1. **Question 1. What Is Uvm? What Is The Advantage Of Uvm?**
   Answer :
   UVM (Universal Verification Methodology) is a standardized methodology for verifying the both complex & simple digital design in simple way.

   UVM Features:
   - First methodology & second collection of class libraries for Automation
   - Reusability through test bench
   - Plug & Play of verification IPs
   - Generic Test bench Development
   - Vendor & Simulator Independent
   - Smart Test bench i.e. generate legal stimulus as from pre-planned coverage plan
   - Support CDV –Coverage Driven Verification
   - Support CRV –Constraint Random Verification
   - UVM standardized under the Accelerate System Initiative
   - Register modeling
   -
2. **Question 2. Uvm Derived From Which Language?**
   Answer :
   Here is the detailed connection between SV, UVM, OVM and other methodologies.

3. **Question 3. What Is The Difference Between Uvm_component And Uvm_object? Or We Already Have Uvm_object, Why Do We Need Uvm_component Which Is Actually Derived Class Of Uvm_object?**
   Answer :
   uvm_component:
   - Quasi Static Entity (after build phase it is available throughout the simulation).
   - Always tied to a given hardware (DUT Interface) or a TLM port.
   - Having phasing mechanism for control the behavior of simulation.
   - Configuration Component Topology.

   uvm_object:
   - Dynamic Entity (creates when needed, transfer from one component to other & then dereference).
   - Not tied to a given hardware or any TLM port.
   - Not phasing mechanism.
   -
4. **Question 4. Which Uvm Phase Is Top - Down, Bottom – Up & Parallel?**
   Answer :
   Only build phase is a top-down & other phases are bottom-up except run phase which is parallel. The build phase works top-down since the test bench hierarchy may be configure so we need to build the branches before leafs.

5. **Question 5. Why Build Phase Is Top – Down & Connect Phase Is Bottom – Up?**
   Answer :
   The connect phase is intended to be used for making TLM connections between components, which is why it occur after build phase. It work bottom-up so that its got the correct implementation all the way up the design hierarchy, if worked top-down this would be not possible.

6. **Question 6. Which Phase Is Function & Which Phase Is Task?**
   Answer :
   Only run phase is a task (time consuming phase) & other phases are functions (non-blocking).

7. **Question 7. Which Phase Takes More Time And Why?**
   Answer :
   As previously said the run phase is implemented as task and remaining all are function. run phase will get executed from start of simulation to till the end of simulation. run phase is time consuming, where the test case is running.

8. **Question 8. How Uvm Phases Initiate?**
   Answer :
   UVM phases initiate by calling run test ("test1") in top module. When run test() method call, it first create the object of test top & then call all phases.

9. **Question 9. How Test Cases Run From Simulation Command Line?**
   Answer :
   In top module write run test(); i.e. Don't give anything in argument.
   Then in command line: +UVM_TESTNAME=test name.

10. **Question 10. Difference Between Module & Class Based Tb?**
    Answer :
    - A module is a static object present always during of the simulation.
    - A Class is a dynamic object because they can come and go during the life time of simulation.

11. **Question 11. What Is Uvm_config_db? What Is Difference Between Uvm_config_db & Uvm_resource_db?**
    Answer :
    - Uvm_config_db is a parameterized class used for configuration of different type of parameter into the uvm database, So that it can be used by any component in the lower level of hierarchy.
    - Uvm_config_db is a convenience layer built on top of uvm_resource_db, but that convenience is very important. In particular, uvm_resource_db uses a "last write wins" approach. The uvm_config_db, on the other hand, looks at where things are in the hierarchy up through end_of_elaboration, so "parent wins." Once you start start_of_simulation, the config_db becomes "last write wins."

- All of the functions in uvm_config_db#(T) are static, so they must be called using the :: operator. It is extended from the uvm_resource_db#(T), so it is child class of uvm_resource_db#(T).

12. **Question 12. What Is The Advantage And Difference Of `uvm_component_utils() And `uvm_object_utils()?**

Answer :

- The utile macros define the infrastructure needed to enable the object/component for correct factory operation.
- The reason there are two macros is because the factory design pattern fixes the number of arguments that a constructor can have. Classes derived from uvm_object have constructors with one argument, a string name. Classes derived from uvm_component have two arguments, a name and a uvm_component parent.
- The two `uvm_*utile macros inserts code that gives you a factory create() method that delegates calls to the constructors of uvm_object or uvm_component. You need to use the respective macro so that the correct constructor arguments get passed through. This means that you cannot add extra constructor arguments when you extend these classes in order to be able to use the UVM factory.

13. **Question 13. Difference Between `uvm_do And `uvm_rand_send ?**

Answer :

uvm_do perform the below steps:

- Create
- Start item
- Randomize
- Finish item
- get response (optional)

While `uvm_rand_send perform all the above steps except create. User needs to create sequence / sequence item.

Question 14. Difference Between Uvm_transaction And Uvm_seq_item?

Answer :

class uvm_sequence_item extends uvm_transaction

Uvm_sequence_item extended from uvm_transaction only, uvm_sequence_item class has more functionality to support sequence & sequencer features. Uvm_sequence_item provides the hooks for sequencer and sequence, So you can generate transaction by using sequence and sequencer , and uvm_transaction provide only basic methods like do print and do record etc .

Question 15. Is Uvm Is Independent Of Systemverilog?

Answer :

UVM is a methodology based on Systemverilog language and is not a language on its own. It is a standardized methodology that defines several best practices in verification to enable efficiency in terms of reuse and is also currently part of IEEE 1800.2 working group.

Question 16. What Are The Benefits Of Using Uvm?

Answer :

Some of the benefits of using UVM are:

- **Modularity and Reusability** – The methodology is designed as modular components (Driver, Sequencer, Agents , env etc) which enables reusing components across unit level to multi-unit or chip level verification as well as across projects.
- **Separating Tests from Test benches** – Tests in terms of stimulus/sequencers are kept separate from the actual test bench hierarchy and hence there can be reuse of stimulus across different units or across projects.
- **Simulator independent** – The base class library and the methodology is supported by all simulators and hence there is no dependence on any specific simulator.
- **Better control on Stimulus generation** – Sequence methodology gives good control on stimulus generation. There are several ways in which sequences can be developed which includes randomization, layered sequences, virtual sequences etc which provides a good control and rich stimulus generation capability.
- **Easy configuration** – Config mechanisms simplify configuration of objects with deep hierarchy. The configuration mechanism helps in easily configuring different test bench components based on which verification environment uses it and without worrying about how deep any component is in test bench hierarchy.
- **Factory mechanism** – Factory mechanisms simplifies modification of components easily. Creating each components using factory enables them to be overridden in different tests or environments without changing underlying code base.

Question 17. Can We Have User Defined Phase In Uvm?

Answer :

In addition to the predefined phases available in uvm , the user has the option to add his own phase to a component. This is typically done by extending the uvm_phase class the constructor needs to call super. new which has three arguments.

- Name of the phase task or function
- Top down or bottom up phase
- Task or function

The call task or call_func and get_type_name need to be implemented to complete the addition of new phase.

Below is a simple example

Example:

```
Class custom phase extends uvm_phase;
  Function new ();
    Super. New (“custom”, 1, 1);
  End function
 Task call task (uvm_component parent);
   My_comp_type comp;
   If ( $cast(comp, parent) )
        comp.custom phase ();
```

```
  End task
  Virtual function string get_type_name ();
    Return “custom”;
  End function
End class
```

Question 18. What Is Uvm Ral Model? Why It Is Required?

Answer :

In a verification context, a register model (or register abstraction layer) is a set of classes that model the memory mapped behavior of registers and memories in the DUST in order to facilitate stimulus generation and functional checking (and optionally some aspects of functional coverage). The UVM provides a set of base classes that can be extended to implement comprehensive register modeling capabilities.

Question 19. What Is The Difference Between New() And Create?

Answer :

We all know about new () method that is use to allocate memory to an object instance. In UVM (and OVM), the create () method causes an object instance to be created from the factory. This allows you to use factory overrides to replace the desired object with an object of a different type without having to recode.

Question 20. What Is Analysis Port?

Answer :

Analysis port (class uvm_tlm_analysis_port) — a specific type of transaction-level port that can be connected to zero, one, or many analysis exports and through which a component may call the method write implemented in another component, specifically a subscriber.

port, export, and imp classes used for transaction analysis.

uvm_analysis_port

Broadcasts a value to all subscribers implementing a uvm_analysis_imp.

uvm_analysis_imp

Receives all transactions broadcasted by a uvm_analysis_port.

uvm_analysis_export

Exports a lower-level uvm_analysis_imp to its parent.

Question 21. What Is Tlm Fifo?

Answer :

In simpler words TLM FIFO is a FIFO between two UVM components, preferably between Monitor and Scoreboard. Monitor keep on sending the DATA, which will be stored in TLM FIFO, and Scoreboard can get data from TLM FIFO whenever needed.

```
// create a FIFO with depth 4
  tlm_fifo = new ("uvm tlm_fifo", this, 4);
```

Question 22. How Sequence Starts?

Answer :

Start item starts the sequence

Virtual task start item (uvm_sequence_item item,

int set priority = -1,

Uvm_sequencer_base sequencer = null)

Start item and finish item together will initiate operation of a sequence item. If the item has not already been initialized using create item, then it will be initialized here to use the default sequencer specified by m_sequencer.

Question 23. What Is The Difference Between Uvm Ral Model Backdoor Write/read And Front Door Write/read?

Answer :

- Font door access means using the standard access mechanism external to the DUTY to read or write to a register. This usually involves sequences of time-consuming transactions on a bus interface.
- Backdoor access means accessing a register directly via hierarchical reference or outside the language via the PLI. A backdoor reference usually in 0 simulation time.

Question 24. What Is Objection?

Answer :

- The objection mechanism in UVM is to allow hierarchical status communication among components which is helpful in deciding the end of test.
- There is a built-in objection for each in-built phase, which provides a way for components and objects to synchronize their testing activity and indicate when it is safe to end the phase and, ultimately, the test end.
- The component or sequence will raise a phase objection at the beginning of an activity that must be completed before the phase stops, so the objection will be dropped at the end of that activity. Once all of the raised objections are dropped, the phase terminates.

Raising an objection: phase.raise_objection (this);
Dropping an objection: phase.drop_objection (this);

Question 25. What Is M_sequencer? Or Difference Between M_sequencer And M_sequencer?

Answer :

M_sequencer is the default handle for uvm_vitual_sequencer and m_sequencer is the hook up for child sequencer.

M_sequencer is the generic uvm_sequencer pointer. It will always exist for the uvm_sequencer and is initialized when the sequence is started.

P_sequencer is a typed-specific sequencer pointer, created by registering the sequence to the sequencer using macros (`uvm_declare_p_sequencer) . Being type specific, you will be able to access anything added to the sequencer (i.e. pointers to other sequencers, etc.).

M_sequencer will not exist if we have not registered the sequence with the `uvm_declare_p_sequencer macros.

The drawback of m_sequencer is that once the m_sequencer is defined, one cannot run the sequence on any other sequencer type.

Question 26. What Is The Difference Between Active Mode And Passive Mode With Respect To Agent?

Answer :

An agent is a collection of a sequencer, a driver and a monitor.

In active mode, the sequencer and the driver are constructed and stimulus is generated by sequences sending sequence items to the driver through the sequencer. At the same time the monitor assembles pin level activity into analysis transactions.

In passive mode, only the monitor is constructed and it performs the same function as in an active agent. Therefore, your passive agent has no need for a sequencer. You can set up the monitor using a configuration object.

Question 27. What Is The Difference Between Copy And Clone?

