INTERVIEW QUESTIONS

VERILOG

JAIRAJ MIRASHI
DESIGN VERIFICATION ENGINEER

Verilog Q&A

Verilog

1) Write a verilog code to swap contents of two registers with and without a temporary register?
With temp register (using blocking assignment)

Answer:

With temp register (using blocking assignment)

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

Without temp register (using non-blocking assignment)

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

2) Difference between blocking and non-blocking?

Answer:
In Verilog , we have two forms of the procedural assignment statement:

1. blocking assignment (represented by "=" operator)
2. non-blocking assignment (represented by "<=" operator).

In blocking assignment statement (i.e. "=" operator), whole statement executes in Active region of current time stamp and then control goes to the next statement.
Whereas,
In non-blocking assignment statement (i.e. "<=" operator), right-hand sides gets evaluated first in Active region and assigns it to left-hand side in the NBA region of the current time stamp.

Example code is mentioned below

```
module blocking;
reg [0:7] A, B;
initial begin: init1
A = 3;
#1 A = A + 1; // blocking procedural assignment
B = A + 1;

$display("Blocking: A= %b B= %b", A, B );
A = 3;
#1 A <= A + 1; // non-blocking procedural assignment
B <= A + 1;
#1 $display("Non-blocking: A= %b B= %b", A, B );
end
endmodule
```

Output

```
Blocking:     A= 00000100 B= 00000101
Non-blocking: A= 00000100 B= 00000100
```

3) 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.

4) Difference between inter statement and intra statement delay?

Answer:
Inter statement delay: It is the delay where the instruction execution is performed after the specified number of time steps. In the expression the values of b and c are sampled after the specified amount of delay and perform the addition and assign the result to the LHS immediately.
#10 a = b+c ;
Wait for 10 time units, sample the b and c values, perform addition, assign to ‘a’ immediately.

Intra statement delay: Here the result of the evaluated expression is assigned to LHS after the specified number of time units.
a = #10 b+c ;
First sample b and c values, perform addition, wait for 10 time units, then assign the result to ‘a’

5) What is delta simulation time in HDL Design Simulation?

Answer:
In an event driven logic simulator, the concept of physical time is abstracted away. The simulator only cares about changes on signals. Each signal change fans out to logic that causes other signals to change. In fact in a purely synchronous RTL description, you could simulate your design with out using any time, just clock cycles. However, it is much more convenient to associate a physical time period to the clock so you can display wave-forms and diagnostic reports are more human readable.

Assuming again that your RTL has no delays in its description and the only delays in your testbench are in the generation of your clock, the simulation executes exactly the same regardless of what the clock period is. All of the activity in your design, both sequential and combinational logic, evaluates on the clock edge, and time is just a bookkeeping chore between clock edges.

Delta cycles come into play within that instantaneous moment when everything gets evaluated. Different languages may have different definitions of what constitutes a delta cycle, but it is basically a simulation time step without advancing physical time. When ever a signal changes, that signal output fans out to the input of other logic that may cause other signals to change. This happens over and over again and each iteration could be considered a delta cycle. The iteration stops when there are no more signals changing and then the simulator is ready to advance time.

The significance of this come into play when you look at the simulation of a shift register. In a perfect situation, all of the registers see a clock edge in the same delta cycle and change the Q output of each register to the value of the D input. But the D input is the Q output of another register. How do you guarantee that you are getting the old value of the Q and not the new value? By placing the assignment to Q of each register in the next delta cycle.

Verilog, VHDL, and SystemC all have different ways of dealing with this, so you need to become a little familiar with the scheduling semantics of each language that you plan on using.

6) What is the difference between display, monitor and strobe in verilog?

Answer:
All the three statements mentioned in the question are used to monitor values of variables. They also have similar syntaxes. They only differ in the way they are used and the time they get executed.

● **$display:** The $display statement is used to display the immediate values of variables or signals. It gets executed in the active region.

●**$monitor:** The $monitor statement displays the value of a variable or a signal when ever its value changes. It gets executed in the postponed region. To monitor the value of a variable throughout the simulation, we would have to write the $monitor statement only once in our code. Since the $monitor statement gets executed in the postponed region, updates to the signal done in the inactive, NBA, re-active, re-inactive and re-NBA regions will also be displayed in addition to that of active region, depending on which region updated the variable last.

●**$strobe:** $strobe signal displays the value of a variable or a signal at the end of the current time step i.e the postponed region.

Unlike $monitor, $strobe and $display need to be added as many times as the value of the signal is to be displayed.

7) What is difference between Verilog full case and parallel case?

Answer:
A case statement in Verilog is said to be a full case when it specifies the output for every value of the input. In the context of synthesizable code, this means specifying the output for all combinations of zeros and ones in the input. Case items involving X’s and Z’s are ignored by synthesis tools.

The significance of full case statements is that case statements that are not full infer latches unless a synthesis pragma - full_case is used. A latch is inferred because it matches the behavior in simulation which is to remember the previous output when input changes to an unspecified value. In most cases, such latches are unintended and undesirable.

