Coverage Interoperability and Unification
What can be expected from the Accellera Unified Coverage Interoperability Standard?

Richard Ho, Ambar Sarkar, Mike Burns, Rajeev Ranjan

1. Introduction

Standards are created for various reasons, but when the user community asks for one, they demonstrate the degree of pain they feel. One such pain point is verification closure and the need for metrics that can reliably predict design quality, and degree of confidence in the verification process. Users are faced with formidable challenges including a vast array of coverage types coming from multiple tool classes, from multiple levels of abstraction and from various stages in the verification process.

The Accellera Unified Coverage Interoperability Standard (UCIS) committee has been tasked with developing solutions for these problems. The need for a Unified Coverage Database (UCDB) was defined along with an Application Programming Interface (API) to store, access and manipulate data contained within the database. This involves a number of technical challenges including naming conventions, coverage semantics, merging of data and integration of formal and simulation data that are being address by the committee.

Goals

The ultimate goal, for a user, is a metric that defines verification completeness. This is defined differently by every company and development team and defies consistency or unification. It is not the objective of the UCIS committee to standardize this. Typically, measures of verification progress or completeness are called coverage goals or targets and the overall quality is stated as the completion of all coverage goals. The ideal situation is to take multiple types of coverage goals and combine them into a single metric. The definition of those coverage metrics needs to be consistent within the industry such that data collected from one simulation vendor is compatible with similar data from a second vendor.

In order for a user, or tool, to be able to create such completion metrics, they must have access to all of the necessary data in a form such that the meaning of the data is completely unambiguous. This is the goal of the UCIS committee, and will enable the maximum amount of creativity in tools and methodologies to satisfy the user goals.

2. Technical Challenges

In this section, we will briefly overview some of the challenges and the approaches that are being taken to solve them.

Naming of Objects

In order to have multiple applications perform the creation, merging and display functions, the syntax and semantics associated with the data have to be unambiguous. For example, while it

may be clear what name to use to refer a specific line of code, an expression within a line of code, or even the name associated with a covergroup bin, not all coverage objects are as simple. Consider the name that should be associated with a "bin" of signal values used in expression coverage. Unnamed scopes and generated scopes also present naming problems. A significant requirement for the UCDB is performance such that, for example, simulation is not unduly slowed down, and results can be analyzed quickly. This requires fast access mechanisms into the data which is usually accomplished using a unique object ID or pointer. However, these mechanisms are generally not portable, and named based techniques too slow. Thus ID based techniques are used within a vendor's implementation, but translation tables are provided for portability.

Semantics of metrics

A common complaint, from users is that they obtain different coverage results from different vendors. Some metrics come from industry standards, such as the SystemVerilog covergroups, but many metrics evolved from proprietary verification methodologies. Consider assertion coverage. No two vendors have the same interpretation of this. After a company has invested time and effort into such a capability, and their customers have been using it, changing the way they are measured is problematic.

Even after the semantics of common coverage metrics are standardized, there remain implementation dependencies that are difficult to remove. Consider expression coverage: it is possible to standardize exactly which expressions are included, when coverage data is gathered, when the values are sampled, and everything necessary to understand the meaning of the data; however, when attempts are made to merge the data from separate tests run on different standard-compliant tools, it is possible that expression values were "binned" differently by each tool. Thus, if the expression is "xyz", one tool may decide that "x=1, y=1, z=dontcare" is a bin for which coverage data is collected, while a second tool defines "x=1, y=dc, z=1" as a bin. These two bins partially overlap; the remaining bins may also display various amounts of overlap. In general, coverage data utilizing different bin definitions cannot be merged without loss of resolution.

Standardizing binning would seem to be a solution, but also presents difficulties. Users or tools may want to divide the input space of a given expression according to design-specific characteristics, such as word length or page boundaries. Restricting the bin structure to that allowed by the standard would confound this effort.

Merging of Coverage Data

Coverage data merging is primarily a combination of the three operations:

1. Temporal Merging. Merging of coverage data across multiple runs of the same verification process. Conceptually, the same design under verification with the same coverage items are being exercised through different scenarios.
2. Spatial Merging. Merging of coverage data across different parts of a design. For example, some verification may occur on subsystems of a design independently. This data needs to be merged with verification data for the full design or some overlapping section of the design to incorporate all coverage across all verification runs.

3. Heterogeneous Data Merging. Coverage data from different verification processes may be semantically equivalent, but more frequently, they are different. It is desirable to merge data from formal verification tools with simulation coverage data, as well as being able to merge data associated with multiple types of simulation coverage metrics.

The standard will not provide merging functions except for basic temporal merging. Merging is the responsibility of the application developers. If a company devises a way to merge data from two different tool types, or from two different coverage types, they would create a new custom coverage type and populate it with the merged data. All reader tools should be able to display this new coverage type and use it for basic completion criteria.

Setup and Test Information

For every test run, or formal proof, there is data related to system initialization: what files were used, the random generator seed etc. Setup conditions are difficult to standardize and in general are not transferable between vendors. Consider the case of a constrained random stimulus generator. At the heart of this generator is a randomization function. At the start of simulation, a seed is used to initialize the random number. Each time that same seed is used, the same series of random numbers is produced, making the run repeatable. But that is within a vendor's tool. Unless all vendors use the same random number generator they will produce different results from that seed.