Answer :

The built-in copy () method executes the __m_uvm_field_automation() method with the required copy code as defined by the field macros (if used) and then calls the built-in do copy() virtual function. The built-in do copy () virtual function, as defined in the uvm_object base class, is also an empty method, so if field macros are used to define the fields of the transaction, the built-in copy() method will be populated with the proper code to copy the transaction fields from the field macro definitions and then it will execute the empty do copy() method, which will perform no additional activity.

The copy() method can be used as needed in the UVM test bench. One common place where the copy() method is used is to copy the sampled transaction and pass it into a sb_calc_exp() (scoreboard calculate expected) external function that is frequently used by the scoreboard predictor.

The clone () method calls the create () method (constructs an object of the same type) and then calls the copy() method. It is a one-step command to create and copy an existing object to a new object handle.

Question 28. What Is Uvm Factory?

Answer :

UCM Factory is used to manufacture (create) UVM objects and components. Apart from creating the UVM objects and components the factory concept essentially means that you can modify or substitute the nature of the components created by the factory without making changes to the test bench.

For example, if you have written two driver classes, and the environment uses only one of them. By registering both the drivers with the factory, you can ask the factory to substitute the existing driver in environment with the other type. The code needed to achieve this is minimal, and can be written in the test.

Question 29. What Are The Types Of Sequencer? Explain Each?

Answer :

There are two types of sequencers:

uvm_sequencer #(REQ, RSP) :

When the driver initiates new requests for sequences, the sequencer selects a sequence from a list of available sequences to produce and deliver the next item to execute. In order to do this, this type of sequencer is usually connected to a driver uvm_driver #(REQ, RSP).

uvm_push_sequencer #(REQ, RSP) :

The sequencer pushes new sequence items to the driver, but the driver has the ability to block the item flow when it's not ready to accept any new transactions. This type of sequencer is connected to a driver of type uvm_push_driver # (REQ, RSP).

SYSTEM VERILOG

1. **Question 1. What Is Callback ?**

   Answer :

   In computer programming, a callback is executable code that is passed as an argument to other code. It allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.

2. **Question 2. What Is Factory Pattern ?**

   Answer :

   Factory Pattern Concept :

   Methodologies like OVM and VMM make heavy use of the factory concept. The factory method pattern is an object-oriented design pattern. Like other creational patterns, it deals with the problem of creating objects (products) without specifying the exact class of object that will be created. The factory method design pattern handles this problem by defining a separate method for creating the objects, whose subclasses can then override to specify the derived type of product that will be created. More generally, the term factory method is often used to refer to any method whose main purpose is creation of objects.

   Or in simple terms factory pattern help in creation of the object when you dont know the exact type of the object. the normal way of creating the object is :

```
01.// Normal Type based object creation
02.
03.// Class object
04.class my_class;
05.int i;
06.endclass
07.
08.program main;
09.// Create object type my_class
10.my_class obj1;
```

```
11.obj1 = new
12.endprogram
13.
14.// Using Factory I should be able to do the following
15.
16.program main;
17.base_class my_class_object;
18.
19.base_class = factory.create_object("my_class"); // See here the type of the object to be created is passed as a string so we dont know the exact type of the object
20.endprogram
```

3. **Question 3. Explain The Difference Between Data Types Logic And Reg And Wire ?**
   Answer :
   Wire are Reg are present in the verilog and system verilog adds one more data type called logic.

   Wire : Wire data type is used in the continuous assignments or ports list. It is treated as a wire So it can not hold a value. It can be driven and read. Wires are used for connecting different modules.
   Reg : Reg is a date storage element in system verilog. Its not a actual hardware register but it can store values. Register retain there value until next assignment statement.
   Logic : System verilog added this additional datatype extends the rand eg type so it can be driven by a single driver such as gate or module. The main difference between logic dataype and reg/wire is that a logic can be driven by both continuous assignment or blocking/non blocking assignment.

4. **Question 4. What Is The Need Of Clocking Blocks ?**
   Answer :
   - 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

   Example :

```
01.Module M1(ck, enin, din, enout, dout);
02.input        ck,enin;
03.input  [31:0] din    ;
04.output       enout  ;
05.output [31:0] dout   ;
06.
07.clocking sd @(posedge ck);
08.input  #2ns ein,din    ;
```

```
09.output #3ns enout, dout;
10.endclocking:sd
11.
12.reg [7:0] sab ;
13.initial begin
14.sab = sd.din[7:0];
15.end
16.endmodule:M1
```

5. **Question 5. What Are The Ways To Avoid Race Condition Between Testbench And Rtl Using Systemverilog?**

Answer :

There are mainly following ways to avoid the race condition between testbench and RTL using system verilog

- Program Block
- Clocking Block
- Using non blocking assignments.

Question 6. What Are The Types Of Coverages Available In Sv ?

Answer :

Using covergroup : variables, expression, and their cross
Using cover keyword : properties

Question 7. What Is Oops?

Answer :

Here are some nice OOP links on SystemVerilog OOP which can be used as a good starting point/reference.

- Object Oriented Programming for Hardware Verification
- Improve Your SystemVerilog OOP Skills by Learning Principles and Patterns
- SystemVerilog OOP OVM Feature Summary
- Enhancing SystemVerilog with AOP Concepts (On how to mimic AOP features in OOP, good for guyz coming from e background)
- Testbench.in OOP Tutorial

Question 8. What Is The Need Of Virtual Interfaces ?

Answer :

An interface encapsulate a group of inter-related wires, along with their directions (via modports) and synchronization details (via clocking block). The major usage of interface is to simplify the connection between modules.

But Interface can't be instantiated inside program block, class (or similar non-module entity in SystemVerilog). But they needed to be driven from verification environment like class. To solve this issue virtual interface concept was introduced in SV.

Virtual interface is a data type (that implies it 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 that make up the design. Another big advantage of virtual interface is that class can dynamically connect to different physical interfaces in run time.

Question 9. What Is The Difference Between Mailbox And Queue?

Answer :

A queue is a variable-size, ordered collection of homogeneous elements. A Queue is analogous to one dimensional unpacked array that grows and shrinks automatically. Queues can be used to model a last in, first out buffer or first in, first out buffer.

```
// Other data type as reference
// int q[]; dynamic array
// int q[5]; fixed array
// int q[string]; associate array
// include <
// List#(integer) List1;    //
int q[$] = { 2, 4, 8 };
int p[$];
int e, pos;
e = q[0]; // read the first (leftmost) item
e = q[$]; // read the last (rightmost) item
q[0] = e; // write the first item
p = q; // read and write entire queue (copy)
```

A mailbox is a communication mechanism that allows messages to be exchanged between processes. Data can be sent to a mailbox by one process and retrieved by another.

Question 10. What Data Structure You Used To Build Scoreboard?

Answer :

In SV, we use mailbox to get data from different modules and compare the result.

```
class Scoreboard;
mailbox drvr2sb;
mailbox rcvr2sb;
function new(mailbox drvr2sb,mailbox rcvr2sb);
  this.drvr2sb = drvr2sb;
  this.rcvr2sb = rcvr2sb;
endfunction:new
task start();
  packet pkt_rcv,pkt_exp;
  forever
  begin
    rcvr2sb.get(pkt_rcv);
    $display(" %0d : Scorebooard : Scoreboard received a packet from receiver ",$time);
    drvr2sb.get(pkt_exp);
    if(pkt_rcv.compare(pkt_exp))
    $display(" %0d : Scoreboardd :Packet Matched ",$time);
    else
      $root.error++;
```

```
  end
endtask : start
endclass
```

In VMM, we use channels to connect all the modules and compare the result.

```
class Scoreboard extends vmm_xactor;
   Packet_channel   drvr2sb_chan;
   Packet_channel   rcvr2sb_chan;
function new(string inst = "class",
             int unsigned stream_id = -1,
             Packet_channel   drvr2sb_chan = null,
             Packet_channel   rcvr2sb_chan = null);
      super.new("sb",inst,stream_id);
         if(drvr2sb_chan == null)
          `vmm_fatal(this.log,"drvr2sb_channel is not constructed");
      else
          this.drvr2sb_chan = drvr2sb_chan;
           if(rcvr2sb_chan == null)
          `vmm_fatal(this.log,"rcvr2sb_channel is not constructed");
      else
          this.rcvr2sb_chan = rcvr2sb_chan;
         `vmm_note(log,"Scoreboard created ");
endfunction:new
task main();
  Packet pkt_rcv,pkt_exp;
  string msg;
  super.main();
  forever
  begin
    rcvr2sb_chan.get(pkt_rcv);
    $display(" %0d : Scoreboard : Scoreboard received a packet from receiver ",$time);
    drvr2sb_chan.get(pkt_exp);
    if(pkt_rcv.compare(pkt_exp,msg))
    $display(" %0d : Scoreboard :Packet Matched ",$time);
    else
    `vmm_error(this.log,$psprintf(" Packet MissMatched n %s ",msg));
  end
endtask : main
endclass
```

Question 11. What Is The Difference Between $random() And $urandom()?

Answer :

- $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. (newly added in SV, not present in verilog)

Question 12. What Is Scope Randomization?

Answer :

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

```
01.module MyModule;
02.integer var, MIN;
03.
04.initial begin
05.MIN = 50;
06.for ( int i = 0;i begin
07.if( randomize(var) with { var < 100 ; var > MIN ;})
 08.$display(" Randomization sucsessfull : var = %0d Min = %0d",var,MIN);
09.else
10.$display("Randomization failed");
11.end
12.
13.$finish;
14.end
15.endmodule
```

Question 13. List The Predefined Randomization Methods.

Answer :

- randomize
- pre_randomize
- post_randomize

Question 14. What Is The Dfference Between Always_combo And Always@(*)?

Answer :

From SystemVerilog LRM 3.1a:-

- always_comb get executed once at time 0, always @* waits till a change occurs on a signal in the inferred sensitivity list
- Statement within always_comb can't have blocking timing, event control, or fork-join statement. No such restriction of always @*
- Optionally EDA tool might perform additional checks to warn if the behavior within always_comb procedure doesn't represent combinatorial logic
- 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.
- always_comb is sensitive to changes within content of a function, whereas always @* is only sensitive to changes to the arguments to the function.

A small SystemVerilog code snippet to illustrate #5

```
01.module dummy;
02.logic a, b, c, x, y;
03.
04.// Example void function
```

```
05.function void my_xor;
06.input a;         // b and c are hidden input here
07.x = a ^ b ^ c;
08.endfunction : my_xor
09.
10.function void my_or;
11.input a;         // b and c are hidden input here
12.y = a | b | c;
13.endfunction : my_xor
14.
15.always_comb          // equivalent to always(a,b,c)
16.my_xor(a);       // Hidden inputs are also added to sensitivity list
17.
18.always @*          // equivalent to always(a)
19.my_or(a);     // b and c are not added to sensitivity list
20.endmodule
```

Question 15. What Is The Use Of Packages?

Answer :

In Verilog declaration of data/task/function within modules are specific to the module only. They can't be shared between two modules. Agreed, we can achieve the same via cross module referencing or by including the files, both of which are known to be not a great solution.

The package construct of SystemVerilog aims in solving the above issue. It allows having global data/task/function declaration which can be used across modules. It can contain module/class/function/task/constraints/covergroup and many more declarations (for complete list please refer section 18.2 of SV LRM 3.1a)

The content inside the package can be accessed using either scope resolution operator (::), or using import (with option of referencing particular or all content of the package).

```
01.package ABC;
02.// Some typedef
03.typedef enum {RED, GREEN, YELLOW} Color;
04.
05.// Some function
06.void function do_nothing()
07....
08.endfunction : do_nothing
09.
10.// You can have many different declarations here
11.endpackage : ABC
12.
13.// How to use them
14.import ABC::Color;   // Just import Color
15.import ABC::*;     // Import everything inside the package
```

Question 16. What Is The Use Of $cast?

Answer :

Type casting in SV can be done either via static casting (', ', ') or dynamic casting via $cast task/function. $cast is very similar to dynamic_cast of C++. It checks whether the casting is possible or not in run-time and errors-out if casting is not possible.

