SYSTEM VERILOG

FREQUENTLY
ASKED
QUESTIONS

JAIRAJ MIRASHI
DESIGN VERIFICATION
ENGINEER

1. **What is virtual function?**

In SystemVerilog, a virtual function is a type of function that allows a base class to define a function signature which can be overwritten in a derived class. This means that a virtual function can be customized by a subclass to perform a different function than the base class.

Virtual functions are an important aspect of object-oriented programming (OOP) and are used heavily in verification methodologies such as the Universal Verification Methodology (UVM). In UVM, virtual functions are used to customize the behavior of verification components and facilitate the reuse of code across different testbenches.

2. **Difference between static and dynamic casting.**

Static and dynamic casting are type conversion operations used to convert between different data types.

Static casting is a compile-time operation in which the compiler converts a given data type into another data type. It is called static casting because the conversion is determined and enforced by the compiler at the time of compilation. Static casting is a simple and efficient process, but it can lead to errors or loss of information if the conversion is not compatible.

Dynamic casting, on the other hand, is a run-time operation in which the type of an object is determined at run-time and then explicitly converted to a different type. Dynamic casting is more complex and less efficient than static casting, but it is safer as it can detect and handle errors during runtime.

3. **Difference between virtual and pure virtual function.**

In Verilog and SystemVerilog, virtual and pure virtual functions are used in polymorphism and inheritance.

A virtual function is a function declared in a base class that can be overwritten in a derived class, enabling runtime polymorphism. When the function is called on an object of the derived class, the derived class's implementation of the function is executed. If there is no implementation in the derived class, the base class's implementation is executed. The syntax for a virtual function is declared by using the keyword virtual in the base class.

A pure virtual function, on the other hand, is a virtual function that has no implementation in the base class. This means that derived classes must provide an implementation for the function. If a derived class doesn't overwrite the pure virtual function, it will remain abstract and cannot be instantiated. Pure virtual functions are used to create abstract classes that act as a blueprint for derived classes.

```
virtual class BasePacket;
pure virtual function int transfer(bit[31:0] data); // No implementation
endclass

class ExtPacket extends BasePacket;
virtual function int transfer(bit[31:0] data);
...
endfunction
endclass
```

4. Why do we need randomization in SystemVerilog?

In SystemVerilog, randomization is the process of generating random stimuli or inputs to the design, which can help verify its functionality and reliability. Here are some reasons why we need randomization in SystemVerilog.

5. Difference between while and do while.

In programming languages, the while loop and do-while loop are used for repetitive execution of a code block based on certain conditions. Here are the differences between while and do-while loops:

1) Execution: In the while loop, the condition is checked first, and if it is true, then the code block is executed. However, in the do-while loop, the code block is executed first, and then the condition is checked. This means that the code block will be executed at least once in the do-while loop, even if the condition is initially false.
2) Loop condition: In the while loop, the loop condition is checked at the beginning of each iteration. If the condition is false, the loop is terminated, and the control goes to the next statement after the loop. However, in the do-while loop, the loop condition is checked at the end of each iteration. This means that the code block is executed at least once, even if the condition is false.
3) Initialization: In the while loop, the initialization of the loop variable or counter happens outside of the loop. This makes it possible to create an infinite loop if the initialization is incorrect. However, in the do-while loop the initialization of the loop variable or counter can be done within the loop.
4) Use Cases: The while loop is used in cases where the condition is unknown, and the loop should not execute if the condition is false. In contrast, the do-while loop is used when we want the loop to execute at least once, even if the condition is false.

6. Explain bidirectional constraints

Bidirectional constraints are used to specify a relationship between two or more variables or signals, such that the value of one variable is dependent on the value of the other variable(s). In SystemVerilog, constraints are solved bidirectionally.

For example, if we have two signals, data_valid and data_ready we can specify the constraint that when data_valid is asserted data_ready must be asserted, using the following conditional constraint:

```
data_valid -> data_ready;
```

7. Difference between integer and int.

integer is a 4-state data type where as int is a 2-state data type. Both can hold 32-bit signed numbers.

