Coverage Data Exchange is no robbery...or is it?

Darron May and Samiran Laha, Mentor Graphics Inc
Darron_May@mentor.com; Samiran_Laha@mentor.com

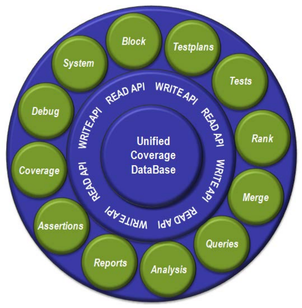
Abstract

Coverage is extremely important to the modern verification flow. Most vendors have already figured that unifying data across all verification engines leads to a more efficient and integrated environment. There are many challenges to be solved unifying and sharing data across a single vendor's tool set which are further complicated when wanting to share data across multiple vendors' tool sets.

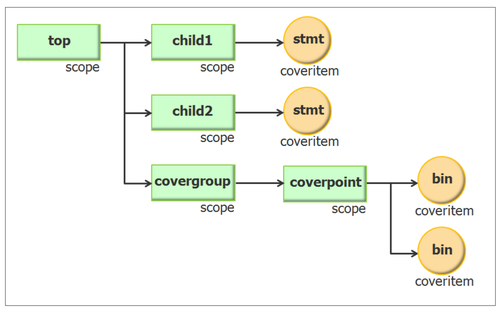
The Unified Coverage Interoperability Standard (UCIS) was created to make data sharing and exchange easier by standardizing data models and data exchange. UCIS uses Extensible Mark-up Language (XML) as an exchange format. This paper introduces a method of exchanging data between two-UCIS compliant coverage database systems without the need for inefficient formats like ASCII or XML.

Mentor's Unified Coverage DataBase (UCDB)

Architected in 2005 to unify coverage collection across all verification engines, UCDB was first released within Questa and ModelSim in early 2006 as a way of natively storing, analyzing and reporting on functional coverage, code coverage and assertions.



Coverage databases however they are implemented contain very similar information, at the very lowest basic point it contains counts of defined events. The way these defined events are labelled or arranged within the database with a set of building blocks is defined as the data model for the particular metric.

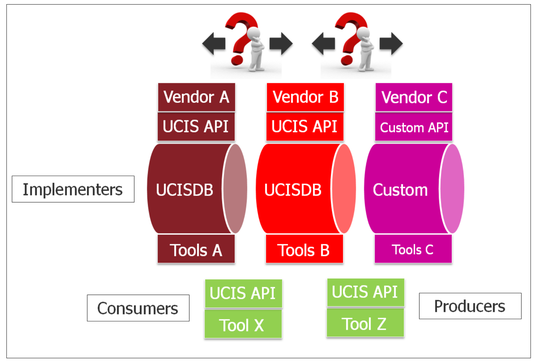


What UCIS provides the industry

Accellera's UCIS subgroup was formed in 2006 with three defined goals combined to encourage user and EDA technology advancement.

- Identify interoperability opportunities for coverage
- Define standard coverage models for the industry
- Define an operability standard for data exchange

Mentor Graphics donated the UCDB API to Accellera and it was chosen as the basis of the UCIS standard where version 1.0 was released in June 2012.



When a vendor supports UCIS users often assume that the implementations are binary compatible, which is not correct. Figure 2 below shows the different combinations of UCIS support; note that the implementers of coverage databases require a persistent form of the data to be stored in an implementation-specific manner. (In the case of Mentor this is a UCDB file stored on a disk.)

UCIS interoperability

UCIS defines all known coverage models and its recommendations on how these models should be represented with the scope and coveritem building blocks plus attributes and flags. The standard provides an XML exchange format allowing vendor A to output XML in a defined format and vendor B to read this XML format and import the data into its database. There are a number of problems that this presents.

1. XML & UCIS Data schemas non-overlap
2. XML files for average sized DB's are huge
3. Data types across Vendors require adaption
4. Linking two UCIS Apps create symbol clashes
5. Vendors do not have access to other Vendors APIs
6. Some Vendors support little or none of the UCIS API
7. Requirement for no intermediate formats

The use of XML

Tables below shows the performance and capacity comparisons between using an XML format and an internal binary format for some typical user designs in the range of 4 to 17 million bins.