Question 17. How To Call The Task Which Is Defined In Parent Object Into Derived Class ?

Answer :

super.task_name();

Question 18. What Is The Difference Between Rand And Randc?

Answer :

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. Resource intensive, use sparingly/judiciously

Question 19. What Is $root?

Answer :

$root refers to the top level instance in SystemVerilog

```
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
```

Question 20. What Are Bi-directional Constraints?

Answer :

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 -> 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)).

Question 21. What Is Solve And Before Constraint ?

Answer :

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. i.e.

```
1....
2.constraint XYZ  {
3.a inside {[0:100]|;
4.b < 20;
5.a + b > 30;
```

6.solve a before b;
7.}
The solution of the constraint doesn't change with solve before construct. But the probability of choosing a particular solution change by it.

Question 22. Without Using Randomize Method Or Rand,generate An Array Of Unique Values?

Answer :

```
1....
2.int UniqVal[10];
3.foreach(UniqVal[i]) UniqVal[i] = i;
4.UniqVal.shuffle();
5....
```

Question 23. Explain About Pass By Ref And Pass By Value?

Answer :

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.

example:

```
task pass(int i)    //  task pass(var int i) pass by reference
{
delay(10);
i = 1;
printf(" i is changed to %d at %dn",i,get_time(LO) );
delay(10);
i = 2;
printf(" i is changed to %d at %dn",i,get_time(LO) );
}
```

Question 24. What Is The Difference Between Byte And Bit [7:0]?

Answer :

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

Question 25. What Is The Difference Between Program Block And Module ?

Answer :

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

- It separates testbench from DUT
- It helps in ensuring that testbench doesn't have any race condition with DUT
- It provides an entry point for execution of testbench
- It provides syntactic context (via program ... endprogram) that specifies scheduling in the Reactive Region.

Having said this the major difference between module and program blocks are

- Program blocks can't have always block inside them, modules can have.
- Program blocks can't contain UDP, modules, or other instance of program block inside them. Modules don't have any such restrictions.
- 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
- Program blocks get executed in the re-active region of scheduling queue, module blocks get executed in the active region
- A program can call a task or function in modules or other programs. But a module can not call a task or function in a program.

Question 26. What Is The Use Of Modports ?

Answer :

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);

Question 27. Write A Clock Generator Without Using Always Block.

Answer :

Use of forever begin end. If it is a complex always block statement like always (@ posedge clk or negedge reset_)

```
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 : reset_logic
     @ (negedge reset_);
     data <= '0;
  end : reset_logic
  begin : clk_logic
     @ (posedge clk);
     if(!reset_)    data <= '0;
     else           data <= data_next;
  end : clk_logic
  join_any
```

```
  disable fork
end
```

Question 28. What Is Circular Dependency And How To Avoid This Problem ?

Answer :

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. Example

```
1....
2.int x, y, z;
3.constraint XYZ  {
4.solve x before y;
5.solve y before z;
6.solve z before x;
7.....
8.}
```

Question 29. What Is Cross Coverage ?

Answer :

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.

Question 30. How To Randomize Dynamic Arrays Of Objects?

Answer :

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

Question 31. What Is The Need Of Alias In Sv?

Answer :

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.

Question 32. What Is "this"?

Answer :

"this" pointer refers to current instance.

Question 33. What Is Tagged Union ?

Answer :

An union is used to stored multiple different kind/size of data in the same storage location.

```
1.typedef union{
2.bit [31:0]  a;
3.int         b;
4.} data_u;
```

Now here XYZ union can contain either bit [31:0] data or an int data. It can be written with a bit [31:0] data and read-back with a int data. There is no type-checking done.

In the case where we want to enforce that the read-back data-type is same as the written data-type we can use tagged union which is declared using the qualifier tagged. Whenever an union is defined as tagged, it stores the tag information along with the value (in expense of few extra bits). The tag and values can only be updated together using a statically type-checked tagged union expression. The data member value can be read with a type that is consistent with current tag value, making it impossible to write one type and read another type of value in tagged union. (the details of which can be found in section 3.10 and 7.15 of SV LRM 3.1a).

```
01.typedef union tagged{
02.bit [31:0]  a;
03.int         b;
04.} data_tagged_u;
05.
06.// Tagged union expression
07.data_tagged_u data1 = tagged a 32'h0;
08.data_tagged_u data2 = tagged b 5;
09.
10.// Reading back the value
11.int xyz = data2.b;
```

Question 34. What Is "scope Resolution Operator"?

Answer :

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

```
class XYZ;
// SayHello() will be declared outside the body
// of the class
extern void task SayHello();
endclass : XYZ
void task XYZ :: SayHello();
$Message("Hello !!!n");
endtask : SayHello
```

Question 35. What Is The Difference Between Bits And Logic?

Answer :

bits is 2-valued (1/0) and logic is 4-valued (0/1/x/z)

Question 36. What Is The Difference Between $rose And Posedge?

Answer :

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

Question 37. What Is Layered Architecture ?

Answer :

In SystemVerilog based constrained random verification environment, the test environment is divided into multiple layered as shown in the figure. It allows verification component re-use across verification projects.

Question 38. What Is The Difference Between Initial Block And Final Block?

Answer :

There are many difference between initial and final block. I am listing the few differences that is coming to mind now.

- The most obvious one : Initial blocks get executed at the beginning of the simulation, final block at the end of simulation
- Final block has to be executed in zero time, which implies it can't have any delay, wait, or non-blocking assignments. Initial block doesn't have any such restrictions of execution in zero time (and can have delay, wait and non-blocking statements)

Final block can be used to display statistical/genaral information regarding the status of the execution like this:-

```
1.final begin
2.$display("Simulation Passed");
3.$display("Final value of xyz = %h",xyz);
4.$display("Bye :: So long, and Thanks for all the fishes");
5.end
```

Question 39. How To Check Weather A Handles Is Holding Object Or Not ?

Answer :

It is basically checking if the object is initialized or not. In SystemVerilog all uninitialized object handles have a special value of null, and therefore whether it is holding an object or not can be found out by comparing the object handle to null. So the code will look like:-

```
01.usb_packet My_usb_packet;
02....
03.if(My_usb_packet == null) begin
04.// This loop will get exited if the handle is not holding any object
05.....
06.end else begin
07.// Hurray ... the handle is holding an object
```

08....
09.end

DIGITAL

1. **Question 1. What Is Difference Between Latch And Flip-flop?**
   Answer :
   The main difference between latch and FF is that latches are level sensitive while FF is edge sensitive. They both require the use of clock signal and are used in sequential logic. For a latch, the output tracks the input when the clock signal is high, so as long as the clock is logic 1, the output can change if the input also changes.

   FF on the other hand, will store the input only when there is a rising/falling edge of the clock. Latch is sensitive to glitches on enable pin, whereas flip-flop is immune to glitches. Latches take fewer gates (also less power) to implement than flip-flops. Latches are faster than flip-flops

2. **Question 2. Given Only Two Xor Gates One Must Function As Buffer And Another As Inverter?**
   Answer :
   Tie one of xor gates input to 1 it will act as inverter.
   Tie one of xor gates input to 0 it will act as buffer.

3. **Question 3. Difference Between Mealy And Moore State Machine?**
   Answer :
   A) Mealy and Moore models are the basic models of state machines. A state machine which uses only Entry Actions, so that its output depends on the state, is called a Moore model. A state machine which uses only Input Actions, so that the output depends on the state and also on inputs, is called a Mealy model. The models selected will influence a design but there are no general indications as to which model is better. Choice of a model depends on the application, execution means (for instance, hardware systems are usually best realized as Moore models) and personal preferences of a designer or programmer
   B) Mealy machine has outputs that depend on the state and input (thus, the FSM has the output written on edges) Moore machine has outputs that depend on state only (thus, the FSM has the output written in the state itself.
   Advantage and Disadvantage
   •In Mealy as the output variable is a function both input and state, changes of state of the state variables will be delayed with respect to changes of signal level in the input variables, there are possibilities of glitches appearing in the output variables.

   •Moore overcomes glitches as output dependent on only states and not the input signal level.
   •All of the concepts can be applied to Moore-model state machines because any Moore state machine can be implemented as a Mealy state machine, although the converse is not true.
   •Moore machine: the outputs are properties of states themselves... which means that you get the output after the machine reaches a particular state, or to get some output your machine has to be taken to a state which provides you the output. The outputs are held until you go to

some other state Mealy machine:
•Mealy machines give you outputs instantly, that is immediately upon receiving input, but the output is not held after that clock cycle.

4. **Question 4. Difference Between One Hot And Binary Encoding?**
   Answer :
   Common classifications used to describe the state encoding of an FSM are Binary (or highly encoded) and One hot.

   A binary-encoded FSM design only requires as many flip-flops as are needed to uniquely encode the number of states in the state machine. The actual number of flip-flops required is equal to the ceiling of the log-base-2 of the number of states in the FSM.A one hot FSM design requires a flip-flop for each state in the design and only one flip-flop (the flip-flop representing the current or "hot" state) is set at a time in a one hot FSM design.

   For a state machine with 9- 16 states, a binary FSM only requires 4 flip-flops while a one hot FSM requires a flip-flop for each state in the design FPGA vendors frequently recommend using a one hot state encoding style because flip-flops are plentiful in an FPGA and the combinational logic required to implement a one hot FSM design is typically smaller than most binary encoding styles.

   Since FPGA performance is typically related to the combinational logic size of the FPGA design, one hot FSMs typically run faster than a binary encoded FSM with larger combinational logic blocks

5. **Question 5. How To Achieve 180 Degree Exact Phase Shift?**
   Answer :
   Never tell using inverter
   a) DCM an inbuilt resource in most of FPGA can be configured to get 180 degree phase shift.
   b) BUFGDS that is differential signaling buffers which are also inbuilt resource of most of FPGA can be used.

6. **Question 6. What Is Significance Of Ras And Cas In Sdram?**
   Answer :
   SDRAM receives its address command in two address words. It uses a multiplex scheme to save input pins. The first address word is latched into the DRAM chip with the row address strobe (RAS).

   Following the RAS command is the column address strobe (CAS) for latching the second address word. Shortly after the RAS and CAS strobes, the stored data is valid for reading.

7. **Question 7. Tell Some Of Applications Of Buffer?**
   Answer :
   a) They are used to introduce small delays.
   b) They are used to eliminate cross talk caused due to inter electrode capacitance due to close routing.
   c) They are used to support high fan-out, e.g.: bufg

8. **Question 8. Give Two Ways Of Converting A Two Input Nand Gate To An Inverter?**
   Answer :
   a) Short the 2 inputs of the nand gate and apply the single input to it.
   b) Connect the output to one of the input and the other to the input signal.

9. **Question 9. Why Is Most Interrupts Active Low?**
   Answer :
   This answers why most signals are active low if you consider the transistor level of a module, active low means the capacitor in the output terminal gets charged or discharged based on low to high and high to low transition respectively. When it goes from high to low it depends on the pull down resistor that pulls it down and it is relatively easy for the output capacitance to discharge rather than charging. Hence people prefer using active low signals.

10. **Question 10. Design A Four-input Nand Gate Using Only Two-input Nand Gates.**
    Answer :
    Basically, you can tie the inputs of a NAND gate together to get an inverter.

11. **Question 11. What Will Happen If Contents Of Register Are Shifter Left, Right?**
    Answer :
    It is well known that in left shift all bits will be shifted left and LSB will be appended with 0 and in right shift all bits will be shifted right and MSB will be appended with 0 this is a straightforward answer What is expected is in a left shift value gets Multiplied by 2 e.g.: consider 0000_1110=14 a left shift will make it 0001_110=28, it the same fashion right shift will Divide the value by 2.

