SystemVerilog Interview Questionsa

1. What is callback?

Callback is mechanism of changing to behavior of a verification component such as driver or generator or monitor without actually changing to code of the component. It's used for functional coverage, inject errors and output transaction in a scoreboard.

```
class abc_transactor;
    virtual task pre_send();
    endtask
    virtual task post_send();
    endtask
    task xyz();
        this.pre_send();
        this.post_send();
    endtask : xyz
    endclass : abc_transactor

class my_abc_transactor extend abc_transactor;
    virtual task pre_send();
        ... // This function is implemented here
    endtask
    virtual task post_send();
        ... // This function is implemented here
    endtask
    endclass : my_abc_transactor
```

The base class abc_transactor has 3 tasks, 2 of which are declared virtual and are not implemented. But they are being called from another task xyz() which is fully implemented. The unimplemented virtual task are called callback taskes. The child class, which extends from the base class, implements the previous unimplemented tasks. It inherits the xyz() task from the base class and hence doesn't need to change it.

By this we can inject executable code to a function here is xyz() without modifying it.

1) The biggest advantage is that you can modify the behavior of task xyz() without modifying it in the base or child class.

2) Consider a case where you are writing a base class which is going to be used by multiple test environment, and for each test environment a known part of the code, or a known function/task is going to change. The natural choice is to implement those change-in-every-case functions/tasks as callback method and let the user extend your base class with specifying only that part of the code which need to be changed in his case.

3).at VMM,

```
(3.1)callback files
typedef class wif_wr_data;
class wif_wr_data_cbs extends vmm_xactor_callbacks;
  virtual task post_tx(data_to_sb packet);
  endtask // pre_tx
endclass // wif_wr_data_cbs
class wif_wr_data_callbacks extends wif_wr_data_cbs;
  mvc_scoreboard scbd;

  function new (mvc_scoreboard scbd);
    this.scbd = scbd;
  endfunction // new

  virtual task post_tx(data_to_sb packet);
   scbd.save_expected_packet(packet);
  endtask // pre_tx
endclass

(3.2)wr_monitor,
//Need to declare the callback first
wif_wr_data_callbacks wr_data_cbs;

`vmm_callback(wif_wr_data_callbacks, post_tx(pkt));

(3.3)mvc_scoreboard,
//Need to declare and define the function
function mvc_scoreboard::save_expected_packet(data_to_sb wr_pkt);

(3.4)in UVM,
class uvm_callbacks #(
type T = uvm_object,
type CB = uvm_callback
) extends uvm_typed_callbacks#(T)
```

2. What is factory pattern?

Factory pattern as the name suggest, is aimed at solving the issue of creation of object. (Factory pattern is not the only pattern to deal with creation of objects -> creational patterns)

```
class TOY; // Common data memeber string type;
    string type;
    virtual function string get_type();
endclass
class TOY_Tank extends TOY;
    function new();
        this.string = "Toy Tank";
```

```
    endfunction
    string function string get_type();
        return this.string;
    endfunction
endclass
class TOY_Bus extends TOY;
    function new();
        this.string = "Toy Bus";
    endfunction
    string function string get_type();
        return this.string;
    endfunction
endclass
```

Now we are done with the bothering about the objects to be created. The next problem that we need to solve is to write the toy factory class itself. For simplicity, let's consider the case where we will want to pass 1 to get an instance of tank class and 2 for getting an instance of bus class from the factory. Now the factory class will look like this.

```
class TOY_factory;
    Toy my_toy
    // Common methods
    function toy get_toy(int type);
        if(type == 1)
            this.my_toy = new TOY_Tank();
        if(type == 2)
            this.my_toy = new TOY_Bus();
        return this.my_toy;
    endfunction
endclass
```

Note that we are using virtual function for bringing polymorphism in action and save us from having an individual instance of the toy type in the factory class.

3. Explain the difference between data types logic and reg and wire

Wire:-

1. Wires are used for connecting different elements
2. They can be treated as a physical wire
3. They can be read or assigned
4. No values get stored in them
5. They need to be driven by either continuous assign statement or from a port of a module

Reg:-

1. Contrary to their name, regs doesn't necessarily corresponds to physical registers
2. They represents data storage elements in Verilog/SystemVerilog

3. They retain their value till next value is assigned to them (not through assign statement)

4. They can be synthesized to FF, latch or combinational circuit (They might not be synthesizable !!!)

Logic:-

1. Reg/Wire data type give X if multiple driver try to drive them with different value. logic data cannot be driven by multiple structural drivers, In this case, the variable needs to be a net-type such as wire/tri

2. SystemVerilog improves the classic reg data type so that it can be driven by continuous assignments, gates, and modules, in addition to being a variable

4. What is need of clocking block?

An interface can specify the signals or nets through which a testbench communicates with a device under test (DUT). However, an interface does not explicitly specify any timing disciplines, synchronization requirements, or clocking paradigms. SystemVerilog adds the clocking block that identifies clock signals and captures the timing and synchronization requirements of the blocks being modeled. Any signal in a clocking block is now driven or sampled synchronously, ensuring that your testbench interacts with the signals at the right time.

Specify synchronization characteristics of the design

- Offer a clean way to drive and sample signals
- Provides race-free operation if input skew > 0
- Helps in testbench driving the signals at the right time
- Features
- Clock specification
- Input skew,output skew
- Cycle delay (##)
- Can be declared inside interface (mostly),module or program

5. What are the ways to avoid race condition between testbench and RTL using SystemVerilog?

Likewise, initial blocks within program blocks are scheduled in the Reactive region; in contrast, initial blocks in modules are scheduled in the Active region. In addition, design signals driven from within the program must be assigned using non-blocking assignments and are updated in the re-NBA region. Thus, even signals driven with no delay are propagated into the design as one event. With this behavior, correct cycle semantics can be modeled without races, thereby making program-based testbenches compatible with clocked assertions and formal tools.

Programs that read design values exclusively through clocking blocks with #0 input skews are insensitive to read-write races. It is important to understand that simply sampling input signals (or setting nonzero skews on clocking block inputs) does not eliminate the potential for races. Proper input sampling only addresses a single clocking block. With multiple clocks, the arbitrary order in which overlapping or simultaneous clocks are processed is still a potential source for races. The program construct addresses this issue by scheduling its execution in the Reactive region, after all design events have been processed, including clocks driven by non-blocking assignments.

6. What are the types of coverage available in SV ?

(1. Code Coverage: Here you are measuring how many lines of code have been executed (line coverage), which paths through the code and expressions have been executed (path coverage), which single bit variables have had the values 0 or 1 (toggle coverage), and which states and transitions in a

state machine have been visited (FSM coverage). Code coverage measures how thoroughly your tests exercised the "implementation" of the design specification, and not the verification plan.

(2. Functional Coverage: Functional coverage is tied to the design intent and is sometimes called "specification coverage," while code coverage measures the design implementation.

(3. Assertion coverage: We can measure how often these assertions are triggered during a test by using assertion coverage. A cover property observes sequences of signals, whereas a cover group (described below) samples data values and transactions during the simulation

Using Cover Groups: Variables, expressions and their cross
Using Cover Properties

7. What is the need of virtual interfaces?

Interface can't be instantiated inside non-module entity in SystemVerilog. But they needed to be driven from verification environment like class. Virtual interface is a data type (can be instantiated in a class) which hold reference to an interface (that implies the class can drive the interface using the virtual interface). It provides a mechanism for separating abstract models and test programs from the actual signals made up the design. Another big advantage of virtual interface is that class can dynamically connect to different physical interfaces in run time.

8. Explain Abstract classes and virtual methods.

A set of classes can be created that can be viewed as all being derived from a common base class. A base class sets out the prototype for the subclasses. Because the base class is not intended to be instantiated and can only be derived, it can be made abstract by specifying the class to be virtual:
A virtual method overrides a method in all the base classes, whereas a normal method only overrides a method in that class and its descendants. Virtual methods provide prototypes for subroutines. In general, if an abstract class has any virtual methods, all of the methods must be overridden for the subclass to be instantiated.

Polymorphism: dynamic method lookup

Polymorphism allows to reference the methods of those subclasses directly from the super class variable. The OOP term for multiple methods sharing a common name is "polymorphism".