8. Write a constraint to have a variable divisible by 5.

```
class DivisibleBy5Class;
  rand bit[3:0] my_variable;

  constraint c_div5 {
    my_variable % 5 == 0;
  }

  function void post_randomize();
    // Additional post-randomization checks or actions can be added here
    // if needed.
    $display("Randomized value: %0d", my_variable);
  endfunction

endclass

module test;
  initial begin
    DivisibleBy5Class obj;
    obj = new;

    repeat (5) begin
      obj.randomize();
      obj.post_randomize();
    end

  end
endmodule
```

Here, the modulo operator % is used to retrieve the remainder of a division operation, and the constraint ensures that the remainder of the division of my_variable by 5 is equal to zero, which means my_variable is divisible by 5.

9. What are parameterized classes?

Parameterized classes in object-oriented programming (OOP) are classes that are defined with one or more parameters, allowing them to be customized or specialized during their

instantiation or creation.

```
class stack #(type T = int); T item;

function T add_a (T a); return item + a;
endfunction
endclass
```

10. How can you establish communication between monitor and scoreboard in SystemVerilog?

In SystemVerilog, a monitor and scoreboard are typically used in a verification environment to check that a design is behaving correctly. The monitor observes the signals of the design and generates transactions, while the scoreboard compares these transactions against expected values and generates results, and they can be connected using a mailbox.

11. Is it possible to override existing constraints?

Yes, it's possible to override existing constraints in SystemVerilog using inline constraints or inheritance.

```
class ABC;
rand bit [3:0] data;

constraint c_data { data inside {[5:10]}; } endclass

module tb;
initial begin
ABC abc = new;
end
endmodule
// Use inline constraint to override with new value
// Note that this should not contradict the hard constraints in ABC
abc.randomize() with { data == 8; };
```

Another way is to redefine the constraint block in an inherited class.

```
class DEF extends ABC;

constraint c_data { data inside {[0:5]}; } endclass

module tb;
initial begin
DEF def = new; def.randomize();
end
endmodule
```

12. What will be your approach if functional coverage is 100% but code coverage is too low ?

If the functional coverage is 100% but the code coverage is too low, it typically means that there are parts of the code that are never executed and therefore not covered by the test cases. Here are some possible approaches to improve code coverage:

1) Review the code and identify the uncovered areas: You can go through the code manually or use a code coverage tool to identify the parts of the code that are not covered by the tests. This will help you to focus on the areas that need to be tested more thoroughly.
2) Add new test cases: Once you have identified the uncovered areas, you can add new test cases to cover them. You can use different techniques such as random testing, directed testing or assertion-based testing to create new test cases.
3) Modify existing test cases: You can also modify the existing test cases to cover the uncovered areas. For example, you can change the parameters or stimuli of the test cases to target specific areas of the code.
4) Use code coverage metrics: You can use code coverage metrics as a guide to track your progress and ensure that you are improving overall coverage. You can set targets for specific types of coverage, such as branch or statement coverage, and monitor your progress as you add new test cases or modify existing ones.
5) Use techniques like coverage-driven verification and constrained-random testing: These techniques focus on generating test cases to cover specific portions of the design, which can help you achieve better code coverage.

13. What are the types of assertions?

Assertions are statements that specify a particular relationship between two or more signals or variables. There are two main types of assertions, which are as follows:

1) Immediate assertion: Checks whether a particular condition is true at a particular point in time.
2) Concurrent assertion: Checks whether a particular condition is always true over a period of time.

14. How to find indices associated with associative array items?

Array manipulation functions can be used to query indices and values in SystemVerilog arrays.

```
module tb;
int fruit_hash [string];
string  idx_q [$];

initial begin
fruit_hash["apple"] = 5;
fruit_hash["pear"] = 3;
fruit_hash["mango"] = 9;

idx_q = fruit_hash.find_index with (1);
$display("idx_q= %p", idx_q);
end
endmodule

// Output
// idx_q= '{"apple", "mango", "pear"}
```

15. Give an example of a function call inside a constraint.

The function must return a value that can be used in the constraint expression.

Here's an example:

```
function int rand_range(int a, b);
return (a + b) % 2;
endfunction

class ABC;
rand bit my_val;

constraint my_val_c {
my_val == rand_range(a, b);
}
endmodule
```

16. What are pass-by-value and pass-by-reference methods?

Pass-by-value and pass-by-reference are two ways of passing arguments to a function or a method in programming languages.