12. **Question 12. Given The Following Fifo And Rules, How Deep Does The Fifo Need To Be To Prevent Underflow Or Overflow?**
    Answer :
    RULES:
    1) frequency(clk_A) = frequency(clk_B) / 4
    2) period(en_B) = period(clk_A) * 100
    3) duty cycle(en_B) = 25%
    Assume clk_B = 100MHz (10ns)
    From (1), clk_A = 25MHz (40ns)
    From (2), period(en_B) = 40ns * 400 = 4000ns, but we only output for 1000ns,due to (3), so 3000ns of the enable we are doing no output work. Therefore, FIFO size = 3000ns/40ns = 75 entries

13. **Question 13. Differences Between D-latch And D Flip-flop?**
    Answer :
    D-latch is level sensitive where as flip-flop is edge sensitive. Flip-flops are made up of latches.

14. **Question 14. What Is A Multiplexer?**
    Answer :
    Is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
    (2n =>n). Where n is selection line.

15. **Question 15. What Are Set Up Time & Hold Time Constraints? What Do They Signify? Which One Is Critical For Estimating Maximum Clock Frequency Of A Circuit?**
    Answer :
    Set up time is the amount of time the data should be stable before the application of the clock signal, where as the hold time is the amount of time the data should be stable after the application of the clock. Setup time signifies maximum delay constraints; hold time is for minimum delay constraints. Setup time is critical for establishing the maximum clock frequency.

16. **Question 16. How Can You Convert An Sr Flip-flop To A Jk Flip-flop?**
    Answer :
    By giving the feedback we can convert, i.e. !Q=>S and Q=>R.Hence the S and R inputs will act as J and K respectively.

17. **Question 17. How Can You Convert The Jk Flip-flop To A D Flip-flop?**
    Answer :
    By connecting the J input to the K through the inverter.

18. **Question 18. How Do You Detect If Two 8-bit Signals Are Same?**
    Answer :
    XOR each bits of A with B (for e.g. A [0] xor B [0]) and so on. The o/p of 8 xor gates is then given as i/p to an 8-i/p nor gate.
    if o/p is 1 then A=B.

19. **Question 19. Convert D-ff Into Divide By 2. (not Latch) What Is The Max Clock Frequency The Circuit Can Handle, Given The Following Information?**
    Answer :
    T_setup= 6nsT_hold = 2nS T_propagation = 10nS
    Circuit: Connect Qbar to D and apply the clk at clk of DFF and take the O/P at Q. It gives freq/2. Max. Freq of operation: 1/ (propagation delay+setup time) = 1/16ns = 62.5 MHz

20. **Question 20. 7 Bit Ring Counter's Initial State Is 0100010. After How Many Clock Cycles Will It Return To The Initial State?**
    Answer :
    6 cycles

21. **Question 21. Design All The Gates (not, And, Or, Nand, Nor, Xor, Xnor) Using 2:1 Multiplexer?**

Answer :

Using 2:1 Mux, (2 inputs, 1 output and a select line)
a) NOT :Give the input at the select line and connect I0 to 1 & I1 to 0. So if A is 1, we will get I1 that is 0 at the O/P.
b) AND: Give input A at the select line and 0 to I0 and B to I1. O/p is A & B
c) OR: Give input A at the select line and 1 to I1 and B to I0. O/p will be A | B
d) NAND: AND + NOT implementations together
e) NOR: OR + NOT implementations together
f) XOR: A at the select line B at I0 and ~B at I1. ~B can be obtained from (a)
g) XNOR: A at the select line B at I1 and ~B at I0

22. **Question 22. Design A Circuit That Calculates The Square Of A Number?**

Answer :

It should not use any multiplier circuits. It should use Multiplexers and other logic?

1^2=0+1=1
2^2=1+3=4
3^2=4+5=9
4^2=9+7=16
5^2=16+9=25
See a pattern yet? To get the next square, all you have to do is add the next odd number to the previous square that you found. See how 1,3,5,7 and finally 9 are added. Wouldn't this be a possible solution to your question since it only will use a counter, multiplexer and a couple of adders? It seems it would take n clock cycles to calculate square of n.

23. **Question 23. N Number Of Xnor Gates Is Connected In Series Such That The N Inputs (a0, A1, A2.....) Are Given In The Following Way: A0 & A1 To First Xnor Gate And A2 & O/p Of First Xnor To Second Xnor Gate And So On..... Nth Xnor Gates Output Is Final Output. How Does This Circuit Work? Explain In Detail?**

Answer :

If N=Odd, the circuit acts as even parity detector, i.e. the output will 1 if there are even number of 1's in the N input...This could also be called as odd parity generator since with this additional 1 as output the total number of 1's will be ODD. If N=Even, just the opposite, it will be Odd parity detector or Even Parity Generator.

24. **Question 24. What Is Race-around Problem? How Can You Rectify It?**

Answer :

The clock pulse that remains in the 1 state while both J and K are equal to 1 will cause the output to complement again and repeat complementing until the pulse goes back to 0, this is called the race around problem. To avoid this undesirable operation, the clock pulse must have a time duration that is shorter than the propagation delay time of the F-F, this is restrictive so the alternative is master-slave or edge-triggered construction.

25. **Question 25. An Assembly Line Has 3 Fail Safe Sensors And One Emergency Shutdown Switch. The Line Should Keep Moving Unless Any Of The Following Conditions Arise:**
Answer :
(i) If the emergency switch is pressed
(ii) If the senor1 and sensor2 are activated at the same time.
(iii) If sensor 2 and sensor3 are activated at the same time.
(iv) If all the sensors are activated at the same time
suppose a combinational circuit for above case is to be implemented only with NAND Gates. How many minimum number of 2 input NAND gates are required?

No of 2-input NAND Gates required = 6 you can try the whole implementation.

26. **Question 26. How Will You Implement A Full Subtractor From A Full Adder?**
Answer :
All the bits of subtrahend should be connected to the xor gate. Other input to the xor being one. The input carry bit to the full adder should be made 1. Then the full adder works like a full subtract.

27. **Question 27. What Is Difference Between Setup And Hold Time. The Interviewer Was Looking For One Specific Reason, And Its Really A Good Answer Too..the Hint Is Hold Time Doesn't Depend On Clock, Why Is It So...?**
Answer :
Setup violations are related to two edges of clock, i mean you can vary the clock frequency to correct setup violation. But for hold time, you are only concerned with one edge and do not basically depend on clock frequency.

28. **Question 28. In A 3-bit Johnson's Counter What Are The Unused States?**
Answer :
2(power n)-2n is the one used to find the unused states in Johnson counter.

So for a 3-bit counter it is 8-6=2.Unused states=2. the two unused states are 010 and 101.

29. **Question 29. What Is Difference Between Ram And Fifo?**
Answer :
FIFO does not have address lines

Ram is used for storage purpose where as FIFO is used for synchronization purpose i.e. when two peripherals are working in different clock domains then we will go for FIFO.

30. **Question 30. Consider Two Similar Processors, One With A Clock Skew Of 100ps And Other With A Clock Skew Of 50ps. Which One Is Likely To Have More Power? Why?**
Answer :
Clock skew of 50ps is more likely to have clock power. This is because it is likely that low-skew processor has better designed clock tree with more powerful and number of buffers and overheads to make skew better.

31. **Question 31. Is It Possible To Reduce Clock Skew To Zero? Explain Your Answer?**
Answer :
Even though there are clock layout strategies (H-tree) that can in theory reduce clock skew to zero by having the same path length from each flip-flop from the pll, process variations in R and C across the chip will cause clock skew as well as a pure H-Tree scheme is not practical (consumes too much area).

32. **Question 32. The Circle Can Rotate Clockwise And Back. Use Minimum Hardware To Build A Circuit To Indicate The Direction Of Rotating?**
Answer :
2 sensors are required to find out the direction of rotating. They are placed like at the drawing. One of the m is connected to the data input of D flip-flop, and a second one - to the clock input. If the circle rotates the way clock sensor sees the light first while D input (second sensor) is zero - the output of the flip-flop equals zero, and if D input sensor "fires" first - the output of the flip-flop becomes high.

33. **Question 33. You Have Two Counters Counting Upto 16, Built From Negedge Dff , First Circuit Is Synchronous And Second Is "ripple" (cascading), Which Circuit Has A Less Propagation Delay? Why?**
Answer :
The synchronous counter will have lesser delay as the input to each flop is readily available before the clock edge. Whereas the cascade counter will take long time as the output of one flop is used as clock to the other. So the delay will be propagating. For E.g.: 16 state counter = 4 bit counter = 4 Flip flops Let 10ns be the delay of each flop The worst case delay of ripple counter = 10 * 4 = 40ns The delay of synchronous counter = 10ns only.(Delay of 1 flop)

34. **Question 34. Difference Between Synchronous And Asynchronous Reset?**
Answer :
Synchronous reset logic will synthesize to smaller flip-flops, particularly if the reset is gated with the logic generating the dinput. But in such a case, the combinational logic gate count grows, so the overall gate count savings may not be that significant. The clock works as a filter for small reset glitches; however, if these glitches occur near the active clock edge, the Flip-flop could go metastable. In some designs, the reset must be generated by a set of internal conditions. A synchronous reset is recommended for these types of designs because it will filter the logic equation glitches between clocks.

Disadvantages of synchronous reset:
Problem with synchronous resets is that the synthesis tool cannot easily distinguish the reset signal from any other data signal. Synchronous resets may need a pulse stretcher to guarantee a reset pulse width wide enough to ensure reset is present during an active edge of the clock. if you have a gated clock to save power, the clock may be disabled coincident with the assertion of reset. Only an asynchronous reset will work in this situation, as the reset might be removed prior to the resumption of the clock. Designs that are pushing the limit for data path timing, cannot afford to have added gates and additional net delays in the data path due to logic inserted to handle synchronous resets.

Asynchronous reset:
The biggest problem with asynchronous resets is the reset release, also called reset removal. Using an asynchronous reset, the designer is guaranteed not to have the reset added to the data path. Another advantage favoring asynchronous resets is that the circuit can be reset with or without a clock present.

Disadvantages of asynchronous reset: ensure that the release of the reset can occur within one clock period. if the release of the reset occurred on or near a clock edge such that the flip-flops went metastable.

35. **Question 35. Implement The Following Circuits:**
Answer :
(a) 3 input NAND gate using min no of 2 input NAND Gates
(b) 3 input NOR gate using min no of 2 input NOR Gates
(c) 3 input XNOR gate using min no of 2 input XNOR Gates
Assuming 3 inputs A,B,C?
3 input NAND Connect:
a) A and B to the first NAND gate
b) Output of first Nand gate is given to the two inputs of the second NAND gate (this basically realizes the inverter functionality)4
c) Output of second NAND gate is given to the input of the third NAND gate, whose other input is C ((A NAND B) NAND (A NAND B)) NAND C Thus, can be implemented using '3' 2-input NAND gates. I guess this is the minimum number of gates that need to be used.

VERILOG

1. **Question 1. Write A Verilog Code To Swap Contents Of Two Registers With And Without A Temporary Register?**
Answer :
With temp reg ;

```
always @ (posedge clock)
begin
temp=b;
b=a;
a=temp;
end
```

Without temp reg;

```
always @ (posedge clock)
begin
a <= b;
b <= a;
end
```

2. **Question 2. Difference Between Task And Function?**

Answer :

Function:

- A function is unable to enable a task however functions can enable other functions.
- A function will carry out its required duty in zero simulation time. (The program time will not be incremented during the function routine)
- Within a function, no event, delay or timing control statements are permitted
- In the invocation of a function their must be at least one argument to be passed.
- Functions will only return a single value and can not use either output or inout statements.

Tasks:

- Tasks are capable of enabling a function as well as enabling other versions of a Task
- Tasks also run with a zero simulation however they can if required be executed in a non zero simulation time.
- Tasks are allowed to contain any of these statements.
- A task is allowed to use zero or more arguments which are of type output, input or inout.
- A Task is unable to return a value but has the facility to pass multiple values via the output and inout statements .

