1. How to override parameters inside the transaction class?

In SystemVerilog, parameters of a class (like a transaction class) are typically defined at the time of class instantiation. To override parameters inside a transaction class, you can redefine the parameters when creating an object of the class.

Example:

class transaction;

parameter int WIDTH = 8;

logic [WIDTH-1:0] data;

endclass

// Creating an object and overriding the WIDTH parameter

transaction #(.WIDTH(16)) txn; // Overriding WIDTH to 16

1. Difference between bounded and unbounded mailbox?

Bounded mailbox:

* A mailbox with a specified maximum size. Once the mailbox reaches its limit, further attempts to put items into it will block or fail.
* It is useful for controlling the size of the communication queue to avoid excessive memory usage.

Unbounded mailbox:

* A mailbox without a specified size limit. It can hold an indefinite number of items until memory is exhausted.
* It is useful for scenarios where you do not want to limit the number of messages that can be stored.

1. Difference between randcase and randsequence?

randcase:

* A construct used for random selection from multiple cases, each with its own probability.
* Similar to a case statement, but it randomly selects one of the randcase branches.
* Example:

randcase

1: out = 1; // 1st case

2: out = 2; // 2nd case

3: out = 3; // 3rd case

endcase

randsequence:

* A sequence of random events or transactions with constraints that define the order and conditions under which they occur.
* Allows the randomization of sequences with specific ordering or dependencies.
* Example:

randsequence seq;

seq = {1, 0, 1}; // Randomize a sequence

1. Why is SV preferred as a Verification language over Verilog?

* Object-Oriented Programming (OOP): SystemVerilog supports OOP, allowing for more modular, reusable, and structured verification environments (e.g., classes, inheritance).
* Randomization: SystemVerilog supports constrained randomization (randc, randcase, etc.) for better test coverage.
* Assertions: Built-in support for assertions, including immediate and concurrent assertions, improves verification quality.
* Interfaces and Modports: SystemVerilog has a more robust way to handle communication between modules through interfaces, making it easier to manage complex interactions.
* Coverage: SystemVerilog supports functional coverage (covergroups) and code coverage, essential for thorough verification.
* Concurrency and Synchronization: With constructs like semaphores, mailboxes, and event controls, SystemVerilog makes it easier to handle concurrency in verification.

1. How is garbage collection done in SV?

* SystemVerilog does not have automatic garbage collection like some high-level programming languages (e.g., Java, Python). However, it supports manual memory management through handles.
* The new operator allocates memory dynamically, and unreferenced objects will be removed when they go out of scope or when explicitly deleted using delete.
* Example:

class MyClass;

int data;

endclass

MyClass obj;

initial begin

obj = new MyClass(); // Allocates memory

delete obj; // Frees memory

end

1. List the synthesizable and non-synthesizable constructs in SV.

Synthesizable constructs:

* always\_ff and always\_comb: Used to describe flip-flops and combinational logic, respectively.
* if-else, case, for loops (when controlling hardware behavior).
* assign: Used for continuous assignments.
* logic, reg, wire: For defining hardware signals.
* Module Instantiation: Creating and connecting modules in a design.

Non-synthesizable constructs:

* initial blocks: Used for testbenches and simulation initialization, not synthesizable.
* $display, $monitor, $finish, and other system functions used for simulation.
* Dynamic memory allocation (new/delete in classes).
* Randomization (randc, constraint): Randomization is only applicable in verification environments, not synthesizable RTL.
* Assertions: Typically for simulation-based verification, not synthesizable.

1. What values does c take in this example?

rand int c;

int lo, hi;

constraint c\_range {

!(c inside {[lo:hi]});

c will take values other than in the range lo to hi, inclusive(lo and hi included in the range)..

1. What values will b,c, and d take in this example?

rand logic [15:0] b,c,d;

constraint c\_bidir{

b<d;

c==b;

d<30;

c>25;

* b will take values greater than 25 and less than 30 provided its less than d.
* d will take values less than 30 and greater than 25 and will be greater than b.
* c will be same as b.