In pass-by-value method, a copy of the value of the argument is passed to the function or method. Any change made to the value inside the function or method does not affect the original value of the argument.

```
function void abc(int a, b);
```

In pass-by-reference method, a reference to the memory location of the argument is passed to the function or method. Any changes made to the value inside the function or method will affect the original value of the argument.

```
function void abc(ref int a, b);
```

17. Difference between initial and final block.

The initial block is executed at the start of the simulation, i.e. at time 0 units. This is useful for initializing variables and setting up initial configurations. This is the very basic procedural construct supported since the first version of Verilog.

However, the final block was introduced in SystemVerilog and is executed just before the simulation ends. This does not consume any time and hence is very ideal to do last minute housekeeping tasks and print reports.

18. What are the default values of variables in the SystemVerilog ?

The default value of all 2-state variables are zero, and 4-state variables are X. Inputs that are not connected are Z.

19. What is polymorphism and its advantages?

Polymorphism is a fundamental concept in object-oriented programming that allows objects of different classes to be treated as if they are objects of the same class.

The advantages of polymorphism in object-oriented programming are:

1) Code reusability: Polymorphism enables code reuse, as objects of different classes can be treated as if they are objects of the same class. This means that common behaviors and methods can be defined in a superclass and inherited by multiple subclasses.
2) Flexibility and extensibility: Polymorphism allows objects to behave differently based on the context in which they are called, improving the flexibility of the code. It also enables

extensibility by allowing new subclasses to be added to a program easily without modifying the existing code.

3) Code readability and maintainability: Polymorphism can improve the readability and maintainability of the code by reducing the amount of redundant code and making it easier to understand the relationship between different classes.

20. What is the purpose of this pointer in SystemVerilog ?

The this keyword is used to explicitly reference the current class object. This is mostly used within class definitions to be able to specifically reference variables and methods belonging to the same class.

21. What is a SystemVerilog interface?

SystemVerilog interfaces are a way to create structured hierarchical connections between modules and blocks in a design. They provide a way to bundle signals and functionality into reusable components, which can be easily instantiated and connected in a design.

1) Modular design: Interfaces provide a modular approach to design, making it easier to create and reuse building blocks in a system.
2) Encapsulation: They help in encapsulating the functionality and signals inside a module or block, making it easier to understand and maintain.
3) Configurability: Interfaces can be parameterized, allowing for easy configurability and scalability.

22. Difference between reg and logic.

Both reg and logic are used to store 4-state logic values that can be referenced later. The main difference is that logic signals can be used in both procedural and continuous assignments whereas reg can only be used in procedural code.

23. Difference between :/ and := operators in randomization.

Both operators are used in distribution constraints to assign weightage to different values in the distribution.

The :/ operator assigns the specified weight to the item, or if the item is a range, then the weight of each value is divided by N. The := operator assigns the specified weight to the item or if the item is a range, then to every value value in the range.

24. How to disable randomization?

Randomization of variables can be disabled with rand_mode.

```
class ABC;
rand bit [3:0] data;
endclass

module tb;
initial begin
ABC m_abc = new;
m_abc.data.rand_mode(0);     // Disable randomization

m_abc.data.rand_mode(1);     // Enable randomization
end
endmodule
```

25. Different types of code coverage.

Code coverage is a metric used to measure how well tests have exercised the design under test. It typically reports on the percentage of execution achieved across various dimensions of the design, such as statements, branches, and expressions, toggle, assertions and FSM.

26. Write a constraint to detect odd numbers of ones in an 8-bit sequence.

Here's an example constraint that detects odd numbers of ones in an 8-bit sequence:

```
class ABC;
rand bit [7:0] data;

constraint c_data { $countones(data) % 2 != 0; }
endclass

module tb;
initial begin
ABC m_abc = new;

for (byte i = 0; i < 20; i++)
begin
m_abc.randomize();
$display("data = 0b%0b", m_abc.data);
end
end
endmodule
```

27. What are local and protected access qualifiers?

A class member declared as local is available only to methods inside the class and are not visible within subclasses.

A class member declared as protected is available to methods inside the class and also to methods within subclasses.

28. How does OOP concepts help in verification?

1) Modular Design: Each module/class represents different aspects of the design. As a result, it becomes easier to develop a test bench by reusing and integrating these modular components. This saves time and effort in developing test cases and makes the system more scalable.
2) Encapsulation: Operations performed on any data object can be defined within the same class. For example, a function to pack a data array into a stream of 64 bit data.
3) Inheritance: Allows to create a base class and extend it with further derived classes.
4) Common properties and methods are usually defined in base classes so that all subclasses have access and hence avoid code duplication
5) Polymorphism: Allows different implementations of the same method or function.
6) Polymorphism helps to implement alternate test case scenarios without affecting the rest of the code significantly.