3. **Question 3. Difference Between Inter Statement And Intra Statement Delay?**

Answer :

```
//define register variables
reg a, b, c;
//intra assignment delays
initial
begin
a = 0; c = 0;
b = #5 a + c; //Take value of a and c at the time=0, evaluate
//a + c and then wait 5 time units to assign value
//to b.
end
//Equivalent method with temporary variables and regular delay control
initial
begin
a = 0; c = 0;
temp_ac = a + c;
#5 b = temp_ac; //Take value of a + c at the current time and
//store it in a temporary variable. Even though a and c
//might change between 0 and 5,
//the value assigned to b at time 5 is unaffected.
end
```

4. **Question 4. Difference Between $monitor,$display & $strobe?**
   Answer :
   These commands have the same syntax, and display text on the screen during simulation. They are much less convenient than waveform display tools like cwaves?. $display and $strobe display once every time they are executed, whereas $monitor displays every time one of its parameters changes.

   The difference between $display and $strobe is that $strobe displays the parameters at the very end of the current simulation time unit rather than exactly where it is executed. The format string is like that in C/C++, and may contain format characters. Format characters include %d (decimal), %h (hexadecimal), %b (binary), %c (character), %s (string) and %t (time), %m (hierarchy level). %5d, %5b etc. would give exactly 5 spaces for the number instead of the space needed. Append b, h, o to the task name to change default format to binary, octal or hexadecimal.

   Syntax:
   $display ("format_string", par_1, par_2, ...);
   $strobe ("format_string", par_1, par_2, ...);
   $monitor ("format_string", par_1, par_2, ...);

5. **Question 5. What Is Difference Between Verilog Full Case And Parallel Case?**
   Answer :
   A "full" case statement is a case statement in which all possible case-expression binary patterns can be matched to a case item or to a case default. If a case statement does not include a case default and if it is possible to find a binary case expression that does not match any of the defined case items, the case statement is not "full."

   A "parallel" case statement is a case statement in which it is only possible to match a case expression to one and only one case item. If it is possible to find a case expression that would match more than one case item, the matching case items are called "overlapping" case items and the case statement is not "parallel."

6. **Question 6. What Is Meant By Inferring Latches,how To Avoid It?**
   Answer :
   Consider the following :

   ```
   always @(s1 or s0 or i0 or i1 or i2 or i3)
   case ({s1, s0})
   2'd0 : out = i0;
   2'd1 : out = i1;
   2'd2 : out = i2;
   endcase
   ```

   in a case statement if all the possible combinations are not compared and default is also not specified like in example above a latch will be inferred ,a latch is inferred because to reproduce the previous value when unknown branch is specified.

   For example in above case if {s1,s0}=3 , the previous stored value is reproduced for this storing a latch is inferred.

   The same may be observed in IF statement in case an ELSE IF is not specified.

To avoid inferring latches make sure that all the cases are mentioned if not default condition is provided.

7. **Question 7. Tell Me How Blocking And Non Blocking Statements Get Executed?**

Answer :

Execution of blocking assignments can be viewed as a one-step process:

1. Evaluate the RHS (right-hand side equation) and update the LHS (left-hand side expression) of the blocking assignment without interruption from any other Verilog statement. A blocking assignment "blocks" trailing assignments in the same always block from occurring until after the current assignment has been completed

Execution of nonblocking assignments can be viewed as a two-step process:

- Evaluate the RHS of nonblocking statements at the beginning of the time step.
- Update the LHS of nonblocking statements at the end of the time step.
-

Question 8. Variable And Signal Which Will Be Updated First?

Answer :

Signals

Question 9. What Is Sensitivity List?

Answer :

The sensitivity list indicates that when a change occurs to any one of elements in the list change, begin…end statement inside that always block will get executed.

Question 10. In A Pure Combinational Circuit Is It Necessary To Mention All The Inputs In Sensitivity Disk? If Yes, Why?

Answer :

Yes in a pure combinational circuit is it necessary to mention all the inputs in sensitivity disk other wise it will result in pre and post synthesis mismatch.

Question 11. Tell Me Structure Of Verilog Code You Follow?

Answer :

A good template for your Verilog file is shown below.

```
// timescale directive tells the simulator the base units and precision of the simulation
`timescale 1 ns / 10 ps
module name (input and outputs);
// parameter declarations
parameter parameter_name = parameter value;
// Input output declarations
input in1;
input in2; // single bit inputs
output [msb:lsb] out; // a bus output
// internal signal register type declaration - register types (only assigned within always statements). reg register
```

```
variable 1;
reg [msb:lsb] register variable 2;
// internal signal. net type declaration - (only assigned outside always statements) wire net
variable 1;
// hierarchy - instantiating another module
reference name instance name (
.pin1 (net1),
.pin2 (net2),
.
.pinn (netn)
);
// synchronous procedures
always @ (posedge clock)
begin
.
end
// combinatinal procedures
always @ (signal1 or signal2 or signal3)
begin
.
end
assign net variable = combinational logic;
endmodule
```

Question 12. Difference Between Verilog And Vhdl?

Answer :

Compilation

VHDL. Multiple design-units (entity/architecture pairs), that reside in the same system file, may be separately compiled if so desired. However, it is good design practice to keep each design unit in it's own system file in which case separate compilation should not be an issue.

Verilog. The Verilog language is still rooted in it's native interpretative mode. Compilation is a means of speeding up simulation, but has not changed the original nature of the language. As a result care must be taken with both the compilation order of code written in a single file and the compilation order of multiple files. Simulation results can change by simply changing the order of compilation.

Data types

VHDL. A multitude of language or user defined data types can be used. This may mean dedicated conversion functions are needed to convert objects from one type to another. The choice of which data types to use should be considered wisely, especially enumerated (abstract) data types. This will make models easier to write, clearer to read and avoid unnecessary conversion functions that can clutter the code. VHDL may be preferred because it allows a multitude of language or user defined data types to be used.

Verilog. Compared to VHDL, Verilog data types a re very simple, easy to use and very much geared towards modeling hardware structure as opposed to abstract hardware modeling. Unlike VHDL, all data types used in a Verilog model are defined by the Verilog language and not by the user. There are net data types, for example wire, and a register data type called reg.

A model with a signal whose type is one of the net data types has a corresponding electrical wire in the implied modeled circuit. Objects, that is signals, of type reg hold their value over simulation delta cycles and should not be confused with the modeling of a hardware register. Verilog may be preferred because of it's simplicity.

Design reusability

VHDL. Procedures and functions may be placed in a package so that they are avail able to any design-unit that wishes to use them.

Verilog. There is no concept of packages in Verilog. Functions and procedures used within a model must be defined in the module. To make functions and procedures generally accessible from different module statements the functions and procedures must be placed in a separate system file and included using the `include compiler directive.

Question 13. Can You Tell Me Some Of System Tasks And Their Purpose?

Answer :

$display, $displayb, $displayh, $displayo, $write, $writeb, $writeh, $writeo.

The most useful of these is $display.This can be used for displaying strings, expression or values of variables.

Here are some examples of usage.

```
$display("Hello oni");
--- output: Hello oni
$display($time) // current simulation time.
--- output: 460
counter = 4'b10;
$display(" The count is %b", counter);
--- output: The count is 0010
```

$reset resets the simulation back to time 0; $stop halts the simulator and puts it in interactive mode where the user can enter commands; $finish exits the simulator back to the operating system

Question 14. Can You List Out Some Of Enhancements In Verilog 2001?

Answer :

In earlier version of Verilog ,we use 'or' to specify more than one element in sensitivity list . In Verilog 2001, we can use comma as shown in the example below.

```
// Verilog 2k example for usage of comma
always @ (i1,i2,i3,i4)
```

Verilog 2001 allows us to use star in sensitive list instead of listing all the variables in RHS of combo logics . This removes typo mistakes and thus avoids simulation and synthesis mismatches, Verilog 2001 allows port direction and data type in the port list of modules as shown in the example below

```
module memory (
input r,
input wr,
input [7:0] data_in,
input [3:0] addr,
```

output [7:0] data_out
);

Question 15. Write A Verilog Code For Synchronous And Asynchronous Reset?

Answer :

Synchronous reset, synchronous means clock dependent so reset must not be present in sensitivity disk

eg: always @ (posedge clk)

begin if (reset)
. . . end

Asynchronous means clock independent so reset must be present in sensitivity list.

Eg: Always @(posedge clock or posedge reset)

begin
if (reset)
. . . end

Question 16. What Is Pli?why Is It Used?

Answer :

Programming Language Interface (PLI) of Verilog HDL is a mechanism to interface Verilog programs with programs written in C language. It also provides mechanism to access internal databases of the simulator from the C program.

PLI is used for implementing system calls which would have been hard to do otherwise (or impossible) using Verilog syntax. Or, in other words, you can take advantage of both the paradigms - parallel and hardware related features of Verilog and sequential flow of C - using PLI.

Question 17. There Is A Triangle And On It There Are 3 Ants One On Each Corner And Are Free To Move Along Sides Of Triangle What Is Probability That They Will Collide?

Answer :

Ants can move only along edges of triangle in either of direction, let's say one is represented by 1 and another by 0, since there are 3 sides eight combinations are possible, when all ants are going in same direction they won't collide that is 111 or 000 so probability of not collision is 2/8=1/4 or collision probability is 6/8=3/4

Question 18. How To Write Fsm Is Verilog?

Answer :

there r mainly 4 ways 2 write fsm code

- using 1 process where all input decoder, present state, and output decoder r combine in one process.
- using 2 process where all comb ckt and sequential ckt separated in different process
- using 2 process where input decoder and persent state r combine and output decoder seperated in other process

- using 3 process where all three, input decoder, present state and output decoder r separated in 3 process.
-

Question 19. What Is Difference Between Freeze Deposit And Force?

Answer :

$deposit(variable, value);

This system task sets a Verilog register or net to the specified value. variable is the register or net to be changed; value is the new value for the register or net. The value remains until there is a subsequent driver transaction or another $deposit task for the same register or net. This system task operates identically to the ModelSim force -deposit command.

The force command has -freeze, -drive, and -deposit options. When none of these is specified, then -freeze is assumed for unresolved signals and -drive is assumed for resolved signals. This is designed to provide compatibility with force files. But if you prefer -freeze as the default for both resolved and unresolved signals.

Question 20. Will Case Infer Priority Register If Yes How Give An Example?

Answer :

yes case can infer priority register depending on coding style

```
reg r;
// Priority encoded mux,
always @ (a or b or c or select2)
begin
r = c;
case (select2)
2'b00: r = a;
2'b01: r = b;
endcase
end
```

Question 21. Given The Following Verilog Code, What Value Of "a" Is Displayed?

Answer :

```
always @(clk) begin
a = 0;
a <= 1;
$display(a);
end
```

This is a tricky one! Verilog scheduling semantics basically imply a four-level deep queue for the current simulation time:

- Active Events (blocking statements)
- Inactive Events (#0 delays, etc)
- Non-Blocking Assign Updates (non-blocking statements)
- Monitor Events ($display, $monitor, etc).

Since the "a = 0" is an active event, it is scheduled into the 1st "queue".

The "a <= 1" is a non-blocking event, so it's placed into the 3rd queue.

Finally, the display statement is placed into the 4th queue. Only events in the active queue are completed this sim cycle, so the "a = 0" happens, and then the display shows a = 0. If we were to look at the value of a in the next sim cycle, it would show 1.

Question 22. What Is The Difference Between The Following Two Lines Of Verilog Code?

Answer :

#5 a = b;
a = #5 b;
#5 a = b;
Wait five time units before doing the action for "a = b;".

a = #5 b; The value of b is calculated and stored in an internal temp register,After five time units, assign this stored value to a.