9. What is the use of the abstract class?

Sometimes it make sense to only describe the properties of a set of objects without knowing the actual behavior beforehand. Abstract classes are those which can be used for creation of handles. However their methods and constructors can be used by the child or extended class. The need for abstract classes is that you can generalize the super class from which child classes can share its methods. The subclass of an abstract class which can create an object is called as "concrete class".

10. What is the difference between mailbox and queue?

	MAILBOX	DATA QUEUE
Characteristics		
Function	Perform data communication between tasks	Perform data communication between tasks
Data Type	Starting address of a message	Actual data
Data Size Limit	No limit	Capacity limit of 65535 and Data size limit of 16 bits
Transfer Order	FIFO/ PRI	FIFO
Task Behaviour	Tasks will not be kept waiting for transmission	Tasks goes into wait state if data queue is full during transmission
Advantages	Small overhead because only the message storing address is transferred	Message size limit at 2 bytes (small overhead)
	No limitation on the message amount because the messages are managed by using a link list	No share memory required
	A large amount of message can be sent	
Disadvantages	Shared memory must be prepared	A large amount of message cannot be sent because the message size is fixed

11. What data structure you used to build scoreboard

    Queue

12. What are the advantages of linked-list over the queue?

Queue has a certain order. It's hard to insert the data within the queue. But Linkedlist can easily insert the data in any location.

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

$random system function returns a 32-bit signed random number each time it is called.
$urandom system function returns a 32-bit unsigned random number each time it is called.

14. What is scope randomization?

Scope randomization in SystemVerilog allows assignment of unconstrained or constrained random value to the variable within current scope

```
module MyModule;
integer var, MIN;

initial begin
    MIN = 50;
    for ( int i = 0;i<10 ;i++) begin
        if( randomize(var) with { var < 100 ; var > MIN ;})
            $display(" Randomization sucsessfull : var = %0d Min = %0d",var,MIN);
        else
            $display("Randomization failed");
    end

    $finish;
end
endmodule
```

15. List the predefined randomization methods.

(1 randomize

(2 pre_randomize

(3 post_randomize

16. What is the dfference between always_combo and always@(*)

(1 always_comb get executed once at time 0, always @* waits till a change occurs on a signal in the inferred sensitivity list.

(2 Statement within always_comb can't have blocking timing, event control, or fork-join statement. No such restriction of always @*.

(3 Variables on the left-hand side of assignments within an always_comb procedure, including variables from the contents of a called function, shall not be written to by any other processes, whereas always @* permits multiple processes to write to the same variable.

(4 always_comb is sensitive to changes within content of a function, whereas always @* is only sensitive to changes to the arguments to the function.

```
module dummy;
    logic a, b, c, x, y;

    // Example void function
    function void my_xor;
        input a;            // b and c are hidden input here
        x = a ^ b ^ c;
    endfunction : my_xor

    function void my_or;
        input a;            // b and c are hidden input here
        y = a | b | c;
    endfunction : my_xor

    always_comb             // equivalent to always(a,b,c)
        my_xor(a);          // Hidden inputs are also added to sensitivity list

    always @*               // equivalent to always(a)
        my_or(a);           // b and c are not added to sensitivity list
endmodule
```

17. What is the use of packages?

In erilog declaration of data/task/function within modules are specific to the module only. The package construct of SystemVerilog allows having global data/task/function declaration which can be used across modules/classes.

It can contain module/class/function/task/constraints/covergroup and many more declarations. The content inside the package can be accessed using either scope resolution operator (::), or using import.

```
package ABC;
    typedef enum {RED, GREEN, YELLOW} Color;
    void function do_nothing()
    endfunction
endpackage : ABC

import ABC::Color;
import ABC::*; // Import everything inside the package
```

18. What is the use of $cast?

Using Casting one can assign values to variables that might not ordinarily be valid because of differing data type. Especially in assigning a base class's handle to an extended class's handle. Static casting and dynamic casting.

e.g.

```
i = int '(10.0-0.1); // static cast convert real to integer
// Dynamic casting
function int $cast( singular dest_var, singular source_exp );
task $cast( singular dest_var, singular source_exp );
```

e.g.

```
$cast( col, 2 + 3 );
```

19. How to call the task which is defined in parent object into derived class ?

```
super.task();
```

20. What is the difference between rand and randc?

rand - Random Variable, same value might come before all the the possible value have been returned. Analogous to throwing a dice.

randc - Random Cyclic Variable, same value doesn't get returned until all possible value have been returned. Analogous to picking of card from a deck of card without replacing.

21. What is $root?

The instance name $root allows you to unambiguously refer to names in the system, starting with the top-level scope.

```
1.package ABC;
2.$root.A; // top level instance A
3.$root.A.B.C; // item C within instance B within top level instance A
```

22. What is $unit?

SystemVerilog introduces the *compilation unit,* which is a group of source files that are compiled together. The scope outside the boundaries of any module, macromodule, interface, program, package, or primitive is known as the *compilation unit scope*, also referred to as $unit. For tools such as VCS that compile all files at once, $root and $unit are equivalent.

23. What are bi-directional constraints?

Constraints by-default in SystemVerilog are bi-directional. That implies that the constraint solver doesn't follow the sequence in which the constraints are specified. All the variables are looked simultaneously. Even the procedural looking constrains like if ... else ... and -> (implication operator) constrains, both if and else part are tried to solve concurrently. For example (a==0) -> (b==1) shall be solved as all the possible solution of (!(a==0) || (b==1)).

24. What is solve...before constraint ?

In the case where the user want to specify the order in which the constraints solver shall solve the constraints, the user can specify the order via solve before construct.

25. Without using randomize method or rand, generate an array of unique values?

```
int UniqVal[10];
foreach(UniqVal[i])
    UniqVal[i] = i;
UniqVal.shuffle();
```

26. Explain about pass by ref and pass by value?

Pass by value is the default method through which arguments are passed into functions and tasks. Each subroutine retains a local copy of the argument. If the arguments are changed within the subroutine declaration, the changes do not affect the caller.

In pass by ref, functions and tasks directly access the specified variables passed as arguments. It's like passing pointer of the variable. For passing Array to the routines, always use ref to improve performance, otherwise, array is coped on stack. If you do not want array value changed, use const ref.

The second benefit of ref arguments is that a task can modify a variable and is instantly seen by the calling function. This is useful when you have several threads executing concurrently and want a simple way to pass information. See Chap. 7 for more details on using fork-join.

27. What's the static variable?

```
class Transaction;
static int count = 0; // Number of objects created
int id; // Unique instance ID
function new();
id = count++; // Set ID, bump count
endfunction
endclass
```

```
Transaction t1, t2;
initial begin
t1 = new(); // 1st instance, id=0, count=1
t2 = new(); // 2nd instance, id=1, count=2
$display("Second id=%d, count=%d", t2.id, t2.count);
end
```

In Sample 5.9, there is only one copy of the static variable count, regardless of how many Transaction objects are created. You can think that count is stored with the class and not the object.

28. What is the difference between bit[7:0] sig_1; and byte sig_2;

byte is signed whereas bit [7:0] is unsigned.

29. What is the difference between program block and module ?

Program block is newly added in SystemVerilog. It serves these purposes

(1. It separates testbench from DUT

(2. It helps in ensuring that testbench doesn't have any race condition with DUT

(3. It provides an entry point for execution of testbench

(4. It provides syntactic context (via program ... endprogram) that specifies scheduling in the Reactive Region.

The major difference between module and program blocks are

(1. Program blocks can't have always block inside them, modules can have.

(2. Program blocks can't contain UDP, modules, or other instance of program block inside them. Modules don't have any such restrictions.

(3. Inside a program block, program variable can only be assigned using blocking assignment and non-program variables can only be assigned using non-blocking assignments. No such restrictions on module

(4. Program blocks get executed in the re-active region of scheduling queue, module blocks get executed in the active region

(5. A program can call a task or function in modules or other programs. But a module cannot call a task or function in a program.

