DAY-78
#100DAYSRTL

"System Verilog:-Assertions Intro"

"Introduction":-

System Verilog revolutionized hardware description languages by introducing powerful assertion constructs. These built-in assertions, including constraints, streamline the creation of intricate designs, enhancing readability and fostering consistency in System Verilog code. This advancement facilitates the seamless development of complex hardware systems with improved reliability and efficiency.

"Assertions":-

- In hardware verification, checkers traditionally focus on validating design functionality by comparing outputs to expected results.
- However, they often fall short in assessing design behavior. Assertions become crucial in such cases.
- For instance, if a specification mandates that an output toggles precisely after 2 clock cycles of a valid input, assertions enable precise behavior verification.
- By embedding assertions in the testbench, designers can ensure adherence to specified temporal and logical conditions, enhancing the verification process.
- This approach goes beyond mere output validation, providing a robust means to verify intricate behavioral requirements and promoting more comprehensive and accurate verification methodologies in hardware design.

"Advantages of Assertions":-

- ✓ **Precision in Issue Localization:** Assertions capture design flaws at their source, minimizing debugging time.
- ✓ **Targeted Debugging:** Assertion failures identify specific signals, streamlining the debugging process.
- ✓ **Functional Coverage:** Assertions serve as effective tools for covering complex behavioral properties.
- ✓ **Formal Verification Integration:** Assertions play a crucial role in Formal verification, a distinct and assertion-centric approach.

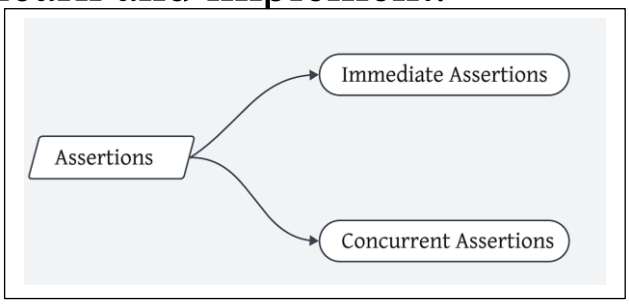
"Different Assertions Languages":-

- ➢ PSL (Property Specification Language) – based on IBM Sugar
- ➢ Synopsys OVA (Open Vera Assertions) and OVL (Open Vera)
- ➢ Assertions in Specman
- ➢ 0-In (0–In Assertions)
- ➢ SystemC Verification (SCV)
- ➢ SVA (System Verilog Assertions)

"System Verilog Assertions":-

Prior to System Verilog, different assertion languages were used due to which there was no uniformity in the assertion constructs. There was a need to natively support assertions in HDL itself due to the increasing complexity of the verification.

- o System Verilog merges benefits from different assertion languages into one, thus giving the best from all languages.
- o Due to native support of assertions in SV, assertions can be added to the design and testbench directly without needing to add special interface.
- o SVA is easy to learn and implement.



“Immediate Assertions”:-

An immediate assertion is a type of instruction that tells a simulator how to check the correctness of a design under test (DUT). It follows the simulation event semantics, which means it will be evaluated in the event regions based on how assertion is used. It is written as a procedural statement, and it is executed like any other statement in a procedural block.

- Immediate assertion as a basic statement that is evaluated for a single time.

```
assert(expression)pass_statement[else fail_statement]
```

- The expression is non-temporal, and it is treated as a condition in an if statement.
- The pass_statement is executed if the expression is true, and the fail_statement is executed if the expression is false.
- The else block is optional, but it can be used to specify the severity level of the assertion failure.

“Concurrent Assertions”:-

Sequence is the basic building block of concurrent assertions in SV. A sequence is a series of Boolean expressions that specify the order and timing of events that form a property. A sequence can be written as a simple sequence, which consists of a single expression, or as a complex sequence, which combines multiple expressions with operators.

- **Property:-** A property is a logical expression that describes a condition or a sequence of events that should hold true for the design. We can use sequence in a property or even directly use an expression, if the expression is simple.
- **Assert directives:-**As we have seen assertion can be used in formal verification as well as coverage, thus there are multiple assert directives provided by SV.
- **assert:** This statement specifies that a given property of the design should always be true in simulation. If the property

evaluates to false, the assertion fails, and an error message is generated.

- **assume:** This statement specifies that a given property is an assumption that guides the formal verification tools to generate input stimulus. The assumption is not checked in simulation, but it can be used to constrain random variables or generate cover points.
- **cover:** This statement evaluates a given property for functional coverage. If the property evaluates to true, a cover point is hit, and a coverage report is generated.
- **restrict:** This statement specifies that a given property is a constraint on formal verification computations. The property is ignored by simulators, but it can be used to prune unreachable states or avoid invalid scenarios.
- **Delay operator:**-Concurrent assertions have temporal expressions, thus we need some way to provide delay in terms of the clock for the expression. ## is the clock cycle delay operator. The delay provided by this operator is not based on time scale but rather based on the clock which is inferred by the assertion property.

```
Eg:- ##1 means delay of one clock cycle
```

"Code Practising":-

```
always @(posedge clk) begin
    // some logic
    sequence seq1 ();
        req ##1 ack;
    endsequence
    assert property (req ##1 ack); // check that ack follows req within one clock cycle
    cover property (seq1); //like above assert property but this time will be used for coverage as cover directive is used.
end
```