1. What are Assertion severity system level task? What happens if we wont specify these tasks?

In SystemVerilog assertions, severity levels indicate how critical a failure is and determine the action taken when an assertion fails. The severity system-level tasks are used to handle assertion failures and specify the level of error reporting.

The System Verilog assertion severity tasks are:

* $fatal: Terminates the simulation immediately. This is used for severe errors that should stop simulation.
* $error: Reports an error message and continues the simulation. This is used for errors that don’t immediately halt simulation.
* $warning: Reports a warning message and continues the simulation. This is used for less critical issues.
* $info: Outputs an informational message without affecting the simulation flow. It’s used for logging purposes.

If severity tasks (e.g., $fatal, $error, $warning, $info) are not explicitly specified in an assertion, the default severity action is taken:

* If an assertion fails, the simulator will generate a message but continue the simulation unless disable or fatal is specifically invoked.
* The failure might be logged, but without a specified severity, the behavior may not be as informative or as controlled as with custom task calls.

1. What is a Consecutive Repetition Operator in SVA?

The Consecutive Repetition Operator ([\*]) is used in SystemVerilog assertions (SVA) to indicate that an event or condition must hold for a certain number of consecutive cycles. It is particularly useful for checking repetitive conditions over multiple time steps.

Example:

property p1;

@(posedge clk) a == 1[\*2]; // 'a' must be 1 for two consecutive cycles

endproperty

1. What is “goto” Replication operator in SVA?

The Goto Replication Operator ([->]) is used to specify a delay in the evaluation of the sequence of events. It allows you to repeat the evaluation of a sequence, delaying the evaluation for a given number of cycles or a condition.

Example:

property p2;

@(posedge clk) a == 1 [->3] b == 0;

endproperty

This property means that after a becomes 1, b should become 0 after 3 cycles. The use of [->3] introduces the delay in evaluating the condition.

1. What is the difference between x [->4:7] and x[=4:7] in SVA?

* x[->4:7]: This is a replication operator used in SVA. It means that x should be true for 4 to 7 consecutive clock cycles. It checks if the condition holds true over multiple consecutive cycles.
* x[=4:7]: This is a value-range check. It means that x should have values between 4 and 7 inclusive, and the value of x should be checked to see if it lies within that range.

1. What are implication operators in Assertions?

Implication Operators in assertions are used to express conditions where one event implies the occurrence of another. The implication operator is represented by |-> (rightward implication) or ->| (leftward implication).

* Rightward Implication (|->): This is used to specify that if the condition on the left occurs, then the condition on the right must follow within the specified time window.

Example: assert (a == 1) |-> (b == 0);

This means if a equals 1, then b must equal 0

* Leftward Implication (->|): This is used to specify that if the condition on the right occurs, then the condition on the left must hold.

Example: assert (b == 0) ->| (a == 1);

This means if b equals 0, then a must equal 1.

1. Write an assertion for glitch detection.

A glitch is a brief, unintended transition in a signal. To detect a glitch, we can check that a signal should not change from 0 to 1 and then back to 0 within a short time (e.g., within one clock cycle).

assert property (@(posedge clk) !(x == 1 && $past(x) == 0 && $past(x, 2) == 1));

1. Write the assertions for the following scenarios:
   1. in\_a & in\_b should never be asserted together.

assert property (@(posedge clk) !(in\_a && in\_b));

* 1. every req should be acknowledged by ack within 2 clk cycles unless rst is asserted.

assert property (@(posedge clk) disable iff (rst) (req |-> ##[1:2] ack));

* 1. idle state should eventually result in to start state.

assert property (@(posedge clk) disable iff (rst) (idle\_state |-> ##[1:$] start\_state));

* 1. in\_a must not rise if we have seen wait and haven't seen the rdy/yet.

assert property (@(posedge clk) disable iff (rst) (wait\_seen && !rdy\_seen) |-> !in\_a);

* 1. If in\_a is down, out may only rise for one cycle before the next time that in\_a is asserted.

assert property (@(posedge clk) disable iff (rst) (in\_a == 0) |-> ##[1] out == 0);

* 1. Check that in\_a is high for minimum 2 and maximum 7 clock cycles

assert property (@(posedge clk) disable iff (rst) (in\_a == 1) |-> ##[2:7] in\_a == 1);