29. What is a virtual interface?

An interface is a collection of signals that allow connections to the DUT from the testbench. The handle to this interface is made available in classes by making it virtual. Hence a virtual interface is nothing but a handle that can hold a reference to the actual interface.

Remember that an interface is declared at the testbench top level so that it can be passed to the DUT. So, rest of the components get access to this interface via a virtual interface.

```
class wishbone_monitor;
virtual wishbone_if m_vif; // A virtual interface handle

virtual task monitor;
forever
begin
@(m_vif.clk);
// Rest of the code
end
endtask
endclass
```

30. Why logic was introduced in SV?

logic is a new SystemVerilog data type, when compared to Verilog which can be used in place of reg and wire in both procedural blocks and continuous assignments. It removes the hassle of having to redefine an existing signal as reg or wire depending on where it is used.

31. Write a small function to push 10 unique values from 0 to 50 into a queue.

```
function random();
bit [7:0] array[$];
for (int i = 0; i <10; i++)
begin
int num;
std::randomize(num) with { num inside {[0:50]};
};
!(num inside {array});
array.push_back(num);
end
endfunction
```

32. Write constraints to randomize with the following requirements.

Assume memory region from 0x0 to 0x100. There is a small region in between from 0x20 to 0xE0 that is reserved. Write system verilog constraints to choose a block of memory of size 16 bytes that is outside the reserved region and inside the entire memory range. The starting address of the block should be 4-byte aligned.

```
rand bit [31:0] addr;
int size = 'h10;

constraint c_addr {
  addr inside { [0:'h100] }; // Ensure it's within memory region
  !(addr inside { ['h20:'hE0] }); // Ensure it's not in the reserved region
  addr % 4 == 0; // Ensure it's 4-byte aligned
  (addr + size) inside { [0:'h20], ['hE0:'h100] }; // Ensure it's either in
the lower or upper region
  !(addr + size inside {'h20, 'h100}); // Ensure the last addr does not hit
the limit
}
```

33. Write constraints to randomize with the following requirements.

Assume a memory region exists from 0x2000 to 0x4000 that is byte addressable. Write SV constraints to randomly pick an address within this memory region that is aligned to 4-byte boundary.

```
bit [31:0] addr;

constraint c_addr {
  addr inside {[32'h2000:32'h4000]};
  addr % 4 == 0;
  // addr[1:0] == 0; Also okay
}
```

Provide a solution for the following requirement.

Assume a class called "ABC" has been used throughout in a project. In a derivative project, you had to extend "ABC" to form "DEF" and add some more variables and functions within it. What will happen if you try to use an object of "ABC" that was created in the legacy testbench to access these new variables or functions?

It will result in a compilation error because the new variables do not exist in the base class. Instead, you need to declare a local variable of type "DEF" and perform a dynamic cast if required to access the new variables and functions.

Randomly generate 8, 16, 32, 64 with equal probability using SystemVerilog constructs.

```
module tb;
  initial begin
    bit [31:0] result;

    result = 1 << $urandom_range(3, 6);
  end
endmodule
```

34. What is the difference between a mailbox and queue?

A queue is a variable size ordered collection of elements of the same type. Read more on SystemVerilog Queues

A mailbox is a communication mechanism used by testbench components to send a data message from one to another. A mailbox has to be parameterized to hold a particular element and can be either bounded or unbounded. It can also suspend the thread by tasks like get() and put(). So a component can wait until an item is available in the mailbox.

35. What is the difference between rand and randc?

SystemVerilog allows us to randomize variables inside a class using rand and randc constructs.

rand randomizes the variable and can have repetitive values before the entire set of allowable values are used. For example, a 2-bit variable when used with rand can give values 1, 3, 3,2,1,3,0.

randc randomizes the variable and repeats a value only after the entire set of allowable values are used. For example, a 2-bit variable when used with randc can give values [1, 3, 2, 0], [0, 3, 1, 2], 1 ... The values in the square brackets show a set.

36. How can we reference variables and methods defined in the parent class from a child class?

The super keyword is used to access variables and methods of the parent class and is a very basic construct of OOP.