30. How to implement always block logic in program block ?

```
always @(posedge clk or negedge reset) begin
   if(!reset) begin
      data <= '0;
   end else begin
      data <= data_next;
   end
```

```
end

// Using forever : slightly complex but doable
forever begin
   fork
   begin
      @ (negedge reset);
      data <= '0;
   end
   begin
      @ (posedge clk);
      if(!reset)    data <= '0;
      else          data <= data_next;
   end
   join_any
   disable fork
end
```

31. What is the use of modports ?

Modports are part of Interface. Modports are used for specifing the direction of the signals with respect to various modules the interface connects to.

```
interface my_intf;
wire x, y, z;
modport master (input x, y, output z);
modport slave (output x, y, input z);
endinterface
```

32. Write a clock generator without using always block.

```
initial begin
    clk = 0;
    forever #(cycle/2) clk <= ~ clk;
end
```

33. What is forward referencing and how to avoid this problem?

Sometimes a class variable needs to be declared before the class itself has been declared. For example, if two classes each need a handle to the other. This is resolved using typedef to provide a forward declaration/referencing for the second class:

```
typedef class C2;
class C1;
    C2 C;
endclass
```

```
class C2;
    C1 C;
endclass
```

34. What is circular dependency and how to avoid this problem ?

Over specifying the solving order might result in circular dependency, for which there is no solution, and the constraint solver might give error/warning or no constraining.

```
int x,y,z;
constraint XYZ{
    solve x before y;
    solve y before z;
    solve z before x;
}
```

35. What is cross coverage ?

Cross allows keeping track of information which is received simultaneous on more than one cover point. Cross coverage is specified using the cross construct.

```
covergroup cg;
    cover_point_y : coverpoint y ;
    cover_point_z : coverpoint z ;
    cross_yz : cross cover_point_y,cover_point_z ;
endgroup
```

36. How to kill a process in fork/join?

```
    disable fork
```

37. Difference b/w Procedural and Concurrent Assertions?

Immediate Assertions: An immediate assertion checks if an expression is true when the statement is executed by using "if()" or "assert()".

Concurrent Assertions: The other type of assertion is the concurrent assertion that is as a small model that runs continuously, checking the values of signals for the entire simulation, which is needed to be specified a sampling clock in the assertion. By using "property".

```
interface arb_if(input bit clk);
    logic [1:0] grant, request;
    logic rst;
    property request_2state;
        @(posedge clk) disable iff (rst)
        $isunknown(request) == 0; // Make sure no Z or X found
    endproperty
    assert_request_2state: assert property (request_2state);
endinterface
```

38. How to randomize dynamic arrays of objects?

```
class ABC;
    rand bit [7:0] data [];
    constraint cc {
        data.size inside {[1:10]}; // Constraining size
        data[0] > 5; // Constraining individual entry
        foreach(data[i]) // All elements
        if(i > 0)
        data[i] > data[i-1];
    }
endclass
```

39. What is randsequence and what is its use?

A randsequence grammar is composed of one or more productions. Each production contains a name and a list of production items. Production items are further classified into terminals and non-terminals. Non-terminals are defined in terms of terminals and other non-terminals. A terminal is an indivisible item that needs no further definition than its associated code block. Ultimately, every non-terminal is decomposed into its terminals. Production lists separated by a | imply a set of choices, which the generator will make at random.

```
randsequence( main )
    main : first second done ;
    first : add | dec ;
    second : pop | push ;
    done : { $display("done"); } ;
    add : { $display("add"); } ;
    dec : { $display("dec"); } ;
    pop : { $display("pop"); } ;
    push : { $display("push"); } ;
endsequence
```

40. What is bin?

A coverage-point bin associates a name and a count with a set of values or a sequence of value transitions. If the bin designates a set of values, the count is incremented every time the coverage point matches one of the values in the set. If the bin designates a sequence of value transitions, the count is incremented every time the coverage point matches the entire sequence of value transitions.

```
program main;
bit [0:2] y;
bit [0:2] values[$]= '{3,5,6};

covergroup cg;
    cover_point_y : coverpoint y {
        option.auto_bin_max = 4 ; }
endgroup

cg cg_inst = new();
```

```
initial
    foreach(values[i])
    begin
        y = values[i];
        cg_inst.sample(); //gather coverage
    end
endprogram
```

41. Why always block is not allowed in program block?

In program block, simulation ends when you complete the last statement in every initial-block, as this is considered the end of the test. It ends even if there are threads still running in the program or modules. Always block might execute on every posedge of clock. Even when program block is terminated, an always block which runs continuously is redundant. But initial forever could be used instead.

42. Which is best to use to model transaction? Struct or class ?

In this question, transaction is regarded as the sequence item concept in UVM, which usually needs to have the constraint random data, thus class is more suitable for it. Struct only groups the data type but not the methods. It can be synthesised, but only used when there is complex data type.

43. How SV is more random stable then Verilog?

SV allows we use the CRT(constrained-random tests), which Verilog cannot be used. By using constrained-random data, we can restrict the test scenarios to those that are both valid and of interest.

44. Difference between assert and expect statements?

An *expect* statement is very similar to an *assert* statement, but it must occur within a procedural block (including initial or always blocks, tasks and functions), and is used to block the execution until the property succeeds.

```
task mytask;
...
if (expr1)
    expect (my_property)
        pass_block();
    else // associated with the 'expect', not with the 'if'
        fail_block();
endtask
```

Unlike an *assert* statement, an *expect* statement starts only a single thread of evaluation. It comes out of the blocking mode only if the property succeeds or fails.

45. How to add a new process without disturbing the random number generator state?

constraint_mode()

46. What is the need of alias in SV?

The Verilog has one-way assign statement is a unidirectional assignment and can contain delay and strength change. To have bidirectional short-circuit connection SystemVerilog has added alias statement

```
module byte_rip (inout wire [31:0] W, inout wire [7:0] LSB, MSB);
    alias W[7:0] = LSB;
    alias W[31:24] = MSB;
endmodule
```

47. How can I model a bi-directional net with assignments influencing both source and destination?

The assign statement constitutes a continuous assignment. The changes on the RHS of the statement immediately reflect on the LHS net. However, any changes on the LHS don't get reflected on the RHS. System Verilog has introduced a keyword alias, which can be used only on nets to have a two-way assignment. For example, in the following code, any changes to the rhs is reflected to the lhs , and vice versa.

```
module test ();
wire rhs, lhs;
alias lhs=rhs;
endmodule
```

48. What is the need to implement explicitly a copy() method inside a transaction , when we can simple assign one object to other ?

If you assign one object to other then both the handles will point to the same object. This will not copy the transaction. By defining copy(), we can use the deep copy but not the shallow copy.

49. How different is the implementation of a struct and union in SV.

Struct:

To be able to share struct using ports and routines, you should create a type.

```
initial begin
    typedef struct {
        int a;
        byte b;
        shortint c;
        int d;
    } my_struct_s;

    my_struct_s st = '{
        32'haaaa_aaaad,
        8'hbb,
        16'hcccc,
        32'hdddd_dddd
    };
    $display("str = %x %x %x %x ", st.a, st.b, st.c, st.d);
```

end

Union

Unlike structures, the components of a union all refer to the same location in memory. In this way, a union can be used at various times to hold different types of objects, without the need to create a separate object for each new type.

```
typedef union { int i; real f; } num_u;
num_u un;
un.f = 0.0; // set value in floating point format
```

Unions are useful when you frequently need to read and write a register in several different formats. But class is more oftenly used.

50. What is "this"?

When we use a variable name, SystemVerilog looks in the current scope for it, and then in the parent scopes until the variable is found. “This” removes the ambiguity to let SystemVerilog know that you are assigning the local variable, to the class variable.

```
class Scoping;
    string oname;
    function new(string oname);
        this.oname = oname; // class oname = local oname
    endfunction
endclass
```

51. What is tagged union ?

A tagged union contains an implicit member that stores a tag, which represents the name of the last union member into which a value was stored/written. If value is read from a different union member than the member into which a tagged expression value was last written, an error message will shown.

```
union tagged {
    int i;
    real r;
} data;
data data_un;
data_un = tagged i 5; //store the 5 in data.i, and set it as the implicit tag.
d_out = data_un.i; //read value
d_out = data_un.r;//ERROR: memeber doesnt match the union's implicit tag
```

52. What is "scope resolution operator"?

Extern keyword allows out-of-body method declaration in classes. Scope resolution operator (::) links method construction to class declaration.

```
class XYZ;
        extern void task SayHello();
endclass
```

```
void task XYZ :: SayHello();
        $Message("Hello !!!\n");
endtask
```

53. What is the difference between?

(1.logic data_1;

(2.var logic data_2; //same as logic data_1

(3.wire logic data_3j;

(4.bit data_4;

(5.var bit data_5; //same as bit data_5

(6.var data_6; //same as logic data_6

For "Var", if a data type is not specified, then the data type logic shall be inferred

54. What is the difference between bits and logic?

bits: 2 states, default value is 0; logic: single bit, 4 states, default value is X

55. What is the difference between $rose and posedge?

```
always @(posedge clk)
    reg1 <= a & $rose(b);
```

In this example, the clocking event (posedge clk) is applied to $rose. $rose is true at the clock ticks when the sampled value of b changed to 1 from its sampled value at the previous tick of the clocking event.

