formal methods. Any number of methods with their respective strengths can be part of an overall verification plan and can be appropriately leveraged to improve overall verification productivity.

Summary

The term "coverage" has multiple usages in the world of functional verification. In this paper, we have described the notions of stimuli coverage, observed coverage, and response checking coverage as a possible framework into which a variety of interpretations and usages can fit. The most common usage of the term coverage refers to the stimuli coverage in the world of simulation-based verification. It is equally important to pay attention to response checking coverage and observed coverage to determine the effectiveness of a verification process.

The notion of combining coverage from different verification methods by simply combining some coverage metric numbers is too simplified, and there is no known practical and technically sound way to achieve this that can lead to verification productivity. Different verification methods can be leveraged in a top-down way by appropriately planning them in the verification plan, and in a bottom-up way by using the methods to provide results for each other. The important point is to plan the verification strategy so as to maximize the strengths of each of the verification methods that are available to you, and then to measure coverage against that plan.

Rajeev Ranjan is CTO of Jasper Design Automation. Jay Littlefield is director of technical marketing and custom support at Jasper. Brian Bailey is an independent consultant with Brian Bailey Consulting.

References

[Bailey2007] Bailey, Brian, The Great EDA Cover-up. ElectronicSystemLevel.com
<http://electronicsystemlevel.com/phpBB/viewtopic.php?t=320>

[Certess2007] Certess releases Certitude for verification QA. EETimes 05/07/2007
<http://www.eetimes.com/news/design/showArticle.jhtml?articleID=199204075>

[CKV2003] H. Chockler, O. Kupferman, and M. Y. Vardi. Coverage metrics for formal verification. Correct Hardware Design and Verification Methods (CHARME), pages 111-125, 2003.

[CLA2006] Classen, Koen, A Coverage Analysis for Safety Property Lists, Workshop on Designing Correct Circuits (DCC), March 2006, <http://citeseer.ist.psu.edu/587670.html>,
<http://www.cs.chalmers.se/~koen/pubs/dcc06-coverage.pdf>

[DeMillo1978] DeMillo, Lipton and Sayward. Hints on test data selection: Help for the practicing programmer. IEEE Computer. (Apr) 1978.

[Fallah2001] Fallah, Devadas and Keutzer, OCCOM—Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification, IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, VOL. 20, NO. 8, August 2001


[GamePlan2007] <http://www.jasper-da.com/gameplan>

[HKHZ1999] Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model

checking. Proc. 36th Design automation conference, pages 300-305, 1999.

[JasperGold2007] http://www.jasper-da.com/products_jaspergold.htm

[TK2001] S. Tasiran and K. Keutzer. Coverage metrics for functional validation of hardware designs. IEEE Design and Test of Computers, 18(4):36-45, 2001

 Add Comment - please [log-in](#) to comment

SCDsource newsletter subscribers may post a comment - [Register for free!](#)

[Back to Home Page](#)

All materials on this site Copyright © 2007-2009 Tech Source Media, Inc. All Rights Reserved | [Privacy Statement](#)