A case statement in Verilog is said to be a parallel case when it isn't possible for multiple case items to be equal in value to the case-expression.

The significance of parallel case statements is that case statements that are not parallel infer priority logic during synthesis. This is done to match the behavior in simulation which

is to assign the output according to the first matching case-item only (even when there are multiple case-item matches).

8) What makes an inferred latch? Why are inferred latches bad? how to avoid inferred latches>

Answer:
What makes an inferred latch?
For combinatorial logic, the output of the circuit is a function of input only and should not contain any memory or internal state (latch).
In Verilog, a variable will keep its previous value if it is not assigned a value in an always block. A latch must be created to store this present value.
An incomplete if-else statement will generate latches. An if-else statement is considered "incomplete" if the output state is not defined for all possible input conditions. The same goes for an incomplete case statement, or a case statement that does not have a default: item.

Why are inferred latches bad?
Inferred latches can serve as a 'warning sign' that the logic design might not be implemented as intended. A crucial if-else or case statement might be missing from the design.
Latches can lead to timing issues and race conditions. They may lead to combinatorial feedback - routing of the output back to the input - which can be unpredictable.

To avoid creating inferred latches:
Include all the branches of an if or case statement
Assign a value to every output signal in every branch
Use default assignments at the start of the procedure, so every signal will be assigned.

9) 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:
1. Evaluate the RHS of nonblocking statements at the beginning of the time step.
2. Update the LHS of nonblocking statements at the end of the time step.

10) 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.

11) What is the difference between synthesis and simulation in verilog?

Answer:
Simulation is the process of using a simulation software (simulator) to verify the functional correctness of a digital design that is modeled using a HDL (hardware description language) like Verilog.

Synthesis is a process in which a design behavior that is modeled using a HDL is translated into an implementation consisting of logic gates. This is done by a synthesis tool which is another software program.

One main difference in terms of modelling using a language like Verilog is that for synthesis , the design behavior should be modeled at an RTL (Register Transfer Level) abstraction. This means modelling in terms of the flow of digital signals between hardware registers (flip-flops) and the logical operations (combinational logic) performed on those signals.

Hence if a Verilog design model is intended for synthesis, then only synthesizable constructs should be used, while for simulation there are no such restrictions.

Some of the non-synthesizable constructs in Verilog are tasks, delays, events, fork..join, initial..begin blocks , force and release statements. These should be avoided if a code is intended for synthesis.

Another main difference is sensitivity list used in always blocks. These are important for a simulator to decide on how to evaluate the logic , but for synthesis it is a don’t care.

12) What are the timing regions in Verilog?

Answer:
In verilog, the timing/clocked/event regions are:

Preponed region: In this region, concurrent assertions are sampled. In each time slot, the preponed region is executed only once.

Active region: This region is responsible for evaluation of blocking assignments, RHS of NBA’s, continuous assignments, evaluation of inputs and updates the output of primitives, execution of $display & $finish command, execution of module block.

Inactive region: In this region, #0 blocking assignments are scheduled.

NBA (Non-blocking assignment) region: In this region, left hand side of the NBA’s is updated.

Postponed region: This region executes $strobe & $monitor command and gives the final updated values for the current time slot and it is also used to collect the functional coverage.

Note: Observed, Re-Active and Re-Inactive regions are the parts of system verilog timing regions including above five verilog regions.

13) What is the exact and main Difference between always and forever in Verilog?

Answer:
The primary mechanism in a behavioral level modeling are structured procedures, also known as concurrent blocks,

They can be expressed in two types of blocks,
initial - they execute only once
always - they execute for ever (until simulation finishes)

The forever is a loop that does not contain any expression and executes forever until the $finish task is encountered.

- A forever loop is typically used in conjunction with timing control constructs.
- A forever loop can be also exited by use of the disable statement.

The very important facts are:
- You can’t nest initial and always in same block.
- Forever can be used inside initial block.
- Forever is not synthesizeable to hardware means, but always can be synthesized to hardware structure.

14) What is an event queue in Verilog?

Answer:
An event queue is what a simulator uses to schedule concurrent behavior. As a Hardware Description Language (HDL), Verilog is meant to describe massive amount of coexisting processes that are usually synchronized to some clock or set of signals. A “event” is a change in signal value, or the passing of time. A rising edge (posedge) of a clock can be consider an event. And when that event happens, there can be thousands or millions of other processes waiting for that to happen.

When your simulator is just software running as a program on a computer, there is no way to give each process its own thread running in parallel. So all the processes get put into an event queue and each process executes sequentially. There can be many queues waiting for different events to happen, but there is only one active event queue.

The active event queue is the the set of processes that need to execute at the current time. Executing those processes can add more events to the active or other event

queues. Once the active queue is drained of all processes to execute, the simulator takes the event queue waiting for the smallest amount of time to pass, advances time to that, and makes that queue the active queue. Most HDL simulations are implemented as discrete event-driven simulators because of these semantics.

15) 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.

16) Difference between Verilog and vhdl?

Answer:
Verilog has its origins in gate and transistor level simulation for digital electronics (logic circuits), and had various behavioral extensions added for verification. VHDL was the winner in a DoD competition to develop an HDL for the VHSIC program and is based on ADA programming language.

