The use of XML

Tables below show 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 bits.

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.

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 implement the data models for the coverage metrics being exchanged.

UCIS interoperability

UCIS defines all known coverage models and its recommendations on how these models should be represented with the scope and covertest 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. XML files for average sized DB’s are huge
4. XML formats are more efficient than binary formats
5. XML files are not human readable
6. XML files are not human readable

The implementation

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 or target sides.

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.