(1 $rose returns true if the LSB of the expression changed to 1. Otherwise, it returns false.

(2 $fell returns true if the LSB of the expression changed to 0. Otherwise, it returns false.

(3 $stable returns true if the value of the expression did not change. Otherwise, it returns false.

posedge return an event, whereas $rose returns a Boolean value. Therefore they are not interchangeable.

56. What is advantage of program block over clock block w.r.t race condition?

Program schedules events in the Reactive region, the clocking block construct is very useful to automatically sample the steady-state values of previous time steps or clock cycles. Programs that read design values exclusively through clocking blocks with #0 input skews are insensitive to read-write races, for single clock.

57. How to avoid the race condition between programblock?

-->program block

--> clocking blocks

--> non-blocking driving (verilog style)

58. What is the difference between assumes and assert?

assert : This statement specifies if the propery holds correct.

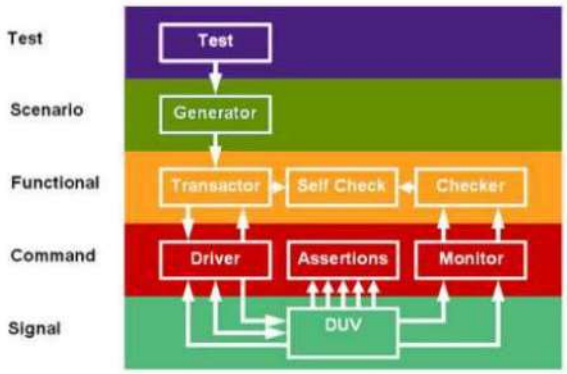
assume : This statement specifies property as assumption for the verification enviroment. This is more usefull with formal verification tools.

cover : This statement monitors property for the sake of coverage. Coverage can be made to be reported at the end of simulation.

59. What is coverage driven verification?

Coverage Driven Verification is a result oriented approach to functional verification.
A random test evolves using feedback. The initial test can be run with many different seeds, thus creating many unique input sequences. Eventually the test, even with a new seed, is less likely to generate stimulus that reaches areas of the design space. As the functional coverage asymptotically approaches its limit, you need to change the test to find new approaches to reach uncovered areas of the design. This is known as "coverage-driven verification".

60. What is layered architecture ?



61. What are the simulation phases in your verification environment?

Build Phase

- Generate configuration : Randomize the configuration of the DUT and surrounding environment
- *Build environment* : Allocate and connect the testbench components based on the configuration. (Connect Phase in UVM)

Run Phase

- *Start environment* : Run the testbench components such as BFMs and stimulus generators
- *Run the test* : Start the test and then wait for it to complete.

Wrap-up Phase

- *Sweep* : After the lowest layer completes, you need to wait for the final transactions to drain out of the DUT.
- *Report* : Once DUT is idle, sweep the testbench for lost data.

62. How to pick an element which is in queue from random index?

```
int q[$] = {1,2,3,4};
int index[4], out;
foreach(index[i])
```

```
        index[i] = i;
        index.shuffle(); //or index[i] = $urandom_range (1,4);
    foreach(q[i])
        out = q[index[i]];
```

63. What data structure is used to store data in your environment and why ?

Mailbox or queue. Because especially for the scoreboard, for each cycle, we have to collect data and drop these data from the driver and monitor respectively. Therefore, queue or mailbox would be more convenient than array.

64. Explain how the timescale unit and precision are taken when a module does not have any time scalar declaration in RTL?

In SV

timeunit 100ps;

timeprecision 10fs;

is as same as `timescale 100ps/10fs in Verilog

If a timeunit is not specified in the module, program, package, or interface definition, then the time unit shall be determined using the following rules of precedence:

a) If the module or interface definition is nested, then the time unit shall be inherited from the enclosing module or interface (programs and packages cannot be nested).

b) Else, if a `timescale directive has been previously specified (within the compilation unit), then the time unit shall be set to the units of the last `timescale directive.

c) Else, if the compilation-unit scope specifies a time unit (outside all other declarations), then the time unit shall be set to the time units of the compilation unit.

d) Else, the default time unit shall be used.

65. What is streaming operator and what is its use?

When used on the right side of an assignment, the streaming operators << and >> take an expression, structure, or array, and packs it into a stream of bits. The >> operator streams data from left to right while << streams from right to left.

66. What are void functions ?

The functions without return value, but they are still functions, thus no timing control allowed.

67. How to make sure that a function argument passed has ref is not changed by the function?

const ref

68. What is the difference between initial block and final block?

(1. the most obvious one: Initial blocks get executed at the beginning of the simulation, final block at the end of simulation

(2. Final block has to be executed in zero time, which implies it can't have any delay, wait, or non-blocking assignments.

(3. Final block can be used to display statistical/general information regarding the status of the execution

69. How to check weather a handles is holding object or not ?

It is basically checking if the object is initialized or not. In SV all uninitialized object handles have a special value of null. E.g assert(obj == NULL)

70. What are semaphores?

Ans: A semaphore is like a bucket and used in synchronization, which is also known as a "mutex"(mutually exclusive access) and is used to control access to a resource. When a semaphore is allocated, the keys are allocated to each and every bucket. The number is fixed. The keys are not identical. Processes using semaphores must definitely have a key from bucket before they start the execution process.

71. Why is reactive scheduler used?

Ans: Code specified in program blocks and pass/fail code from property expressions are scheduled in reactive scheduler.

72. What is the use of always_ff?

Ans: In an always block which is used to model combinational logic. forgetting an else leads to an unintended latch. To avoid this mistake, SystemVerilog adds specialized

always_comb: special always_comb procedure for modeling combinational logic behavior. The procedure is automatically triggered once at time zero, after all initial and always blocks have been started.

```
always_comb
begin
sum = b + a;
parity = ^sum;
end
```

always_latch: SystemVerilog also provides a special always_latch procedure for modeling latched logic behavior. Using "<="

```
always_latch
begin : ADDER
        if (enable) begin
        sum <= b + a;
        parity <= ^(b + a);
        end
end
```

always_ff: The SystemVerilog always_ff procedure can be used to model synthesizable sequential logic behavior

```
always_ff @(posedge clk iff rst == 0 or posedge rst)
begin : ADDER
if (rst) begin
        sum <= 0;
      parity <= 0;
  end
else begin
     sum   <= b + a;
     parity <= ^(b + a);
   end
end
```

73. What are static and automatic functions?

Ans: For overriding the values in the class, static function is used. Whereas in automatic, when one task is called, two separate memories will be allocated. The default one is static. Always use antuomatic for program block.

74. What is the procedure to assign elements in an array in systemverilog?

Ans: Assigning arrays in systemverilog uses the replicate operators.

Eg: int n[1:2][1:3]='{'{0,1,2),'{3{4}}};

75. What are the types of arrays in systemverilog?

Ans: There are two terminologies associated with the arrays in systemverilog. Packed and unpacked arrays. When the dimensions of arrays are declared before the object name is referred to as packed arrays. The unpacked array term is used to refer when the dimensions are declared after the object name. The memory allocation of both packed and unpacked arrays also differs.

E.g.: int [7:0] c1; //packed array

reg r[7:0] //unpacked array

76. Why are assertions used?

Ans: Assertions are mainly used to check the behavior of the design whether it is working correctly or not. They are useful in providing the functional coverage information .i.e. how good the test is and whether the design meets all the requirements for testing. There are mainly two types of assertions in systemverilog. They are: immediate assertions and concurrent assertions.

