We address here Assertion-Based Verification (ABV) for complex SoCs modeled in SystemC TLM (Transaction Level Modeling). ISIS [1] is a tool for the runtime verification of PSL assertions [2] for TLM platforms: assertion checkers (monitors) are automatically generated from PSL properties, and the design is instrumented with these monitors. Then, the SystemC simulation is run as usual, and the assertion checkers report property violations, if any.

We demonstrate here an extension of this tool that enables the designers (in particular the software developers) to customize and to optimize the verification process, and to get concise and easily analyzable verification results. This new version fulfills the following requirements:

- the software designer uses the SystemC platform to debug the embedded software, but he usually does not wish to recompile the platform. Hence selecting the assertions to be attached to the design must be feasible at runtime (not at the SystemC compile time),
- it must be possible to configure the instrumented simulation to take into account relationships between assertions to dynamically disable/enable monitors, thus clarifying and optimizing the verification,
- it must also be possible to store concise information about assertions activations, satisfactions, violations, and to easily analyze verification results.

**Dynamic Configuration.** ISIS takes as input the platform source files and an XML configuration file that specifies the required properties, and it produces the instrumented design. With the original version, the user must execute again this procedure, and recompile the resulting platform, every time he wants to check different requirements. The tool has been improved to enable the optional runtime instantiation of the assertion checkers, according to choices given in a configuration file.

**Monitors Enabling/Disabling.** Originally, the assertion checkers remain active all along the simulation. However, requirements may be correlated, and the tool has been enhanced with a Verification Manager to enable/disable monitors according to these correlations. By means of a configuration file, the user specifies relations of the form $A_i \Rightarrow A_j$, which express that, if assertion $A_i$ experiences violations, then checking assertion $A_j$ becomes worthless. The Manager can be configured such that, when it detects that the monitor of $A_i$ reports violations, either it only disables $A_i$ immediately (Level 1), or it also disables the monitor of $A_j$ (Level 2). This has two main advantages: simulation traces only include the most relevant information provided by the checkers, thus simplifying the interpretation of the verification results; the CPU time overhead induced by the checkers may be minimized.

**Database with Verification Results.** To ease the analysis of the verification results, in addition to the textual reports, verification results are now stored in a database. During a simulation, the monitor stores in a vector of database entries the information about every assertion activation: start time, end time, status (pass or fail). This information is ultimately committed to the database. A post-processing tool extracts a concise and easily analyzable tabular representation.

The case study used for the demonstration is an image processing platform that performs spectral compression [3]. An IO module generates periodic IRQs to Leon_a which configures DMA_a to copy data to Mem_a. Leon_a subsamples the data and writes the result to Mem_a. It configures DMA_a to copy the results to Mem_b. Leon_b configures the FFT module to perform 2D-FFT, and compresses the obtained spectrum. Compressed data are transferred by DMA_b to the IO port. Here are some of the associated requirements:

- $A_1$: DMA_a must not be reconfigured before the end of a transfer.
- $A_2$: The FFT must not be reconfigured before the end of a computation.
- $A_3$ (no loss of input data): Every input data packet must be transferred by DMA_a to Mem_a before the IO module generates a new interrupt.
- $A_4$ (software not too slow): Every data packet stored in Mem_a must be processed by Leon_a before being overwritten.
- $A_5$ (no loss of processed data): Each incoming data packet must have a corresponding output packet, before 3 new incoming data packets are processed.

Conceptual dependencies are identified between these assertions. For example, if $A_3$ is violated (input data are lost), the results of $A_4$ become irrelevant because paying attention to the loss of output data is useless. Property $A_2$ however might still be enabled, to continue checking the interactions between Leon_b and its FFT coprocessor.

**REFERENCES**