Question 23. What Does `timescale 1 Ns/ 1 Ps Signify In A Verilog Code?

Answer :

'timescale directive is a compiler directive.It is used to measure simulation time or delay time. Usage :`timescale / reference_time_unit : Specifies the unit of measurement for times and delays. time_precision: specifies the precision to which the delays are rounded off.

Question 24. What Is The Difference Between === And == ?

Answer :

output of "==" can be 1, 0 or X.
output of "===" can only be 0 or 1.
When you are comparing 2 nos using "==" and if one/both the numbers have one or more bits as "x" then the output would be "X" . But if use "===" outpout would be 0 or 1.

e.g A = 3'b1x0
B = 3'b10x
A == B will give X as output.
A === B will give 0 as output.
"==" is used for comparison of only 1's and 0's .It can't compare Xs. If any bit of the input is X output will be X

"===" is used for comparison of X also.

Question 25. How To Generate Sine Wav Using Verilog Coding Style?

Answer :

The easiest and efficient way to generate sine wave is using CORDIC Algorithm.

Question 26. What Is The Difference Between Wire And Reg?

Answer :

(wire,tri)Physical connection between structural elements. Value assigned by a continuous assignment or a gate output. Register type: (reg, integer, time, real, real time) represents abstract data storage element. Assigned values only within an always statement or an initial

statement. The main difference between wire and reg is wire cannot hold (store) the value when there no connection between a and b like a->b, if there is no connection in a and b, wire loose value. But reg can hold the value even if there in no connection. Default values:wire is Z,reg is x.

Question 27. How Do You Implement The Bi-directional Ports In Verilog Hdl?

Answer :

```
module bidirec (oe, clk, inp, outp, bidir);
// Port Declaration
input oe;
input clk;
input [7:0] inp;
output [7:0] outp;
inout [7:0] bidir;
reg [7:0] a;
reg [7:0] b;
assign bidir = oe ? a : 8'bZ ;
assign outp = b;
// Always Construct
always @ (posedge clk)
begin
b <= bidir;
a <= inp;
end
endmodule
```

Question 28. What Is Verilog Case (1) ?

Answer :

```
wire [3:0] x;
always @(...) begin
case (1'b1)
x[0]: SOMETHING1;
x[1]: SOMETHING2;
x[2]: SOMETHING3;
x[3]: SOMETHING4;
endcase
end
```

The case statement walks down the list of cases and executes the first one that matches. So here, if the lowest 1-bit of x is bit 2, then something3 is the statement that will get executed (or selected by the logic).

Question 29. Why Is It That "if (2'b01 & 2'b10)..." Doesn't Run The True Case?

Answer :

This is a popular coding error. You used the bit wise AND operator (&) where you meant to use the logical AND operator (&&).

Question 30. What Are Different Types Of Verilog Simulators ?

Answer :

There are mainly two types of simulators available.

- Event Driven
- Cycle Based

Event-based Simulator:

This Digital Logic Simulation method sacrifices performance for rich functionality: every active signal is calculated for every device it propagates through during a clock cycle. Full Event-based simulators support 4-28 states; simulation of Behavioral HDL, RTL HDL, gate, and transistor representations; full timing calculations for all devices; and the full HDL standard. Event-based simulators are like a Swiss Army knife with many different features but none are particularly fast.

Cycle Based Simulator:

This is a Digital Logic Simulation method that eliminates unnecessary calculations to achieve huge performance gains in verifying Boolean logic:

- Results are only examined at the end of every clock cycle; and
- The digital logic is the only part of the design simulated (no timing calculations). By limiting the calculations, Cycle based Simulators can provide huge increases in performance over conventional Event-based simulators.

Cycle based simulators are more like a high speed electric carving knife in comparison because they focus on a subset of the biggest problem: logic verification.

Cycle based simulators are almost invariably used along with Static Timing verifier to compensate for the lost timing information coverage.

VLSI

1. **Question 1. Why Does The Present Vlsi Circuits Use Mosfets Instead Of Bjts?**

   Answer :

   Compared to BJTs, MOSFETs can be made very small as they occupy very small silicon area on IC chip and are relatively simple in terms of manufacturing. Moreover digital and memory ICs can be implemented with circuits that use only MOSFETs i.e. no resistors, diodes, etc.

2. **Question 2. What Are The Various Regions Of Operation Of Mosfet? How Are Those Regions Used?**

   Answer :

   MOSFET has three regions of operation: the cut-off region, the triode region, and the saturation region.

   The cut-off region and the triode region are used to operate as switch. The saturation region is used to operate as amplifier.

3. **Question 3. What Is Threshold Voltage?**
   Answer :
   The value of voltage between Gate and Source i.e. V_{GS} at which a sufficient number of mobile electrons accumulate in the channel region to form a conducting channel is called threshold voltage (V_t is positive for NMOS and negative for PMOS).

4. **Question 4. What Does It Mean "the Channel Is Pinched Off"?**
   Answer :
   For a MOSFET when V_{GS} is greater than V_t, a channel is induced. As we increase VDS current starts flowing from Drain to Source (triode region). When we further increase V_{DS}, till the voltage between gate and channel at the drain end to become V_t, i.e. $V_{GS} - V_{DS} = V_t$, the channel depth at Drain end decreases almost to zero, and the channel is said to be pinched off. This is where a MOSFET enters saturation region.

5. **Question 5. Explain The Three Regions Of Operation Of A Mosfet?**
   Answer :
   Cut-off region: When $V_{GS} < V_t$, no channel is induced and the MOSFET will be in cut-off region. No current flows.
   Triode region: When $V_{GS} \geq V_t$, a channel will be induced and current starts flowing if $V_{DS} > 0$. MOSFET will be in triode region as long as $V_{DS} < V_{GS} - V_t$.
   Saturation region: When $V_{GS} \geq V_t$, and $V_{DS} \geq V_{GS} - V_t$, the channel will be in saturation mode, where the current value saturates. There will be little or no effect on MOSFET when V_{DS} is further increased.

6. **Question 6. What Is Channel-length Modulation?**
   Answer :
   In practice, when V_{DS} is further increased beyond saturation point, it does has some effect on the characteristics of the MOSFET. When V_{DS} is increased the channel pinch-off point starts moving away from the Drain and towards the Source. Due to which the effective channel length decreases, and this phenomenon is called as Channel Length Modulation.

7. **Question 7. Explain Depletion Region.**
   Answer :
   When a positive voltage is applied across Gate, it causes the free holes (positive charge) to be repelled from the region of substrate under the Gate (the channel region). When these holes are pushed down the substrate they leave behind a carrier-depletion region.

8. **Question 8. What Is Body Effect?**
   Answer :
   Usually, in an integrated circuit there will be several MOSFETs and in order to maintain cut-off condition for all MOSFETs the body substrate is connected to the most negative power supply (in case of PMOS most positive power supply). Which causes a reverse bias voltage between source and body that effects the transistor operation, by widening the depletion region. The widened depletion region will result in the reduction of channel depth. To restore the channel depth to its normal depth the VGS has to be increased. This is effectively seen as

change in the threshold voltage - V_t. This effect, which is caused by applying some voltage to body is known as body effect.

9. **Question 9. Give Various Factors On Which Threshold Voltage Depends?**
   Answer :
   As discussed in the above question, the V_t depends on the voltage connected to the Body terminal. It also depends on the temperature, the magnitude of V_t decreases by about 2mV for every 1°C rise in temperature.

10. **Question 10. What Are The Steps Required To Solve Setup And Hold Violations In Vlsi?**
    Answer :
    There are few steps that has to be performed to solved the setup and hold violations in VLSI. The steps are as follows:
    - The optimization and restructuring of the logic between the flops are carried way. This way the logics are combined and it helps in solving this problem.
    - There is way to modify the flip-flops that offer lesser setup delay and provide faster services to setup a device. Modifying the launch-flop to have a better hold on the clock pin, which provides CK->Q that makes the launch-flop to be fast and helps in fixing the setup violations.
    - The network of the clock can be modified to reduce the delay or slowing down of the clock that captures the action of the flip-flop.
    - There can be added delay/buffer that allows less delay to the function that is used.
    -

11. **Question 11. What Are The Different Ways In Which Antenna Violation Can Be Prevented?**
    Answer :
    Antenna violation occurs during the process of plasma etching in which the charges generating from one metal strip to another gets accumlated at a single place. The longer the strip the more the charges gets accumulated. The prevention can be done by following method:
    - Creating a jogging the metal line, that consists of atleast one metal above the protected layer.
    - There is a requirement to jog the metal that is above the metal getting the etching effect. This is due to the fact that if a metal gets the etching then the other metal gets disconnected if the prevention measures are not taken.
    - There is a way to prevent it by adding the reverse Diodes at the gates that are used in the circuits.

12. **Question 12. What Is The Function Of Tie-high And Tie-low Cells?**
    Answer :
    Tie-high and tie-low are used to connect the transistors of the gate by using either the power or the ground. The gates are connected using the power or ground then it can be turned off and on due to the power bounce from the ground. The cells are used to stop the bouncing and easy from of the current from one cell to another. These cells are required Vdd that connects to the tie-high cell as there is a power supply that is high and tie-low gets connected to Vss.

This connection gets established and the transistors function properly without the need of any ground bounce occurring in any cell.

13. **Question 13. What Is The Main Function Of Metastability In Vsdl?**

Answer :

Metastability is an unknown state that is given as neither one or zero. It is used in designing the system that violates the setup or hole time requirements. The setup time requirement need the data to be stable before the clock-edge and the hold time requires the data to be stable after the clock edge has passed. There are potential violation that can lead to setup and hold violations as well. The data that is produced in this is totally asynchronous and clocked synchronous. This provide a way to setup the state through which it can be known that the violations that are occuring in the system and a proper design can be provided by the use of several other functions.

14. **Question 14. What Are The Steps Involved In Preventing The Metastability?**

Answer :

Metastability is the unknown state and it prevents the violations using the following steps:

- proper synchronizers are used that can be two stage or three stage whenever the data comes from the asynchronous domain. This helps in recovering the metastable state event.
- The synchronizers are used in between cross-clocking domains. This reduces the metastability by removing the delay that is caused by the data element that are coming and taking time to get removed from the surface of metal.
- Use of faster flip-flops that allow the transaction to be more faster and it removes the delay time between the one component to another component. It uses a narrower metastable window that makes the delay happen but faster flip-flops help in making the process faster and reduce the time delay as well.
-

Question 15. What Are The Different Design Constraints Occur In The Synthesis Phase?

Answer :

The steps that are involved in which the design constraint occurs are:

- first the creation of the clock with the frequency and the duty cycle gets created. This clock helps in maintaining the flow and synchronizing various devices that are used.
- Define the transition time according the requirement on the input ports.
- The load values are specified for the output ports that are mapped with the input ports.
- Setting of the delay values for both the input and output ports. The delay includes the input and output delay.
- Specify the case-settings to report the correct time that are matched with the specific paths.
- The clock uncertainty values are setup and hold to show the violations that are occurring.

Question 16. What Are The Different Types Of Skews Used In Vlsi?

Answer :

There are three types of skew that are used in VLSI. The skew are used in clock to reduce the delay or to understand the process accordingly. The skew are as follows:

Local skew: This contain the difference between the launching flip-flop and the destination flip-flop. This defines a time path between the two.

Global skew: Defines the difference between the earliest component reaching the flip flow and the the latest arriving at the flip flow with the same clock domain. In this delays are not measured and the clock is provided the same.

Useful skew: Defines the delay in capturing a flip flop paths that helps in setting up the environment with specific requirement for the launch and capture of the timing path. The hold requirement in this case has to be met for the design purpose.

Question 17. What Are The Changes That Are Provided To Meet Design Power Targets?