77. What is callback?

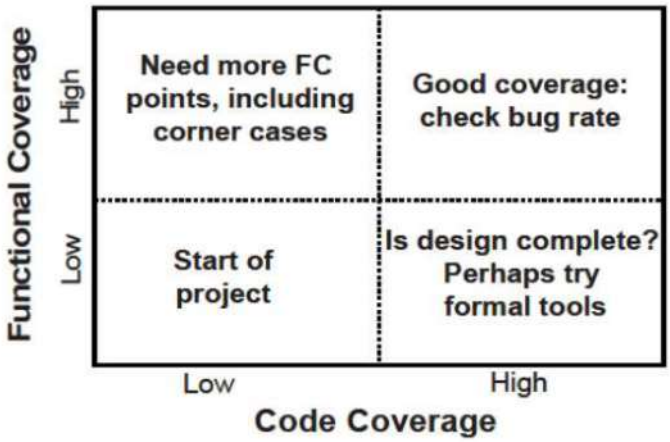
Ans: A callback is a built in systemverilog task/function. Suppose if we are transmitting a data using mailbox and you are sending data from design to transmitter. If you want to get back the data and you need the same data to put back in the scoreboard for comparison, this is called callback. Any change in the transmitted data can be achieved using the callback routine. Generally callbacks are called before transmission and after receiving the data.

78. What is the difference between code coverage and functional coverage?

Ans: Functional coverage: Functional coverage determines how much functionality of the design has been exercised. Functional assertions are used to check whether each and every corner of the design is explored and functions properly. Code coverage: This will give information about how many lines are executed, how many times each expression is executed. It describes the level up to which the code has been tested.

79. If the functional coverage is more that code coverage, what does it means?

Ans: code: High functional: High - You are most likely done
code: High functional: Low - Still need to run more simulation or improve test bench
code: Low functional: High - Probably missing functional coverage points
code: Low functional: Low - Still need to run more simulation or improve test bench



80. How we can have #delay which is independent of time scale in system verilog?

Ans: We can mention it as #5ns.

81. What are the different types of constraints in systemverilog?

Ans: Random constraints, user defined constraints, inline constraints, if-else constraints and global constraints.

82. What is an if-else constraint?

Ans: If the given expression is true, all the constraints in the first constraint block should be satisfied, otherwise all of the constraints in the else constraint block will be satisfied.

83. What is the difference between mailbox and queue?

Ans: Mailbox is similar to a queue, which allows only atomic operations. They can be bounded/unbounded. Get/put/peek task is used to suspend a bounded mailbox. That's why mailbox is

used more for communication between threads. Queues are large structures. Inserting data in queues is very difficult.

84. What is the significance of super keyword?

Ans: The super keyword is used from the derived class to refer to the members of the base class. If we want to access members of base class that are overridden by the derived class, super keyword is used. Super keyword cannot be accessed directly.

85. What are input and output skews in clocking block?

Ans: Skews are numbers that are indicated before the input and output ports. A skew number indicated before an input defines that the input is sampled before the clocking event occurs, ie a posedge or negedge occurs. A skew number in the output indicates that the output is synchronized and then sent to the clocking blocks. A skew must be a constant expression. It can also be specified as a parameter.

86. Mention the purpose of dividing time slots in systemverilog?

Ans: The main purpose of dividing the time slots in systemverilog is to provide interaction between the design and the testbench.

87. What is static variable?

Ans: In systemverilog we can create a static variable inside the class. But this variable has a limited scope. The variable declared static is shared among the other variables in the class. A static variable is usually instantiated inside the declaration.

88. In simulation environment under what condition the simulation should end?

Ans: 1) Packet count match
2) Error
3) Error count
4) Interface idle count
5) Global Time out

89. What is public declaration?

Ans: Objects in systemverilog can be declared as public and private. Public declarations are accessible from outside that object, i.e. they are accessible by the users. By default declarations in systemverilog are public

90. What is the use of local?

Ans: In order to make the declarations private, local attribute is used. Once the data is declared as local, it can be accessed on in the particular class.

91. How to take an asynchronous signal from clocking block?

Ans: By Using modports.

```
interface arb_if(input bit clk);
    logic [1:0] grant, request;
    logic rst;
    clocking cb @(posedge clk); // Declare cb
        output request;
        input grant;
    endclocking
    modport TEST (clocking cb, output rst); //rst is asynchronous signal
    modport DUT (input request, rst, output grant);
endinterface

module test(arb_if.TEST arbif);
    initial begin
        arbif.cb.request <= 0;
        @arbif.cb;
        $display("@%0t: Grant = %b", $time, arbif.cb.grant);
    end
endmodule
```

92. What is fork-join, types and differences?

Ans : There are three types of fork.. join blocks
fork .. join: The parent process blocks until all the processes spawned by this fork complete.
fork .. join_any: The parent process blocks until any one of the processes spawned by this fork Complete.
fork .. join_none: The parent process continues to execute concurrently with all the processes Spawned by the fork. The spawned processes do not start executing until the parent thread executes a blocking statement.

93. What is chandle in systemverilog ?

Ans. Chandle is a data type used to store pointers passed from DPI. Size of the chandle is machine dependent. Initialized value of the chandle is null. It can be used to pass arguments to functions or tasks.

94. What are the features added in systemverilog for function and task?

Ans: Begin and end not required. Function can have any number of input, output and inouts including none. Return can be used in task. Function return can have void return type.

95. What is DPI in systemverilog?

Ans: DPI (Direct Programming Interface) is used to call C, C++, System C functions.

96. What is Encapsulation?

Ans: Binds data and function together.

97. How to count number of elements in mailbox?

Ans: Mailbox is used for communication between two processes.
Ex: mailbox mbx;
mbx = new(); // Allocate mailbox
mbx.put (data); //Put data object in mailbox
mbx.get (data); // Data updated with new data from FIFO
count = mbx.num(); // To count number of elements is mailbox

98. What is covergroup?

Ans: Captures result from a random stimulation. Encapsulates the coverage specification.
Ex: enum { red, green, blue } color;
bit [3:0] pixel;
covergroupg1 @ (posedgeclk);
coverpoint color;
coverpoint pixel;
AxC: cross color, pixel;
endgroup

99. What are super, abstract and concrete classes?
Ans: Super class is the class from which child classes can share its methods,(base class/parent class). Abstract class is the class for which we create handles, which is also a class that can be extended, but not instantiated directly. It is defined with the virtual keyword. Extended classes of abstract classes are concrete classes.

100. What is Verification plan? What it contains?

Ans : Creating the real time environment for the (DUT) to check its performance in real time.
The verification plan closely tied with the hardware specification and contains a description of what features need to be verified.
Contains:
Directed testing
Random testing
Assertion
Hardware and software co-verification
Use of verification IP

101. Explain how messages are handled?

Ans: Using mailbox.

102.What is difference between define and parameter?

Ans: The parameter value can be changed during execution but not in the case of define construct/macros.

103. Difference between Associative and dynamic arrays?

Ans: Associative arrays basically used for very large memories. They have feature of string indexing. They executes at compile time.
Value is assigned to dynamic array at run time. New constructor is used to initialize the size of dynamic array.

104.How to check whether randomization is successful or not?
Ans : if (!obj.randomize())
begin
$display("Error in randomization");
$finish;
end
or using assert (!obj.randomize())

105.What is property in SVA?
Ans: 1. Number of sequences can be combined logically or sequentially to create more complex sequences. SVA provides a key word to represent these complex sequential behaviors called "property." 2. The basic syntax of a property is as follows.
property name_of_property;
< test expression > or
< complex sequence expressions >
endproperty;
assert_name_of_property: assert property (name_of_property);

106.What are immediate Assertions?

Ans: Immediate assertion is basically a statement that something must be true, similar to if statement. If assert evaluates to X, Z or 0, then the assertion fails and the simulator writes an error message.
my_assert:assert(condition1 & condition2)
$display("Passed..");
else
$error("Failed..");

107.What are Assertion severity system level task? What happens if we won't specify these tasks?
Ans : When we set property and if we won't specify failure case of the property, then by default language dictates simulator should give error as $error severity level.

$fatal - Run time fatal (quit Simulation)
$error - Run time error.
$warning ? Run time warning
$info ? Means this assertion carries no specific severity.