37. Where is extern keyword used?

An extern keyword is used to define methods and constraints outside the class definition. For example, we could have the declaration of functions and constraints within the class body, but do the complete definition later on outside the class body.

38. Give one way to avoid race conditions between DUT and testbench in a verification environment.

Clock edges are the most probable points for a race condition to arise. The DUT may sample one value but the testbench may sample something else.

Although race conditions may arise from improper coding practices, use of SystemVerilog Clocking Blocks allows the testbench to sample DUT appropriately and drive inputs to the DUT with a small skew. Also, this allows the skews to be changed later on with minimal code change.

Race condition can also be avoided by the use of program blocks and the use of non- blocking assigments.

39. What is the difference between logic and bit in SystemVerilog?

In SystemVerilog, a bit is a single binary digit that can have a value of 0 or 1, while logic is a data type used for representing a single wire or net that can have multiple states such as 0, 1, Z (high-impedance), X (unknown), or L (weakly driven low) and H (weakly driven high).

40. How to check if any bit of the expression is X or Z?

```
// To check if all bits of variable "xyz" are Z
if (xyz === 'Z)
begin
$display("All bits in xyz are Z");
end

// To check if any bit in "xyz" is Z
if ($countbits(xyz, 'Z))
begin
$display("Some bit in xyz is Z");
end

// To check if signal is X
if ($isunknown(xyz))
begin
$display("xyz is unknown or has value X");
end
```

41. Figure out a solution to the following puzzle.

Assume two base classes A and B, and two derived classes C and D where this relation between classes is unknown to end user. How do you find base class for each derived class?

```
class A;
class B;

// Assume this relation is hidden from end user
class C extends B;
class D extends A;


module tb;
  initial begin
    A   m_a = new();
    B   m_b = new();
    C   m_c = new();
    D   m_d = new();

   // Successful cast implies that the second arg is a child of the first arg
    if ($cast(m_a, m_c))
      $display("C is a child of A");

    if ($cast(m_a, m_d))
      $display("D is a child of A");

    if ($cast(m_b, m_c))
      $display("C is a child of B");

    if ($cast(m_b, m_d))
      $display("D is a child of B");
  end
endmodule
```

42. Write SV code to wait for a random delay in range 100 to 500 ns.

Delays are indicated by the # construct and a random delay can be written as follows.

```
`timescale 1ns / 1ps

module tb;
  initial begin
    int delay;

    std::randomize(delay) with {
      delay inside { [100:500] };
    };

    #(delay)
    $display("Statement printed after %0d delay", delay);

    // This is also good enough, although there's no variable to print the
actual randomized delay
    #( $urandom_range(100, 500) )
    $display("Some delay between 100 to 500");
  end
endmodule
```

43. What is the difference between a parameter and typedef?

parameter is used to define compile-time constants used within modules, which are values that can be evaluated and assigned before the simulation starts. It can be used to specify parameters such as width, depth, or delay of modules.

```
module my_module #(parameter WIDTH=8) ( input [WIDTH-1:0] data_in,
output [WIDTH-1:0] data_out
);
// module logic here
endmodule
```

typedef is used to define custom data types that can be reused throughout the design. It can be used to define complex data types such as structures, arrays, and enumerated types. It is used to make the code more readable and easier to understand by encapsulating complex data types within a single type name.

```
typedef struct { logic [7:0] addr;
logic [31:0] data;
} request_t;

request_t my_req;

my_req.addr = 8'h22; my_req.data = 32'h12345678;
```

44. What is constraint solve-before?

In SystemVerilog, solve - before is a constraint solver directive that allows constraints to be solved in a specific order. This directive specifies that a particular constraint should be solved before another constraint. It is useful in cases where certain constraints must be solved before others to avoid conflicts or to ensure that specific constraints are satisfied.

45. What is an alias?

alias is a keyword used to declare an alternate name for a variable or net. It allows access to the same object through multiple names. The new name created using the alias keyword refers to the same variable or memory location as the original variable.

```
wire [7:0] _byte; wire  _bit;

alias bits_9 = { _byte, _bit };
```

46. Write code to extract 5 elements at a time from a queue.

```
module tb;
  bit [7:0] q[$];
  bit [7:0] tmp[$];

  initial begin
    repeat (9) q.push_back($random);

    for (byte i = 0; i < q.size(); i += 5)
      tmp = q[i +: 5];
  end
endmodule
```

47. What is the difference between the clocking block and modport?

A clocking block is used to model clock and reset signals and their associated timing control signals. It provides a way of defining a set of timing signals as well as their phases and signal transitions.

