1. What are assertions?

Assertions are statements in hardware verification languages (such as SystemVerilog) used to specify conditions that must always be true during the simulation or execution of a design. They are a way to check the correctness of the design by ensuring that certain properties or behaviors hold at runtime. Assertions can be used to detect errors, violations of design rules, or unexpected behavior in hardware designs.

1. Why are assertions used?

Assertions are used for:

* Verification: To verify the correctness of the design by checking certain conditions at specific points in the simulation.
* Error Detection: To catch design errors and bugs by monitoring properties in the design, such as incorrect signal values, missed timing, or invalid states.
* Design Intent: To express the designer's intended behavior in a formal way that can be checked automatically.
* Debugging: To help identify where in the design things go wrong by providing error messages when a condition is violated.

1. What are the types of assertions?

* Immediate Assertions: These assertions are evaluated immediately when the statement is encountered. They are checked at the exact moment they are executed.
* Concurrent Assertions: These assertions monitor events that happen over time and are evaluated over multiple simulation cycles.

1. In which event region will concurrent assertions be evaluated?

Concurrent assertions are evaluated during the simulation in the concurrent region, meaning they are evaluated at every clock cycle or event in the simulation. Typically, they are triggered by an event, such as a clock edge or a signal transition, and can span multiple time steps or cycles.

1. What is a property in SVA?

A property in SystemVerilog Assertions (SVA) defines a sequence or behavior that is expected to happen over time in the design. Properties are the building blocks of concurrent assertions. They describe the expected relationships between signals across time, such as sequences of events or conditions.

1. What are the advantages of assertions?

* Automated Verification: Assertions automate the process of checking the correctness of the design and make it easier to detect and diagnose issues.
* Design Intent: Assertions help document the expected behavior of the design, making it easier for others to understand the designer's intent.
* Reduced Debugging Time: Assertions can quickly highlight when and where the design violates its intended behavior, thus reducing debugging time.
* Improved Quality: Assertions help ensure the design behaves correctly during simulation, which leads to higher-quality designs and fewer errors in later stages of development.

1. What are immediate assertions?

Immediate assertions are evaluated immediately at the point in time when the statement is executed. These assertions are used to check conditions in the current time step or simulation cycle.

1. What are the main components of Concurrent assertions?

* Events: An event (like a clock edge or a signal transition) triggers the evaluation of the assertion.
* Properties: A set of expected conditions or behaviors that must hold true over time.
* Sequences: A sequence is a set of ordered events or conditions that describe the behavior over time.
* Assertion: The concurrent assertion that checks if the property or sequence holds true during the simulation.

1. What is the difference between Concurrent and Immediate assertions?

* Immediate Assertions: Evaluated immediately at the time they are encountered in the simulation, checking conditions for the current cycle.

Example: assert(a == 1);

* Concurrent Assertions: Monitors conditions over multiple simulation cycles. These are evaluated across multiple time steps, which can involve events spanning over multiple clock cycles.

Example: assert property (p1);

1. Write an assertion that checks when the “ready” signal is “1”, the “grant” signal is asserted within 7 clock cycles or else displays an assertion failure error.

assert property (@(posedge clk) disable iff (!reset)

(ready == 1) |-> ##[0:7] grant == 1)

else $fatal("Grant not asserted within 7 clock cycles after Ready");

* @(posedge clk): The assertion is checked at every positive edge of the clock.
* disable iff (!reset): The assertion is disabled when the reset signal is active.
* |-> ##[0:7]: This specifies a "non-immediate" assertion, where grant should be 1 within 7 clock cycles after ready becomes 1.

1. What is the syntax for ## delay in assertion sequences?

In SystemVerilog assertions, the ## operator is used to specify a delay in cycles. It indicates that the property should hold after a specified number of cycles from the current time.

1. Difference between @posedge and $rose?

* @posedge is a sensitivity control used to trigger an action on the positive edge of a signal, typically in procedural code or within assertions.

Example: @(posedge clk) triggers at each positive edge of clk.

* $rose(signal) is a system function that checks if the signal has transitioned from 0 to 1 in the current time step (i.e., rising edge detection). It can be used inside procedural blocks or assertions.

Example: assert ($rose(clk)) checks if clk is rising in the current time step.