So Verilog is good at hardware modeling but lacks higher level (programming) constructs. VHDL has a lot of programming constructs but lacks the low level modeling capabilities for accurately representing hardware.

VHDL is popular with (European) FPGA designers because low-level modeling is not required in an FPGA flow. ASIC designers prefer Verilog.

SystemVerilog adds some higher level constructs to Verilog for verification but doesn't extend the hardware modeling capabilities. Verilog-AMS adds analog capabilities - but is a different standard (Accellera vs IEEE). VHDL had analog capabilities added, but they are less well integrated than in Verilog-AMS (which does automatic type conversion on A/D boundaries).

17)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

18) How to write FSM is verilog?

Answer:
there r mainly 4 ways 2 write fsm code
1) using 1 process where all input decoder, present state, and output decoder r combine in one process.
2) using 2 process where all comb ckt and sequential ckt separated in different process
3) using 2 process where input decoder and persent state r combine and output decoder seperated in other process
4) using 3 process where all three, input decoder, present state and output decoder r separated in 3 process.

Answer:
#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.

20) 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.

21) 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.

22) What is the difference between wire and reg?

Answer:
Wire
Wires are used for connecting different elements. They can be treated as physical wires. They can be read or assigned. No values get stored in them. They need to be driven by either continuous assign statement or from a port of a module.

Reg
Contrary to their name, regs don't necessarily correspond to physical registers. They represent data storage elements in Verilog/SystemVerilog. They retain their value till next value is assigned to them (not through assign statement). They can be synthesized to FF, latch or combinatorial circuit. (They might not be synthesizable !!!)

Wires and Regs are present from Verilog timeframe. SystemVerilog added a new data type called logic to them. So the next question is what is this logic data type and how it is different from our good old wire/reg.

Logic
As we have seen, reg data type is bit mis-leading in Verilog. SystemVerilog's logic data type addition is to remove the above confusion. The idea behind is having a new data type called logic which at least doesn't give an impression that it is hardware synthesizable. Logic data type doesn't permit multiple drivers. It has a last assignment wins behavior in case of multiple assignments (which implies it has no hardware equivalence). Reg/Wire data types give X if multiple drivers try to drive them with different values. Logic data type simply assigns the last assignment value. The next difference between reg/wire and logic is that logic can be both driven by assign block, output of a port and inside a procedural block like this

```
logic a;
    assign a = b ^ c; // wire style
    always (c or d) a = c + d; // reg style
    MyModule module(.out(a), .in(xyz)); // wire style
```

23) What are the rules governing usage of a Verilog function?

Answer:
The following rules govern the usage of a Verilog function construct:

A function cannot advance simulation-time, using constructs like #, @. etc.
A function shall not have nonblocking assignments.
A function without a range defaults to a one bit reg for the return value.
It is illegal to declare another object with the same name as the function in the scope where the function is declared.

24) Can there be full or partial no-connects to a multi-bit port of a module during its instantiation?

Answer:
No. There cannot be full or partial no-connects to a multi-bit port of a module during instantiation

25) What happens to the logic after synthesis, that is driving an unconnected output port that is left open (, that is, noconnect) during its module instantiation?

Answer:
An unconnected output port in simulation will drive a value, but this value does not propagate to any other logic. In synthesis, the cone of any combinatorial logic that drives the unconnected output will get optimized away during boundary optimisation, that is, optimization by synthesis tools across hierarchical boundaries.

26) How is the connectivity established in Verilog when connecting wires of different widths?

Answer:
When connecting wires or ports of different widths, the connections are right-justified, that is, the rightmost bit on the RHS gets connected to the rightmost bit of the LHS and so on, until the MSB of either of the net is reached.

27) What is the implication of a combinatorial feedback loops in design testability?

Answer:
The presence of feedback loops should be avoided at any stage of the design, by periodically checking for it, using the lint or synthesis tools. The presence of the feedback loop causes races and hazards in the design, and 104 RTL Design leads to unpredictable logic behavior. Since the loops are delay-dependent, they cannot be tested with any ATPG algorithm. Hence, combinatorial loops should be avoided in the logic.

28) What are the various methods to contain power during RTL coding?

Answer:
Any switching activity in a CMOS circuit creates a momentary current flow from VDD to GND during logic transition, when both N and P type transistors are ON, and, hence, increases power consumption.
The most common storage element in the designs being the synchronous FF, its output can change whenever its data input toggles, and the clock triggers. Hence, if these two

elements can be asserted in a controlled fashion, so that the data is presented to the D input of the FF only when required, and the clock is also triggered only when required, then it will reduce the switching activity, and, automatically the power.

The following bullets summarize a few mechanisms to reduce the power consumption:
Reduce switching of the data input to the Flip-Flops.
Reduce the clock switching of the Flip-Flops.
Have area reduction techniques within the chip, since the number of gates/Flip-Flops that toggle can be reduced.

29) 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
```

30) Variable And Signal Which Will Be Updated First?

Answer:
Signal

31) 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.

32) 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.

33) 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

34) 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.