A modport is used to group a set of port declarations into a named entity. It allows designers to specify multiple port configurations and to limit access to specific module interfaces.

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

$random returns signed integer values, whereas $urandom returns an unsigned integer value.

```
int data1;
bit [31:0]  data2;

data1 = $random;     // signed integer
data2 = $urandom;    // unsigned integer
```

49. Why can't program blocks have an always block in them ?

An always block is a concurrent process that runs forever and gets triggered based on changes to signals in the sensitivity list. A program block is intended to be a testcase that applies stimulus to the DUT and finish at some point in time. Having an always block will stall the program from coming to an end and hence it doesn't make sense to include it in a program block.

```
bit [7:0]   data;
std::randomize(data) with { data > 7; };     // in-line constraint
```

50. What are the advantages of cross-coverage?

Cross-coverage is a type of coverage measurement in SystemVerilog that combines coverage data for two or more variables or conditions. Here are some of the advantages of using cross-coverage:

1) Better coverage granularity: Cross-coverage allows the user to define more specific coverage goals that combine multiple variables, instead of relying on individual coverage points. This provides better visibility into the overall behavior of the design, and helps to identify corner cases or unexpected interactions.
2) Reduced verification effort: By combining coverage data for multiple variables or conditions, cross-coverage reduces the number of individual coverage points that need to be tested. This can save verification time and effort, especially for complex designs with many variables or conditions.
3) Improved result analysis: Cross-coverage generates more detailed coverage reports that show the correlation between different variables or conditions. This helps the user to identify patterns or trends in the data, and to perform more targeted verification activities.

51. What are constraints?

SystemVerilog constraints are used to control the values that are randomized for variables during simulation. Constraints provide a way to specify the valid range of values for a variable, as well as any relationships or conditions between variables.

```
rand bit [7:0] data;
constraint myConstraint {
data inside {[0:10]};
}
```

52. How can we display hex values of a variable in uppercase?

$display format specifier can have %h or %H , and quite intuitively we assume that the latter is used to display it in uppercase. However, that is not the case and we need a workaround.

```
bit [31:0] y = 32'hCAFE_4BED;
string str;

str.hextoa(y);
$display(str.toupper);
```

53. Constrain a dynamic array such that it does not pick values from another array.

```
module tb;
  bit [3:0] da [];
  bit [3:0] myq [$];

  initial begin
    repeat (10) myq.push_back($random);

    std::randomize(da) with {
      da.size == 10;
      foreach (da[i]) {
        !(da[i] inside {myq});
      }
    };

    $display("Randomized da: %p", da);
    $display("myq: %p", myq);
  end
endmodule
```

54. What is the output for the following code?

```
initial begin
  byte loop = 5;

  repeat (loop) begin
    $display("hello");
    loop -= 1;
  end
end
```

The repeat loop iterates a fixed number of times and the iteration count cannot be changed once in the loop. The output will be:

```
hello
hello
hello
hello
hello
```

55. What is the difference between overriding and overloading?

Overriding:

Overriding is a concept in object-oriented programming that involves providing a new implementation for a method inherited from a superclass in a subclass. In this process, the subclass redefines a method with the same signature (name, return type, and parameters) as the one in the superclass. When an object of the subclass calls the overridden method, the new implementation in the subclass is executed instead of the original implementation in the superclass. The main objective of method overriding is to implement a different behavior for the same method in the subclass, allowing for customization and specialization.

Overloading:

In contrast to some programming languages like C++, SystemVerilog does not support function overloading, where multiple functions with the same name but different parameter lists can exist. SystemVerilog allows only a single function with a specific name in a scope. Therefore, function overloading, as supported in some other languages, is not possible in SystemVerilog.

56. What are the different types of verification approaches?

1) Functional Simulation: Running the digital design on a computer or simulator to validate its functionality. Involves test vectors and various inputs to ensure expected behavior.
2) Formal Verification: Using mathematical proofs to verify the correctness of the design. Applied to critical designs, especially in safety-critical systems.
3) Emulation: Testing the digital design on specialized hardware emulating the system's behavior. Suitable for large, complex designs that cannot be simulated on a computer.
4) Prototyping: Building a physical prototype to test the system's functionality in a real-world environment. Applied when real-world inputs and conditions are essential for testing.