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

Ans: The variables used in a concurrent assertion are sampled in the Pre-poned region of a time slot and the assertions are evaluated during the Observe region. Both these regions occur immediately before a clock edge.

109.What are the main components in Concurrent Assertions?

Ans: In concurrent assertion there are three main components.
Sequence
Property
Assert property

110.What is Consecutive Repetition Operator in SVA?

Ans : Consecutive Repetition Operator [*]
It is to specify that a signal or sequence to match continuously for the number of specified clocks.

111.What is goto Replication operator in SVA?

Ans : This operator specifies that an expression will match the number of times specified not necessarily on continuous clock cycles.
Ex: x [->4:7]
x has been true 4, 5, 6 or 7 times, not necessarily on consecutive clocks.

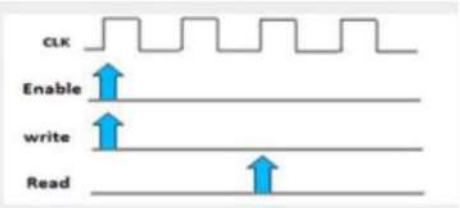
112.What is difference between x [->4:7] and x [=4:7] in SVA?

x [->4:7]	x has been true 4, 5, 6 or 7 times, not necessarily on consecutive clocks
x [=4:7]	x has been true 4,5,6 or 7 times, once again not necessarily on consecutive clocks, and with possible additional clocks after words when x is not true.

113.Implication operators only used inside the property.
Two types of operators

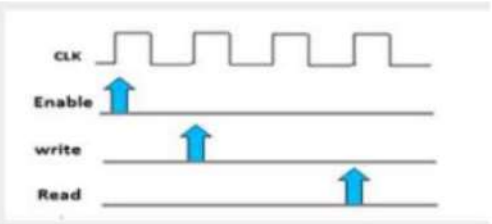
(1. Overlapping ( )

```
property pr1;
  always @(posedge clk) enable |-> (write ##2read);
endproperty

asr1:assert property(pr1);
```



(2. Non Overlapping ( )

```
property pr1;
  always @(posedge clk) enable |=> (write ##2read);
endproperty

asr1:assert property(pr1);
```



114.Can a constructor (new function) qualified as protected or local in systemverilog?

Ans : local

115.What are advantages of Interfaces?

Ans : a. Interface is ideal for design reuse.
b. It reduces the possibility of signal misconnection when the number of signals are large.
c. Easy to add new signals to the design

116.How automatic variables are useful in Threads?

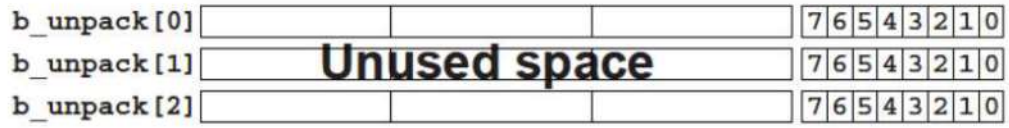
Ans : Sometimes we use loop that spawns threads and we don't save variable values before the next iteration. We should use the automatic variables inside a fork join statement to save the copy of a variable. Key word automatic create a copy of variable in each loop, cuz the stock/fifo storage mechanism.

example,
Sample 3.22 Specifying automatic storage in program blocks
program automatic test;
task wait_for_mem(input [31:0] addr, expect_data,
output success);
while (bus.addr !== addr)
@(bus.addr);
success = (bus.data == expect_data);
endtask
...
endprogram
You can call this task multiple times concurrently, as the addr and expect_data arguments are stored separately for each call. Without the automatic modifier, if you called wait_for_mem a second time while the first was still waiting, the second call would overwrite the two arguments.

117.Difference between unpacked and packed array.

Unpacked array:

bit [7:0] b_unpack[3]; // Unpacked



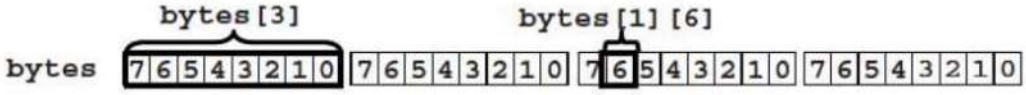
Packed array:

For some data types, you may want both to access the entire value and also to divide it into smaller elements. A SystemVerilog packed array is treated as both an array and a single value. It is stored as a contiguous set of bits with no unused space, unlike an unpacked array.

The packed bit and array dimensions are specified as part of the type, before the variable name.

bit [3:0] [7:0] bytes; // 4 bytes packed into 32-bits

bytes = 32'hCafe_Dada;



Mixed of packed and unpacked array:

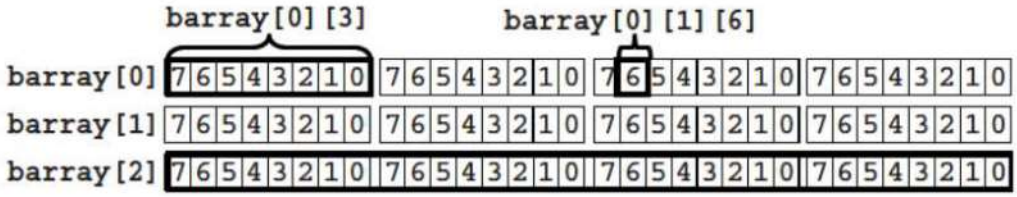
bit [3:0] [7:0] barray [3]; // Packed: 3x32-bit

bit [31:0] lw = 32'h0123_4567; // Word

barray[0] = lw;

barray[0][3] = 8'h01;

barray[0][1][6] = 1'h1;



With a single subscript, you get a word of data, barray[2]. With two subscripts, you get a byte of data, barray[0][3]. With three subscripts, you can access a single bit, barray[0][1][6].

Advantages of packed array:

(1. A packed array is handy if you need to convert to and from scalars. For example, you might need to reference a memory as a byte or as a word.

(2. If you need to wait for a change in an array, you have to use a packed array. Perhaps your testbench might need to wake up when a memory changes value, and so you want to use the @ operator. This is however only legal with scalar values and packed arrays. In the sample above you can block on barray[0], but not the entire array barray unless you expand it: @(barray[0] or barray[1] or barray[2]).

118.Dynamic array

```
int dyn[];
initial begin
    dyn = new[5]; // A: Allocate 5 elements
    foreach (dyn[j]) dyn[j] = j; // B: Initialize the elements
    dyn = new[20](dyn); // F: Allocate 20 ints & copy
    dyn = new[100]; // G: Allocate 100 new ints, Old values are lost
    dyn.size();
    dyn.delete(); // H: Delete all elements
end
```

119.Associate array

(1. When the size of the collection is unknown or the data space is sparse, an associateive array is a better option.

(2. Associative arrays do not have any storage allocated until it is used and the index expression is not restricted to integral expressions, but can be any type.

```
int power_of_2[int] = '{0:1, 1:2, 2:4 }; //0, 1, 2 serve as the index
initial begin
    for (int i=3; i<5; i++)
        power_of_2[i] = 1<<i;//result would be {1,2,4,8,16}
end
```

(3. power_of_2.exist(4) //check if 4 (element) has been allocated

120.Queue

Like a linked list, you can add or remove elements anywhere in a queue, without the performance hit of a dynamic array that has to allocate a new array and copy the entire contents. Like an array, you can directly access any element with an index, without linked list's overhead of stepping through the preceding elements.

```
int j, q2[$] = {3,4}, q[$] = {0,2,5};// Queue literals do not use '
bit [4:0] ack_que[$];
initial begin
    q.insert(1, j); // {0,1,2,5} Insert 1 before 2
    q.insert(3, q2); // {0,1,2,3,4,5} Insert queue in q1
    q.delete(1); // {0,2,3,4,5} Delete elem. #1
    // These operations are fast
    q.push_back(8); // {0,2,3,4,8} Insert at back
    j = q.pop_front; // {2,3,4,8} j = 0
    j = q[$]; //j=8
    foreach (q[i])
        $display(q[i]); // Print entire queue
    q.delete(); // {} Delete entire queue

end
```

121.Automatic routine

In verilog-1995, if you tried to call a task from multiple places in your testbench, the local variables shared common, static storage, and so the different threads stepped on each other's values. In Verilog-2001 you can specify that tasks, functions, and modules use automatic storage, which causes the simulator to use the stack for local variables.

```
program automatic test;
    task wait_for_mem(input [31:0] addr, expect_data, output success);
        while (bus.addr !== addr)
        @(bus.addr);
        success = (bus.data == expect_data);
    endtask
...
endprogram
```

You can call this task multiple times concurrently, as the addr and expect_data arguments are stored separately for each call. Without the automatic modifier, if you called wait_for_mem a second time while the first was still waiting, the second call would *overwrite* the two arguments.