Combining Formal and Simulation

Simulation-based verification is a sampling process of the possible input sequences that could be applied to the design. Given that this is a heuristic process, it is important to measure the completeness of the input sequences. Several coverage metrics are used in practice to measure the stimuli completeness including:

1. Code coverage (line, conditional, expression, block)
2. Finite State Machine (state coverage, transition coverage)
3. Functional coverage points

Formal verification is proving or disproving the correctness of a design by showing that it conforms to a desired specification of behavior, expressed by a set of properties. Verification completeness of a design using formal techniques is dependent on:

1. The completeness of the property set against which the design is formally verified,
2. The accuracy of the constraints, and
3. Whether analysis was complete or partial. An incomplete proof means the computation was not able to complete either because the problem had to be bounded or due to resource constraints such as time or memory.

Measuring the exhaustiveness of a specification in formal verification ("do more properties need to be written?") is different from measuring the exhaustiveness of the input sequences in simulation-based verification ("do more input sequences need to be created?"). Traditional metrics used in simulation, such as structural or functional coverage are not applicable for formal.

Consider line coverage. In simulation it says that the line of code has been activated one or more times. It does not say whether any checkers exist to verify that the line of code operated

correctly. For formal, all lines of code are considered for each property, and some code may be proven to be unreachable. If a proof is obtained it says that the line of code produces no behaviors that are in contradiction to the property being proven. Simulation coverage metrics relate to how much of the input space has been covered, whereas formal metrics relate to how much of the state space has been covered. It is not technically sound to take metrics from the two verification methods and combine them. Tool vendors are working on solutions to these problems, such as the creation of methodologies that effectively combine the two verification techniques. These will be discussed shortly.

Extensibility

While it is one thing to agree on things that already exist, standards need to be extensible, otherwise they can impede innovation and progress. For this to work, extensions must be treated fairly and not penalized in terms of capabilities or performance. This was a requirement demanded by the committee members and discussions continue within the UCIS committee to decide the manner in which this will be achieved.

Interchange Format

Standardizing the API and aspects of the underlying data model are a significant step forward in unifying coverage. However, desirable use cases remain that cannot be realized with these alone.

Consider the creation of a custom merging tool. Each vendor implements the standard API, so the developer of the custom tool does not need to interface with multiple APIs. However, each vendor still has their own nonstandard representation of the underlying coverage data, since no standard representation has been defined. Each vendor's implementation of the API is therefore required to access its own data.

The solution is a standard interchange format – a complete specification of the data model and its representation that can be understood outside the context of the tool that generated it. Such a standard would elevate coverage data to the level of a language, such as HTML, that can be generated, shared, and understood by anyone with minimal dependencies on any vendor.

In practice, this is difficult. First, there is little industry experience with sharing coverage data between vendors and without this, a standards committee has little hope of delivering a viable specification. Second, a standard representation greatly restricts the ability of vendors to optimize. Managing performance and resource usage is particularly critical with coverage data due to the massive amount of data that is generated in a typical design flow. Third, a mechanism for representing vendor-specific data is still required. New coverage metrics, setup information, and other unanticipated data will need to be tracked as well. The existence of such data would bring back the dependency on the vendor, reducing the value of the standard and the ability to share the data.

3. Realistic Goals

Having demonstrated the magnitude of the problems, many high-valued aspects of the problem can be solved in the short term. At the heart of the standard is a UCDB. The standard will not define how that is to be implemented, only the API into it and the data that it holds.

There are three possible types of users for the database, those that create data, those that analyze and display data, and those that manipulate data. The creation of the API conceptually allows all

three of these functions to be provided by different companies or individual users so the best in class capabilities could be selected for each task.

The standard API enables coverage data to be collected and accessed from many sources. Today, temporal merging is fully supported and attempts are being made to ensure that other types of merging are supported such that data could be merged from simulations of different hierarchies of a design, or that simulation coverage requirements could be reduced based on formal verification results.

4. Suggested Usage Models

The first release of the standard will support several use models, some of which are described here.

Top Down

A common practice is to create a verification plan that partitions the problem into a number of technologies or stages. Each of these partitioned verification goals is considered a target, and may be approached using one of many verification technologies. For example, the plan may state that certain functionality is to be verified using formal property checking, while other related functionality will use simulation. Each of these goals can then be approached by the creation of appropriate properties, or the creation of a testbench. The total completeness is the aggregate of the completeness of the partitioned goals.

Bottom Up

Formal analysis can be used to produce specific scenarios aimed at hitting particular coverage points, or to show that certain simulation coverage points are unreachable. This allows the formal engines to "guide" the simulation process. This technique must be used with care because while formal verification identifies correct and incorrect behaviors, simulation separates the notions of verification and completeness. If formal is used to *hit* coverage points, simulation may not adequately verify that the functionality is correct.

Data Merging

The standard creates the opportunity for an independent company or user to provide unique data merging capabilities. For example, if they devise a method to merge data across the design hierarchy, a standalone utility could be created. This creates an opportunity for innovation.

5. Progress towards a Standard

Significant progress has been made by the committee to resolve the technical issues. The base donation from Mentor Graphics provided a good starting point, but practitioners of verification wanted something from UCIS that would enable innovation for coverage reporting and analysis. While it is clear that the ultimate user goals are not all within reach for the first version of the standard, many of the issues discussed in this paper, such as the naming conventions, extensibility, formal API and coverage semantics are expected to be resolved. The work will then continue, until they can achieve what users have wanted from the beginning – a metric that clearly measures progress towards verification goals, identifies the risk associated with a release, and improves the efficiency and effectiveness of their verification efforts.