Effective verification ensures a design meets specifications and standards, identifying and eliminating errors before deployment. The choice of the method is project-dependent and should adequately validate the design to ensure correctness.

57. What is the difference between always_comb() and always@(*)?

always_comb is automatically triggered once at time zero, after all initial and always procedures have been started, whereas always @ (*) will be triggered first when any signal in the sensitivity list changes.

always_comb is sensitive to changes of contents in a function where the latter is sensitive only to the arguments of the function when invoked inside it. always_comb cannot have statements that have delays or timing constructs in it.

Variables used on the LHS inside an always_comb block cannot be assigned to in any other parallel process.

58. What is `timescale?

The `timescale directive is used to set the time units and precision for a design. It specifies the time scale used in the simulation and the unit of time for delays and times associated with signal assignments and other operations.

```
`timescale timeunit/precision
```

59. What are the basic testbench components?

A basic testbench consists of the following components:

- Stimulus generation: Stimulus generation involves creating test vectors or other input signals that will be applied to the inputs of the design under test (DUT).
- Interface: The interface provides the communication between the testbench and the DUT. It includes input and output ports, which are connected to the corresponding signals in the DUT.
- Driver: The driver is responsible for converting stimulus into pin toggles appropriate to the bus protocol used by the DUT.
- Monitor: The monitor observes the output signals of the DUT and produces a stream of data that the testbench can analyze.
- Scoreboard: The scoreboard compares the output of the DUT with the expected output, and generates an error message or other notification if there is a mismatch.
- Coverage analysis: Coverage analysis measures how much of the design has been exercised during simulation, and helps identify parts of the design that may not have been tested thoroughly.

60. What is circular dependency?

Circular dependency is a situation in which two or more components or entities of a system depend on each other in a circular way, creating a cycle. It can occur between modules, functions, or libraries that depend on each other in a way that creates a loop. This can lead to issues when trying to compile or link the code, as the dependencies cannot be resolved. This can be solved with forward declaration by typedef.

```
// Forward declaration of class B
typedef class B;

// Declaration of class A
class A;
  B b;
  // ... other members of class A
endclass

// Definition of class B
class B;
  A a;
  // ... other members of class B
endclass
```

61. What are the advantages of a SystemVerilog program block?

Statements inside a program block are executed in the reactive region which is executed the last in a Verilog simulator event scheduler and hence can react on final state of design signals. It acts as an entry point for test stimulus to be executed by the testbench. It can contain functions, tasks and other supported constructs that enable user to create meaningful test stimulus.

62. What is scope randomization?

Scope randomization allows to define different randomization constraints for different parts or scopes of the design hierarchy.

```
module tb;
byte data;

initial begin
randomize(data) with { data > 0 };
$display("data = 0x%0h", data);
end
endmodule
```

Note that std:: is required if you need to call from within a class method to distinguish it from the class's built-in randomize method.

63. What is the input skew and output skew in the clocking block?

A clocking block is a feature in SystemVerilog used to manage clock signals in a design. Input skew refers to when a signal defined as input to the clocking block should be sampled relative to the given edge of the clock. Output skew refers to when a signal declared as an output to the clocking block should be driven relative to the given edge of the clock.

```
clocking cb @(posedge clk);
default input #1step output #1ns;
input   hready;
output haddr;
endclocking
```

64. What is casting?

Casting is a fundamental concept in programming and refers to the process of converting one data type into another data type. In SystemVerilog, casting is mainly used for type conversion between numeric and non-numeric data types.

```
real pi = 3.14;
int a = int'(pi) * 10;  // Cast real into an integer number

$cast(aa, a);    // Cast object 'a' into 'aa'
```

65. How to generate array without randomization?

This solution does not generate fully random numbers, however it does give unique values.

```
module tb;
byte    data_q [10];

initial begin
// Store numbers 0 through 10
foreach (data_q [i])
data_q[i] = i;

// Shuffle the array
data_q.shuffle();

end
endmodule
```