122.Variable:Scope & lifetime

(1 Static

-Allocated & initialize at time 0;

-Exist for entire simulation.

(2 Automatic

-Enable recursive tasks and funtion

-Reallocated and initialized each time entering block

-Should not be use to trigger an event

(3Global Variable

-Defined under $root (outside any module)

-Must be static

-Accessible from anywhere

-Task and function can be global

(4 Local Variable

-Default to static. Can be automatic

-Accessible at where they are defined and below

-Accessible through hierarchical path

123.clocking skew

(1 inputs are sampled skew time units before clock

(2 outputs are driven skew time units after clock

(3 Default input skew is one step and output skew is 0. Step time is equal to the gobal time precision.

124.Two examples of inheritance

(1)

```
class parent;
    int a, b;
    virtual task display_c();
        $display("Parent");
    endtask
endclass

class child extends parent;
    int a;
    task display_c();
        $display("Child");
    endtask
endclass

initial begin
    parent p;
    child c;
    c = new();
    p = c;
    c.display_c();    //result: Child          Child
    p.display_c();    //result:Parent          Child
end
```

Comments: (1.without virtual function declared in the parent class, although the "p" points to the handle of "c", it's not allowed to call the function in the child class. With virtual, p can access to the child class's method. (2 if several child classes define the "display_c()" by the same name, it is called polymorphism.

(2

```
virtual class parent;
    pure virtual task display_c();
endclass

class child1 extends parent;
    task display_c();
        $display("Child1");
    endtask
endclass

class child2 extends parent;
    task display_c();
        $display("Child2");
    endtask
endclass
```

```
initial begin
    parent p[2];
    p[0] = new();        //illegal
    child1 c1 = new();
    child2 c2 = new();
    p[0] = c1;
    p[1] = c2;
    foreach(p[i]) begin
        p[i].display_c();
end
```

Comments: Assigning virtual class's handle to child object is also a way of polymorphsim.

125.Constructors in Extended Classes

If your base class constructor has any arguments, the constructor in the extended class must have a constructor and must call the base's constructor on its first line.

```
class Base1;
    int var;
    function new(input int var); // Has argument
        this.var = var;
    endfunction
endclass

class Extended extends Base1;
    function new(input int var); // Needs argument
        super.new(var); // Must be first line of new
        // Other constructor actions
    endfunction
endclass
```

126.X and Z in Verlog

(1 Z

-Output of an un-driven tri-state driver.

-Models case where nothing is setting a wire's value

(2 X

-Models when the simulator can't decide the value

-Initial state of registers

-When a wire is being driven to 0 and 1 simultaneously

-Output of a gate with Z input

(3 Uninitialized 4-state variables start at time-0 as unknown (X) values. Non-driven 4-state nets start at time-0 as floating, high impedance (Z) values

127. Event Region

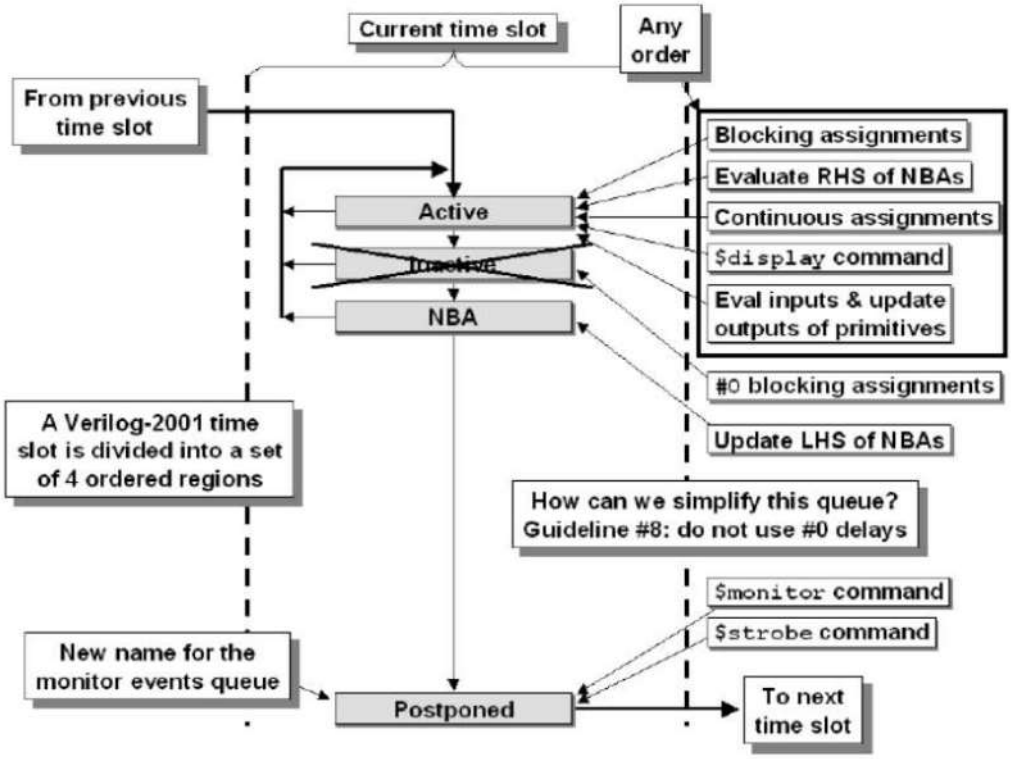
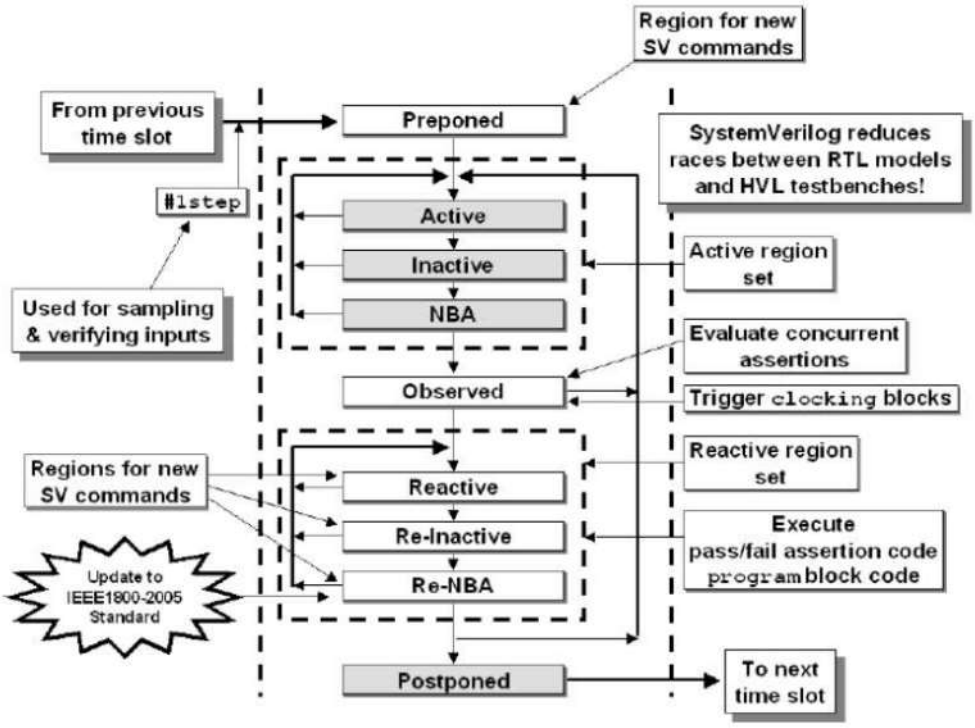
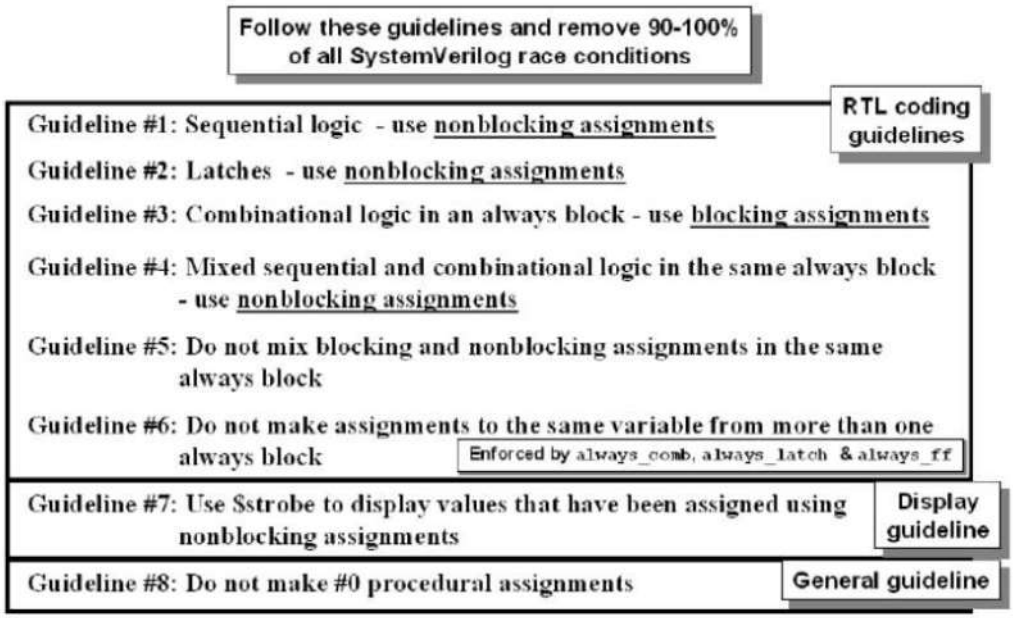