Answer :

To meet the design power target there should be a process to design with Multi-VDD designs, this area requires high performance, and also the high VDD that requires low-performance. This is used to create the voltage group that allow the appropriate level-shifter to shift and placed in cross-voltage domains. There is a design with the multiple threshold voltages that require high performance when the Vt becomes low.

This have lots of current leakage that makes the Vt cell to lower the performance. The reduction can be performed in the leakage power as the clock in this consume more power, so placing of an optimal clock controls the module and allow it to be given more power. Clock tree allow the switching to take place when the clock buffers are used by the clock gating cells and reduce the switching by the power reduction.

Question 18. What Are The Different Measures That Are Required To Achieve The Design For Better Yield?

Answer :

To achieve better yeild then there should be reduction in maufacturability flaws. The circuit perfomance has to be high that reduces the parametric yield. This reduction is due to process variations The measures that can be taken are:

- Creation of powerful runset files that consists of spacing and shorting rules. This also consists of all the permissions that has to be given to the user.
- Check the areas where the design is having lithographic issues, that consists of sharp cuts.
- Use of redundant vias to reduce the breakage of the current and the barrier.
- Optimal placing of the de-coupling capacitances can be done so that there is a reduction in power-surges.

Question 19. What Is The Difference Between The Mealy And Moore State Machine?

Answer :

- Moore model consists of the machine that have an entry action and the output depends only on the state of the machine, whereas mealy model only uses Input Actions and the output depends on the state and also on the previous inputs that are provided during the program.
- Moore models are used to design the hardware systems, whereas both hardware and software systems can be designed using the mealy model.
- Mealy machine's output depend on the state and input, whereas the output of the moore machine depends only on the state as the program is written in the state only.
- Mealy machine is having the output by the combination of both input and the state and the change the state of state variables also have some delay when the change in the signal takes place, whereas in Moore machine doesn't have glitches and its ouput is dependent only on states not on the input signal level.

Question 20. What Is The Difference Between Synchronous And Asynchronous Reset?

Answer :

- Synchronous reset is the logic that will synthesize to smaller flip-flops. In this the clock works as a filter providing the small reset glitches but the glitches occur on the active clock edge, whereas the asynchronous reset is also known as reset release or reset removal. The designer is responsible of added the reset to the data paths.
- The synchronous reset is used for all the types of design that are used to filter the logic glitches provided between the clocks. Whereas, the circuit can be reset with or without the clock present.
- Synchronous reset doesn't allow the synthesis tool to be used easily and it distinguishes the reset signal from other data signal. The release of the reset can occur only when the clock is having its initial period. If the release happens near the clock edge then the flip-flops can be metastable.

Question 21. What Are The Different Design Techniques Required To Create A Layout For Digital Circuits?

Answer :

The different design techniques to create the Layout for digital circuits are as follows:

- Digital design consists of the standard cells and represent the height that is required for the layout. The layout depends on the size of the transistor. It also consists of the specification for Vdd and GND metal paths that has to be maintained uniformly.
- Use of metal in one direction only to apply the metal directly. The metal can be used and displayed in any direction.
- Placing of the substrate that place where it shows all the empty spaces of the layout where there is resistances.
- Use of fingered transistors allows the design to be more easy and it is easy to maintain a symmetry as well.

Question 22. Write A Program To Explain The Comparator?

Answer :

To make a comparator there is a requirement to use multiplexer that is having one input and many outputs. This allows the choosing of the maximum numbers that are required to design the comparator. The implementation of the 2 bit comparator can be done using the law of tigotomy that states that A > B, A < B, A = B (Law of trigotomy). The comparator can be implemented using:

combinational logic circuits or multiplexers that uses the HDL language to write the schematic at RTL and gate level.

Behavioral model of comparator represented like:

```
module comp0 (y1,y2,y3,a,b);
input [1:0] a,b;
output y1,y2,y3;
wire y1,y2,y3;
assign y1= (a >b)? 1:0;
assign y2= (b >a)? 1:0;
assign y3= (a==b)? 1:0;
endmodule
```

Question 23. What Is The Function Of Chain Reordering?

Answer :

The optimization technique that is used makes it difficult for the chain ordering system to route due to the congestion caused by the placement of the cells. There are tool available that automate the reordering of the chain to reduce the congestion that is produced at the first stage. It increases the problem of the chain system and this also allow the overcoming of the buffers that have to be inserted into the scan path.

The increase of the hold time in the chain reordering can cause great amount of delay. Chain reordering allows the cell to be come in the ordered format while using the different clock domains. It is used to reduce the time delay caused by random generation of the element and the placement of it.

Question 24. What Are The Steps Involved In Designing An Optimal Pad Ring?

Answer :

- To make the design for an optimal pad ring there is a requirement for the corner-pads that comes across all the corners of the pad-ring. It is used to give power continuity and keep the resistance low.
- It requires the pad ring that is to fulfil the power domains that is common for all the ground across all the domains.
- It requires the pad ring to contain simultaneous switching noise system that place the transfer cell pads in cross power domains for different pad length.
- Drive strength is been seen to check the current requirements and the timings to make the power pads.
- Choose a no-connection pad that is used to fill the pad-frame when there is no requirement for the inputs to be given. This consumes less power when there is no input given at a particular time.
- Checking of the oscillators pads take place that uses the synchronous circuits to make the clock data synchronize with the existing one.

Question 25. What Is The Function Of Enhancement Mode Transistor?

Answer :

The enhancement mode transistors are also called as field effect transistors as they rely on the electric filed to control the shape and conductivity of the channel. This consists of one type of charge carrier in a semiconductor material environment. This also uses the unipolar transistors to differentiate themselves with the single-carrier type operation transistors that consists of the bipolar junction transistor.

The uses of field effect transistor is to physical implementation of the semiconductor materials that is compared with the bipolar transistors. It provides with the majority of the charge carrier devices. The devices that consists of active channels to make the charge carriers pass through. It consists of the concept of drain and the source.

Question 26. What Is The Purpose Of Having Depletion Mode Device?

Answer :

Depletion modes are used in MOSFET it is a device that remains ON at zero gate-source voltage. This device consists of load resistors that are used in the logic circuits. This types are used in N-type depletion-load devices that allow the threshold voltages to be taken and use of -3 V to +3V is done.

The drain is more positive in this comparison of PMOS where the polarities gets reversed. The mode is usually determined by the sign of threshold voltage for N-type channel. Depletion mode is the positive one and used in many technologies to represent the actual logic circuit. It defines the logic family that is dependent on the silicon VLSI. This consists of pull-down switches and loads for pull-ups.

Question 27. What Is The Difference Between Nmos And Pmos Technologies?

Answer :

- PMOS consists of metal oxide semiconductor that is made on the n-type substrates and consists of active careers named as holes. These holes are used for migration purpose of the charges between the p-type and the drain. Whereas, NMOS consists of the metal oxide semiconductor and they are made on p-type substrates. It consists of electrons as their carriers and migration happens between the n-type source and drain.
- On applying the high voltage on the logic gates NMOS will be conducted and will get activated, whereas PMOS require low voltage to be activated.
- NMOS are faster than PMOS as the carriers that NMOS uses are electrons that travels faster than holes. The speed is twice as fast as holes.
- PMOS are more immune to noice than NMOS.

Question 28. What Is The Difference Between Cmos And Bipolar Technologies?

Answer :

- CMOS technology allows the power dissipation to be low and it gives more power output, whereas bipolar takes lots of power to run the system and the ciricutary require lots of power to get activated.
- CMOS technology provides high input impedance that is low drive current that allow more current to be flown in the cirucit and keep the circuit in a good position, whereas it provides high drive current means more input impedance.
- CMOS technology provides scalable threshold voltage more in comparison to the Bipolar technology that provides low threshold voltage.
- CMOS technology provides high noise margin, packing density whereas Bipolory technology allows to have low noise margin so that to reduce the high volues and give low packing density of the components.

Question 29. What Are The Different Classification Of The Timing Control?

Answer :

There are different classification in which the timing control data is divided and they are:

- Delay based timing control: this is based on timing control that allows to manage the component such that the delay can be notified and wherever it is required it can be given. The delays that are based on this are as:

  - Regular delay control: that controls the delay on the regular basis.
  - Intra-assignment delay control: that controls the internal delays.
  - Zero delay control

- Events based timing control: this is based on the events that are performed when an event happens or a trigger is set on an event that takes place. It includes

  - Regular event control
  - Named event control
  - Event OR control

- Level sensitive timing control: this is based on the levels that are given like 0 level or 1 level that is being given or shown and the data is being modified according the levels that are being set. When a level changes the timing control also changes.

Question 30. We Have Multiple Instances In Rtl(register Transfer Language), Do You Do Anything Special During Synthesis Stage?

Answer :

While writing RTL(Register Transfer language),say in verilog or in VHDL language, we dont write the same module functionality again and again, we use a concept called as instantiation, where in as per the language, the instanciation of a module will behave like the parent module in terms of functionality, where during synthesis stage we need the full code so that the synthesis tool can study the logic , structure and map it to the library cells, so we use a command in synthesis , called as "UNIQUIFY" which will replace the instantiations with the real logic, because once we are in a synthesis stages we have to visualize as real cells and no more modelling just for functionality alone, we need to visualize in-terms of physical world as well.

Question 31. What Is Tie-high And Tie-low Cells And Where It Is Used?

Answer :

Tie-high and Tie-Low cells are used to connect the gate of the transistor to either power or ground. In deep sub micron processes, if the gate is connected to power/ground the transistor might be turned on/off due to power or ground bounce. The suggestion from foundry is to use tie cells for this purpose. These cells are part of standard-cell library. The cells which require Vdd, comes and connect to Tie high...(so tie high is a power supply cell)...while the cells which wants Vss connects itself to Tie-low.

Question 32. What Is The Difference Between Latches And Flip-flops Based Designs?

Answer :

Latches are level-sensitive and flip-flops are edge sensitive. latch based design and flop based design is that latch allowes time borrowing which a tradition flop does not. That makes latch based design more efficient. But at the same time, latch based design is more complicated and has more issues in min timing (races). Its STA with time borrowing in deep pipelining can be quite complex.

Question 33. What Is Local-skew, Global-skew,useful-skew Mean?

Answer :

Local skew : The difference between the clock reaching at the launching flop vs the clock reaching the destination flip-flop of a timing-path.

Global skew : The difference between the earliest reaching flip-flop and latest reaching flip-flop for a same clock-domain.

Useful skew: Useful skew is a concept of delaying the capturing flip-flop clock path, this approach helps in meeting setup requirement with in the launch and capture timing path. But the hold-requirement has to be met for the design.

SystemVerilog Interview Questionsa

Summarized by Chong Yao

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.

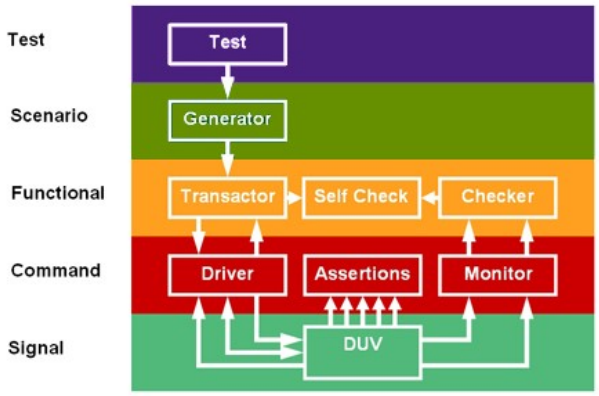
cover : This statement monitors property for the sake of coverage. Coverage can be made to be reported at the end of simulation.

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?

