1. What is the need for randomization in Verification?

Randomization is used in verification to ensure that the design is tested under various conditions and corner cases that might not be easily anticipated. It helps uncover potential bugs or weaknesses that would not be identified with fixed test vectors. By randomizing inputs, you can create a diverse set of test cases that exhaustively cover the design's behavior.

1. What are the constraints in System Verilog?

In SystemVerilog, constraints are conditions or rules applied to random variables or classes during randomization. Constraints define valid values or ranges for variables, guiding the randomization process. Constraints ensure that the randomization adheres to the desired boundaries, which could be ranges, specific bit patterns, or relationships between variables.

1. What are the different types of constraints in System Verilog?

* Simple Constraints: These define specific ranges or values for variables, e.g., x within {1, 3, 5}.
* Range Constraints: Specifies the allowable range of a variable, e.g., a inside {1, 2, 3} or a within [0:255].
* Equality Constraints: Enforces variables to take a specific value, e.g., a == 10.
* Complex Constraints: Combination of logical operators like &&, ||, !, etc., to express more complicated conditions.
* Cross Constraints: These are constraints that express relationships between multiple variables, e.g., if (a > b) a < c.

1. What is an if-else constraint?

In SystemVerilog, an if-else constraint specifies conditional constraints based on the values of other variables. For example:

if (a == 1)

b inside {1, 2, 3};

else

b inside {4, 5, 6};

1. What are bidirectional constraints?

A bidirectional constraint involves constraints where one variable affects the other and vice versa. It represents a mutual dependency between two or more variables. For example:

constraint c {

x == y;

y == z;

}

In this case, x, y, and z are tightly related, with each constraint influencing the others.

1. Difference between :/ and := operators in randomisation.

* :/ (Randomization with a constraint): This operator is used to apply a constraint during randomization. It is used when the randomization must adhere to the constraint but does not enforce the constraint if it fails. Example: a :/ b;.
* := (Assignment with a constraint): The := operator is used for assignment in the context of randomization. It allows the value of the variable to be assigned according to the constraints at that point. Example: a := b;.

1. What is the std::randomisation?

std::randomize is a method in SystemVerilog used to randomize class objects and variables within those classes. It allows you to randomize the values of class members while adhering to the constraints specified.

class my\_class;

rand bit [3:0] a;

rand bit [7:0] b;

constraint c { a < b; }

endclass

my\_class obj;

obj.randomize();

1. What is the difference between keywords, rand and randc?

* rand: This method generates a random value for a variable, but there’s no guarantee that the same value won't be picked multiple times.
* randc: This method generates a random value for a variable but ensures that all values are unique. It will continue to randomize until all possible values have been assigned.

1. How do you implement the randc function in System Verilog?

The randc function is built into SystemVerilog. It's used to generate random values with the constraint that no value is repeated until all possibilities have been exhausted.

Example: randc bit[3:0] a;

1. What is the significance of seed in randomization?

The seed in randomization is used to initialize the random number generator to produce a deterministic sequence of random numbers. When you specify a seed for randomization, the random sequence generated will be the same each time you run the simulation with the same seed value. This is helpful for debugging and verifying test cases because it ensures that the same "random" sequence can be reproduced.

If you do not specify a seed, the random number generator may produce different sequences of random numbers each time the simulation runs, which can be useful for creating non-repetitive, diverse test cases.

Example:

int seed = 42;

int rand\_val;

rand\_val = $random(seed); // Using a fixed seed for deterministic randomization

1. What is the difference between $random and $urandom?

* $random: Generates a random number using a signed 32-bit value. It can generate both positive and negative numbers.
* $urandom: Generates an unsigned random number and always returns a positive value (0 to 32-bit).

1. Difference between pre\_ramdomize and post\_randomize?

* pre\_randomize: This method is called before the randomization process. It allows you to prepare or modify variables before randomizing them.
* post\_randomize: This method is called after the randomization process. It allows you to perform checks or modifications after the values have been randomized.

1. Is it possible to override existing constraints?

Yes, constraints can be overridden in SystemVerilog. For example, by using a new constraint or modifying an existing one during randomization. This can be useful when you need to relax or tighten constraints dynamically. Example:

a inside {1, 2, 3}; // First constraint

a inside {4, 5, 6}; // Override the constraint

1. Suppose a is a 7-bit variable. Write constraint for a condition where for alternate bit positions value is 1.

constraint alternate\_bits {

a[0] == 1;

a[2] == 1;

a[4] == 1;

a[6] == 1;

}

1. a is a 32-bit input to DUT with clk and rst. Outputs are 32 bits b and c. When a[0]==1, a=b when a[0]==0, a=c. Write the constraint for this DUT.

constraint DUT\_behavior {

if (a[0] == 1)

b == a;

else

c == a;

}