Figure 1 - Verilog-2001 event regions





128. Defining Control Fields ("Knobs")

It is not necessary (nor practical) to cover the entire legal space, but it is important to try back-to-back items along with short, medium, and large delays between the items, and combinations of all of these. To do this, define control fields (often called "knobs") to enable the test writer to control these variables. These same control knobs can also be used for coverage collection. For readability, use enumerated types to represent various generated categories.

```
typedef enum {ZERO, SHORT, MEDIUM, LARGE, MAX} simple_item_delay_e;
class simple_item extends uvm_sequence_item;
    rand int unsigned addr;
    rand int unsigned data;
    rand int unsigned delay;
    rand simple_item_delay_e delay_kind; // Control field
    // UVM automation macros for general objects
    `uvm_object_utils_begin(simple_item)
        `uvm_field_int(addr, UVM_ALL_ON)
        `uvm_field_enum(simple_item_delay_e, delay_kind, UVM_ALL_ON)
    `uvm_object_utils_end
    constraint delay_order_c { solve delay_kind before delay; }
    constraint delay_c {
        (delay_kind == ZERO) -> delay == 0;
        (delay_kind == SHORT) -> delay inside { [1:10] };
        (delay_kind == MEDIUM) -> delay inside { [11:99] };
        (delay_kind == LARGE) -> delay inside { [100:999] };
        (delay_kind == MAX ) -> delay == 1000;
        delay >=0; delay <= 1000;
    }
endclass : simple_item
```

129.1) What if design engineer and verification engineer do the same mistake in Test bench BFM (Bus Functional Model) and RTL (DUT)? How can you able to detect errors?

Answer:

1. Code reviews & protocol checkers

2. IP gets verified in multiple environmentslike block level test bench, out of box testbench (connecting DUT back to back) , fullfledged testbench using proven BFM, SoC level testbench using processor and all that etc... This all environments SHOULD be executed by different persons and so you should be able to catch that bug in one of this testbench.

3. Customer will catch the problem (worst case)

2) If you got a failure from the customer, how do you debug this? How do you prevent it to happen again?

Answer:

1. first, try to reproduce the problem in your own environment. Try to get customer's vector, so you can inject the same vector to create the problem in house.

2. If you confirm the problem and fix them, you should put the new assertion or test to catch the problem again. Add this new test in the future test plan, so the problem will not happen again.

130.Explain about pass by ref and pass by value?

Pass by value is the default method through which arguments are passed into functions and tasks. Each subroutine retains a local copy of the argument. If the arguments are changed within the subroutine declaration, the changes do not affect the caller.

In pass by reference functions and tasks directly access the specified variables passed as arguments.Its like passing pointer of the variable.

131.+ntb_opts uvm

$vcdplusdeltacycleon

$vcdplusmemon

1. $range = 100000000;
   $random_number = int(rand($range));
   +ntb_random_seed = $random_number; ##doesn't apply to Verilog $random

2. +ntb_random_seed_automatic

132.How to call the task which is defined in parent object into derived class ?

Answer:

The super keyword is used from within a derived class to refer to members of the parent class. It is necessary to use super to access members of a parent class when those members are overridden by the derived class

133. What is the difference between function overloading and function overriding?

A. Overloading is a method that allows defining multiple member functions with the same name but different signatures. The compiler will pick the correct function based on the signature. Overriding is a method that allows the derived class to redefine the behavior of member functions which the derived class inherits from a base class. The signatures of both base class member function and derived class member function are the same; however, the implementation and, therefore, the behavior will differ

134. What's the difference between data type logic, reg and wire?

Data Type	Wire	Reg	Logic
Assignments	Continuous assignments	blocking/non blocking assignment	Both continuous assignment or blocking/non blocking assignment
Limitation	Wire, cannot hold data	Storage element, store data until next assignment	extends the rand eg type so it can be driven by a single driver such as gate or module.

135. What is the need of clocking blocks?

- It is used to specify synchronization characteristics of the design
- It Offers a clean way to drive and sample signals
- Provides race-free operation if input skew > 0
- Helps in testbench driving the signals at the right time
- Features
  - Clock specification
  - Input skew, output skew
  - Cycle delay (##)
- Can be declared inside interface, module or program

136. Program non-blocking assignment,

```
int a = 1;
a = 2;
a<=3;
```

$display //will give you compile error (dynamic type cannot be used at LHS of non-blocking assignment, as here is automatic type), you cannot use $strobe in program, it will crushed.

```
static int a = 1;
```

```
a = 2;
a<=3;

$display //will give you a = 2;
```

137. Class can have module(to force the signals through hierarchy), module can also have class instant. (refer to 150)

Program cannot have module, interface, or other program; module can include program

Program can instant class, cannot instant other program, class cannot instant program

Program is the link between test-bench (class) and design (module)

138. Unbound mailbox, put() will not block.

```
Tx.qid =1;
Mailbox.put(tx);
Tx.qid =2;      //if we don't new Tx, the qid will be 2 at the get() side.
```

139. Shadow copy

```
class SAMPLE;
int a_m;
function new();
    a_m = 0;
endfunction
```

(1) Shadow copy

```
SAMPLE a, b;

a = new;
a.a_m = 40;
b = new a;      //function new will not copy
b.a_m = 50;     //a and b point to different handle but share the same a_m.
                //a_m will be 50
```

(2) Shadow copy also

```
SAMPLE a, b;

a = new;
b = a;      //b and a point to the same handle
a = new; //b point to the first handle, a point to the second one
```

(3) Bad OOP

```
SAMPLE a, b;
a = new;
b = a;      //b and a point to the same handle, any variable change in one
```

instance will affect the other

140. OOP questions

(1) Class A;

```
Constraint key_con {a == 6;}
…
endclass

Class B extends A;
Constraint key_con {a ==5;}      //overwrite should use the same name
…
endclass

…
A a = new;
Assert(!a.randomize(with {a == 7})) //will give you compile error
```

(2) Class B //think it as a uvm component

```
Uvm_build_phase
  Build_funtion…..
Endphase

Class C extends B      //question: 2 ways to get build funtions from class B

Build_phase

Endphase               //1. Use super.build_phase
                         2. do nothing, it inherited from B automatically
```

141.In uvm

In driver, how did you do pipelining?

Answer:

In systemverilog, you are using mailbox

In UVM, you should explicitly build queues to do pipelining.