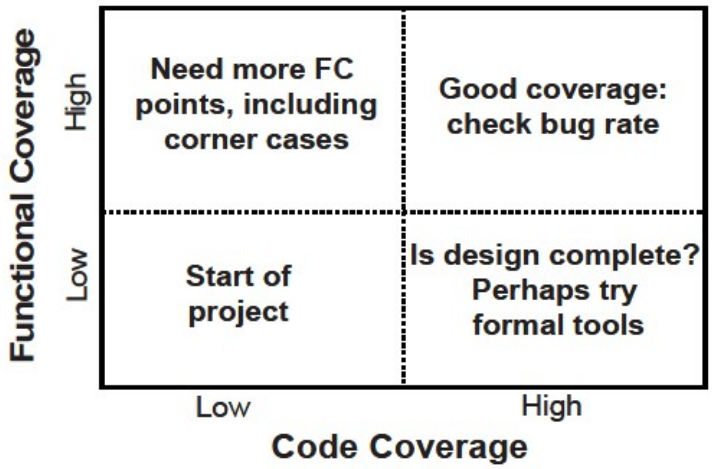
Ans: A callback is a built in systemverilog task/function. Suppose if we are transmitting a data using mailbox and you are sending data from design to transmitter. If you want to get back the data and you need the same data to put back in the scoreboard for comparison, this is called callback. Any change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

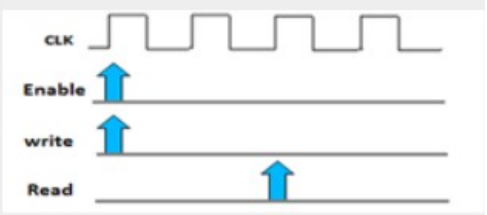
112. What is difference between x [->4:7] and x [=4:7] in SVA?

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

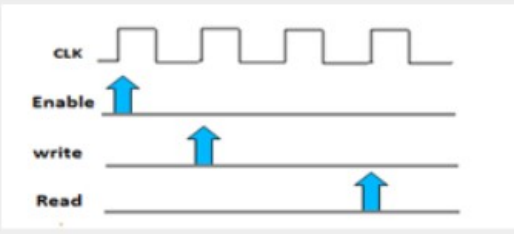
(1. Overlapping ( )

```
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

...

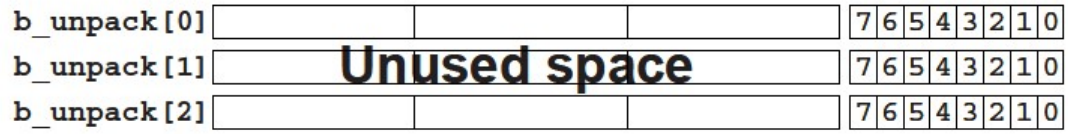
endprogram

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:

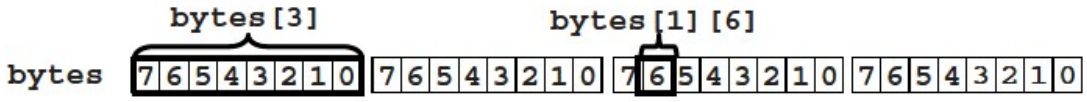
For some data types, you may want both to access the entire value and also to divide it into smaller elements. A SystemVerilog packed array is treated as both an array and a single value. It is stored as a contiguous set of bits with no unused space, unlike an unpacked array.

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:

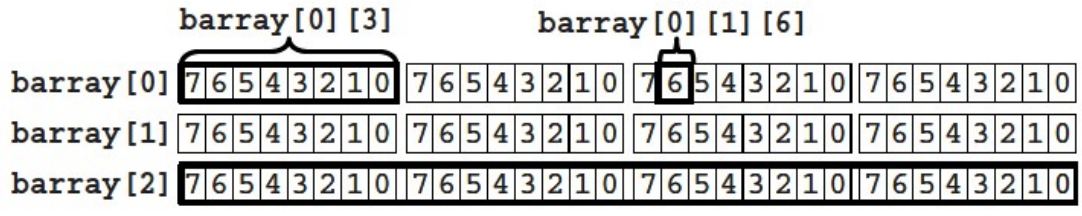
bit [3:0] [7:0] barray [3]; // Packed: 3x32-bit

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

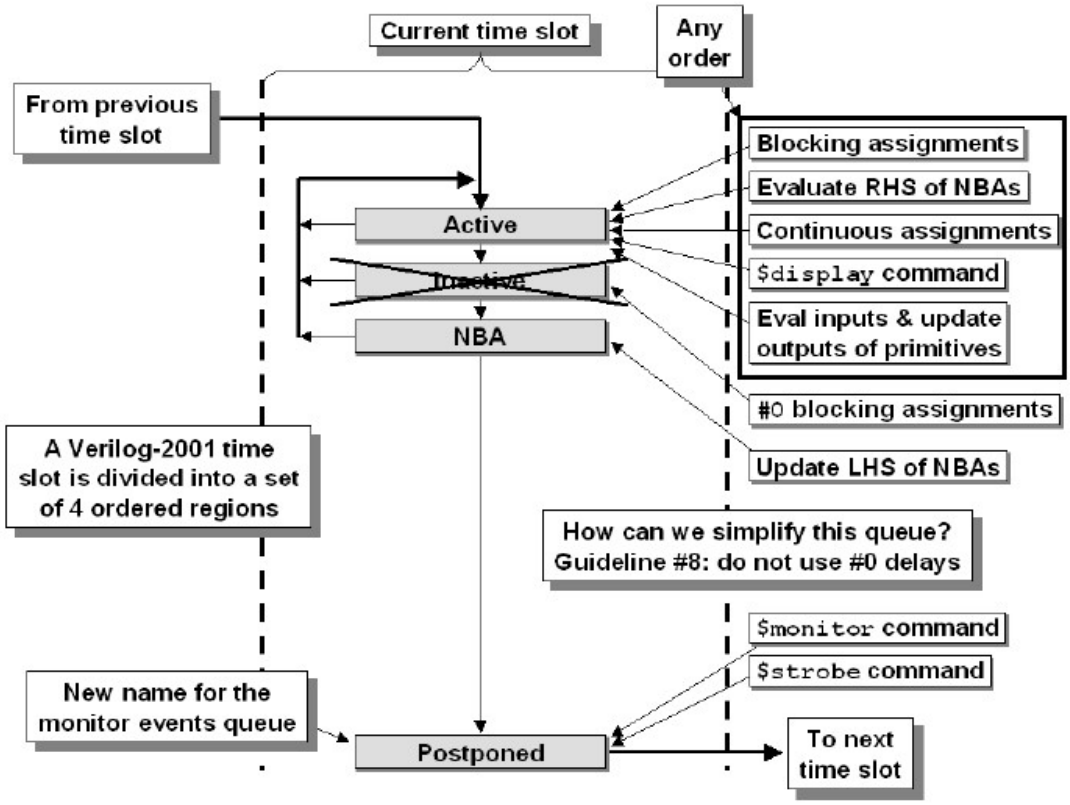
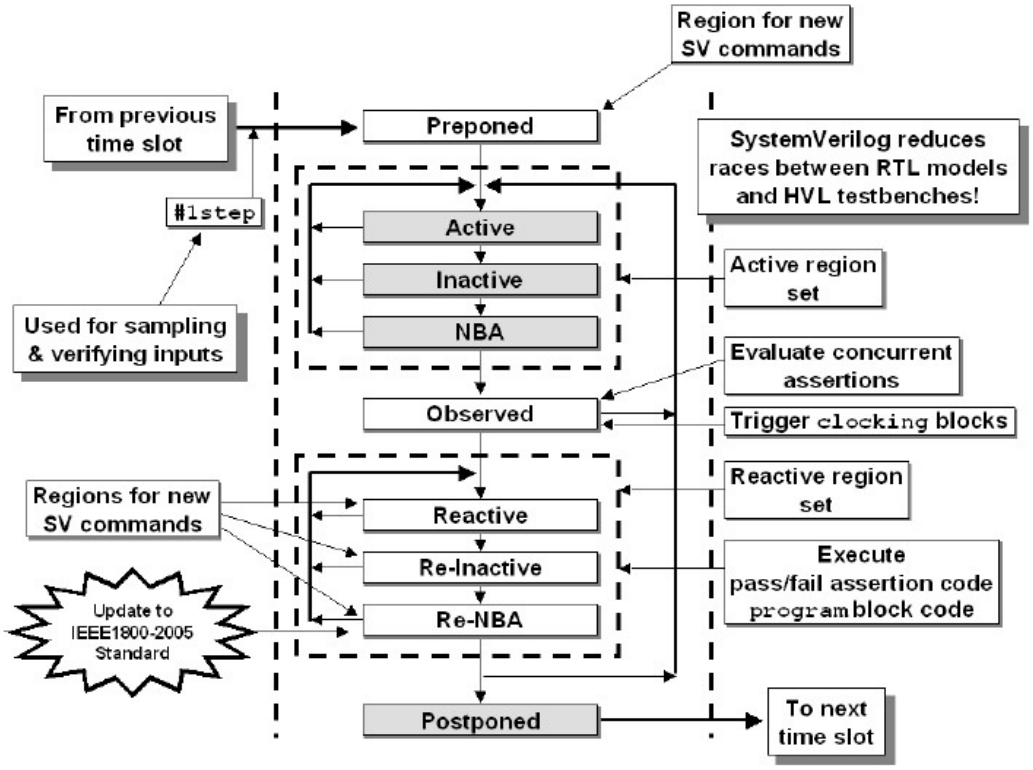
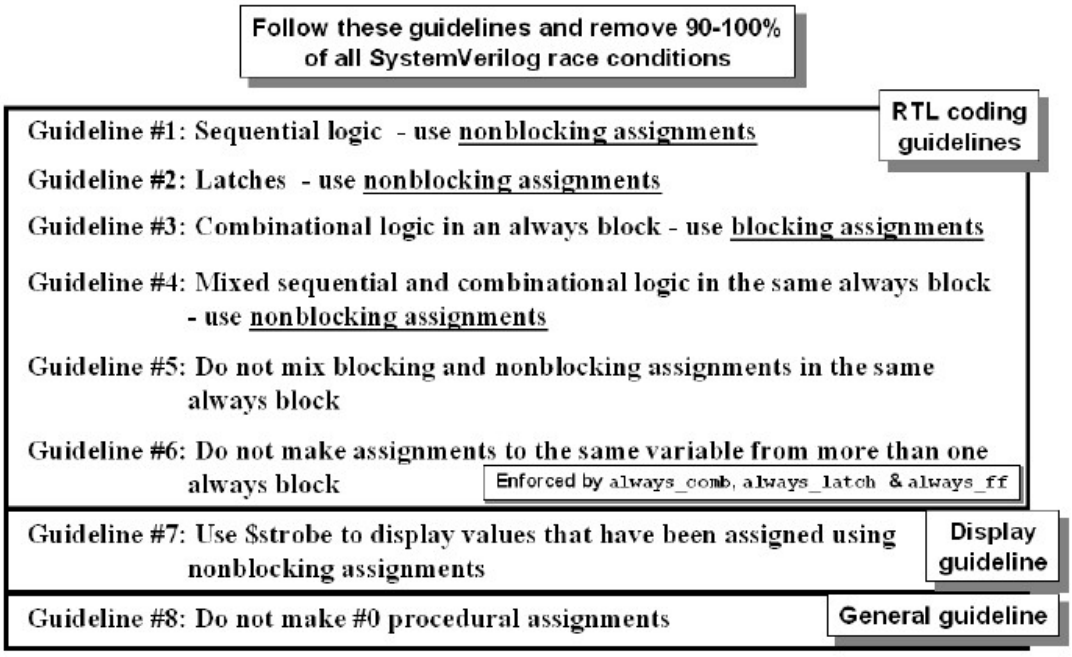


Figure 1 - Verilog-2001 event regions





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	Both continuous assignment or blocking/non

		assignment	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.

Reference

1. http://www.asic-world.com/
2. http://www.testbench.in/
3. SystemVerilog for Verification: A Guide to Learning the Testbench Language Features by Chris Spear