TABLE I
SOME TYPICAL DESIGNS WITH DATABASE SIZE AND PERFORMANCE MEASUREMENTS

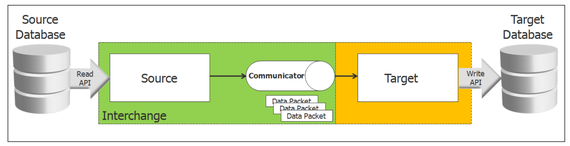
Design	Measurements				
	Size (Millions Bins)	UCDB Size (Mbytes)	XML Size (GBytes)	UCDB size / XML size	Compressed XML (Mbytes)
1	4.5	14.0	1.8	128.5	37.0
2	5.8	19.7	2.3	116.7	68.8
3	7.9	42.5	3.9	91.7	106.0
4	17.4	63.0	19.2	304.7	254.4

From the table we can see that the XML files are on average 160 times larger than the binary representation of a design. With design sizes increasing, tests numbering in the thousands to tens of thousands, and many regressions run on a daily basis, the result can be a huge amount of data. Even compressed XML files (note that compression is not part of the UCIS process) are on average three times larger, not much of a relief considering that this approach also adds an extra burden of compressing/uncompressing any time the data needs to be processed.

Design	Measurements			
	Size (Millions Bins)	XML Comp / UCDB size	Gzip time (seconds)	Gunzip time (seconds)
1	4.5	2.6	8	15
2	5.8	3.5	48	24
3	7.9	2.5	53	37
4	17.4	4.0	262	233

Data exchange using the API

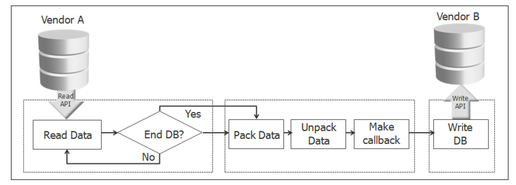
Exchanging data without the need for XML is straightforward when the two implementations exchanging data have full implementations of both the coverage models and API as defined within the standard. Below the graphic shows the basic blocks of a UCIS-to-UCIS exchange application; it requires both sides to support the data models for the coverage metrics being exchanged.



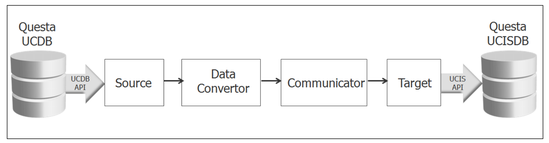
The 'Source' block reads data from the source database using the vendor-specific UCIS API, which created the source database. The 'Source' block is created by linking the UCIS C library supplied by the vendor which contains the definition of UCIS API routines for that vendor. By linking the application against other vendor-supplied UCIS C libraries, the same application can be used by any implementation. Data read by the 'Source' block is passed to the 'Communicator' block by calling some specific C routines defined by the 'Communicator' block.

The implementation

The implementation allows customization on both the read and write sides of the exchange.



Some coverage vendors do not support the UCIS API so there is a need to allow mixing of APIs and the addition of extra customization code to make adjustments between the two formats being exchanged. The architecture of this exchanger is able to plug and play with any coverage API on either the source of target sides.



This means that the differences in the data representation and data models have to be taken care of within the 'Target' and 'Source' blocks. For example the source side could be replaced with the UCAPI API to allow Synopsys coverage databases to be read, and the target side replaced with the Unicov API to allow a Cadence coverage database to be written.

Problems supporting differing coverage models

Functional coverage can be adjusted between coverage database implementations due to the fact that they are representing the same SystemVerilog standard. But as there are no standards for code coverage, this may produce more difficulties as different vendors implement different data models. Here is a possible list of different code coverage data models which could be very hard to transform from one to another.

1. Statement and Branch coverage: Line coverage vs. block coverage
2. Expression coverage: Flat list of nodes and variables vs. hierarchically represented sub-expressions
3. Toggle coverage: Simple transitions among 0, 1, and Z vs. merging Z with 0 or 1 based on some mode

Conclusions

A basic exchange of coverage data between vendors using XML is limited not only with respect to performance and capacity, but also in that data models can vary even between vendors. Therefore a different method is required that allows adjustments to be made during the data exchange process.