66. Write randomization constraints for the following requirements on an array.

An array of size 9 should contain any value from 1 to 9. Two values should be the same between indices 0 and 7.

```
module tb;
  bit [3:0] data_q [9];

  initial begin
    bit [2:0] idx_q [2];
    randomize(data_q, idx_q) with {
      // Let idx_q represent two random indices
      // which should contain the same value
      unique { idx_q };
      solve idx_q before data_q;

      // When indices match, assign the same value to data_q[i]
      // else follow the general rule to keep between 1:9
      foreach (data_q[i]) {
        if (i == idx_q[1]) {
          data_q[i] == data_q[idx_q[0]];
        } else {
          data_q[i] inside {[1:9]};
        }
      }
    };

    $display("idx_q = %p", idx_q);
    $display("data_q = %p", data_q);
  end
endmodule
```

67. What is SVA?

SVA or SystemVerilog Assertions provides a syntax for expressing assertions that describe the expected behavior of a design, allowing for direct verification of its correctness.

Assertions expressed using SVA can be used to verify various types of design properties, such as proper data flow, correct timing constraints, and correct synchronization between different parts of the design. SVA can be used as a standalone language or in conjunction with other formal verification techniques such as model checking and theorem proving. It is an important tool for ensuring the correctness and reliability of digital designs in VLSI and other fields.

68. When you will say that verification is completed?

Verification is typically considered complete when all the specified verification goals and requirements have been met and demonstrated through testing and analysis of the design. This means that all of the verification tests have been run and that the design has passed all of the necessary functional and performance requirements. Verification is a continuous process that starts early in the design cycle and continues until the final stages of the design and development process. Throughout this process, different verification techniques and methodologies are used to ensure that the design is free from errors.

Verification completeness is typically determined using a set of predefined metrics and criteria that are used to evaluate the overall quality and reliability of the design. Such metrics may include functional coverage, code coverage, and timing analysis. Ultimately, the decision to declare verification complete is based on the verification team's confidence in the design's functionality and reliability to be used in the intended application and environment.

69. What are system tasks?

System tasks are pre-defined functions or built-in functions in SystemVerilog that are used to execute certain tasks related to the simulation and verification of a design. Some common system tasks in SystemVerilog include:

$display: Used to display formatted output to the console or log file.

$time: Used to retrieve simulation time and wall-clock time, respectively.

$finish: Used to end the simulation after a specific amount of time or when a specific condition is met.

$random: Used to generate random values for variables or signals.

70. In SystemVerilog which array type is preferred for memory declaration and why?

The preferred array type for memory declaration is an associative array because it is more efficient in storing data at random address locations. It does not require all addresses in memory to be pre-allocated before usage unlike a dynamic array.

71. What is the advantage of seed in randomization?

In SystemVerilog, seed is used as a starting point or initial value for the random number generator. The advantage of using the seed in randomization is that it allows for a more deterministic and reproducible behavior of the randomized simulation.

By setting a seed, a specific set of randomized values can be generated consistently, making it easier to replicate specific test scenarios and debug issues that arise during simulation. It also allows for better verification of the design as specific tests can be rerun with the same seed to ensure that issues have been resolved and that the behavior of the design is as expected.

72. Is it possible to write assertions in a class?

Yes, assertions using assert and assume are used to check the correctness of the design, and they can be written in any of the SystemVerilog constructs including modules, interfaces, programs or classes.

In SystemVerilog, assertions can be written using the assert and assume keywords. These keywords can be used directly inside a SystemVerilog class, with the assertion check being triggered when the appropriate method of the class is called.

73. What is a clocking block?

A clocking block is a SystemVerilog construct that provides a way to model clock-related events that occur in a design. It is specifically used to define the timing and synchronization of signals that are driven by a clock. The clocking block can be used to drive and sample signals using the clock signal, with the signals being synchronized at specific edges of the clock.

74. What is an abstract class?

An abstract class is a class in object-oriented programming that cannot be instantiated, meaning it cannot be used to create objects. Instead, it is used as a superclass to other classes, providing a common set of properties and methods that subclasses can inherit and implement as necessary.

75. How to disable a coverpoint ?

Covergroups and coverpoint weight can be disabled by setting its weight to zero.

```
covergroup cg_ahb @ (posedge hclk);
cp_haddr  :  coverpoint haddr;
cp_htrans :  coverpoint htrans;
...
endgroup

cg_ahb m_cg_ahb = new();
m_cg_ahb.cp_htrans.option.weight = 0; // disable coverpoint by setting weight
to 0
```

76. What is super keyword ?

The super keyword in SystemVerilog or even any OOP language refers to the superclass of a class. It is used to access methods and variables of the superclass from within a subclass.