

Assertions Based Verification

It's known that close to 70% of project time for chip design is spent on verification. Assertion based Verification is a methodology that helps come out of this tiring, tedious, error-prone, long-time problem.

Assertion based Verification is something that has been used in the industry since long, though it was never standardized and named. In its simplest form, Verilog users coded conditional loops to check for certain erroneous conditions or used "\$display" for monitoring and VHDL programmers, used assert statement.

Having said this, the three basic questions that arise are:

1. What are Assertions?
2. How they work.
3. Why Assertion Based Verification?

What are Assertions?

Assertions are design checks, either embedded deep-down the design by a designer to catch the bugs early on or some pin-checks written by a verification engineer for black-box testing of the code. They express all information about the functional/behavioral nature of the block just as the designer intended it to be used. They can also be thought of as internal software test points that wait for a particular predefined condition to occur and then notify the designer about the occurrence of the same. This is much better since the bugs are reported at the cause point and not the effect point, saving us from tedious backtracing of waveforms. Without assertions, test vectors had to be much longer to ensure that triggered bugs were propagated to observable outputs, else the errors remained undetected. By adding assertions, bad behavior inside the design could be checked and the bugs were observed instantly, at their source.

Assertion-based interface specification can also be used to constrain pseudo-random simulation and formal analysis. This ensures that the circuit will work properly under all legal input stimuli, and characterizes behavior in the presence of illegal input sequences.

The biggest advantage of assertions is that they become a part of the design. Modern assertion methods enable you to specify assertions and monitors inline with the HDL code for the module. Placing assertions inline is key because inline assertions cannot get lost.

Assertions were also used at the design boundaries to provide checks for interface behavior. This was useful as design modules from different designers are integrated. Such interface-checking assertions made modules more portable; they preserved the intimate design knowledge needed to verify them as part of a larger system. As a result, modules with embedded assertions became self-checking wherever they were later reused. The net benefits resulted in finding problems sooner, spending less time analyzing them, reaching a stable design verification point sooner, and simplifying integration and design reuse.

Generally, there are two kinds of assertions. Concurrent assertions state that a given property must always be true throughout a simulation, while procedural (sometimes called temporal) assertions apply only for a limited time, or under specified conditions.

How they work.

Assertions are design checks that continuously get evaluated in form of an expression. When any expression does not hold true, an assertion flag is raised. This allows the verification engineer to directly go to the problem instead of tedious back-tracing of waveforms or analyzing a MegaByte large log file.

Assertions can be utilized in various ways. They can be included directly within the hardware-description language (HDL) code that comprises the register-transfer level (RTL) description of the design. Or, they can be applied from outside in the form of test benches, or collections of test vectors, to check the response of the design to stimulus, or to control a stimulus generator or model checker.

In the design, assertions can be developed by the designer and embedded in the design so that typical logic bugs can be found early on in the project. The Verification engineer typically creates interface level assertions (Black Box testing) to check for the defined relationship between all the inputs and outputs of the chip.

Advantage with embedded assertions is that the bug shows up at the point of cause and not a million clocks later, when it propagates to the output of the chip. Without assertions, there are chances that the bug is nipped in the bud by the sub-module in it and is not all propagated to the output. If detected at the later stages of the design, this could prove lethal.

Why Assertion Based Verification?

Using assertions to specify interfaces is far superior to natural language and waveforms because assertions are unambiguous and executable. The clarity of assertions comes from having well-defined syntax and semantics. Assertion languages enable us to clearly specify any interface, but often require significant coding to describe complex interfaces and RTL components, which is definitely worth it, since it would take months off the project schedule.

There are different languages available for Assertions, such as Sugar from IBM, OVA (Open Vera) from Synopsys and OVL (Open Verification Library) from Accellera. Currently, Sugar has been accepted by Accellera as a basis for standardized Property specification Language (PSL). Since PSL is now standardized by the IEEE, it will be an added benefit for assertion users.

Assertion-based verification benefits users by:

- Simplifying the diagnosis and detecting bugs by localizing the occurrence of a suspected bug. It thereby reduces simulation-debug time significantly.
- Self-checking code helps a lot in reuse of design
- Interface assertions help find the interface issues early on