Interview Questions in Verilog

1. What is the difference between wire and reg?

Wire	Reg
Assumes Value	Holds Value
wire needs drivers to get output values	reg does not need driver
wire elements can only be used to model combinational logic	reg can be used for combinational as well as sequential logic
wire can be used as the left hand side of an assign statement	reg cannot be used on the left-hand side of an assign statement

2. What do we mean by continuous assignment ?

Continuous assignment is used to drive values to net. Left hand side can be scalar or vector net or concatenation of both while right hand side can be scalar or vector net or register or concatenation of both. Read more here .

3. What do we mean by full case and parallel case?

A full case is a case statement in which all possible all possible case expression can be matched with case items or case default. If it is possible to find binary expression that doesn't case item the case statement is not full. A parallel case statement is a case statement in which it is possible to match a case expression with 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 is called overlapping or non parallel.

4. What is the difference between $monitor and $display.

$monitor and $display both system functions and are both used to see the testbench results.

$monitor	$display
Monitors change in the value of signal. Signal can be variable, strings or expression.	Displays the value of signal.
Can be invoked once	Can be invoked more than once.

5. What is $strobe?

It is a synchronization mechanism in which data is displayed only after all other statements are executed unlike $display in which execution order could be nondeterministic.

6. What is meant by inferring latches and how to avoid it?

Inferring latch means to reproduce last value when unknown branch is specified. For example, to avoid latches make sure that all cases are mentioned in case statements if not case default is mentioned. In the same way latch is inferred in IF statement if ELSE IF is not specified.

```
always @(s1 or s0 or i0 or i1 or i2 or i3)
case ({s1,s0})
2'b00 : out = i0;
2'b01 : out = i1;
2'b10 : out = i2;
endcase
```

In above example code of 4to1 multiplexer all combinations are not compared and default is also not used so latch is inferred to reproduce previous value.

7. What is the difference between == and === operator?

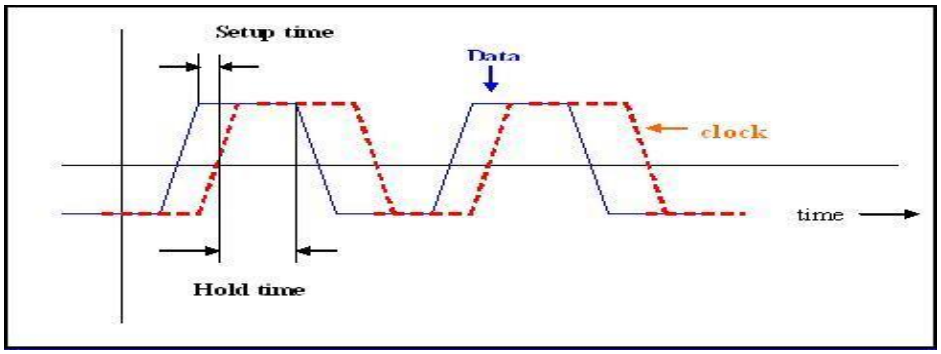
Output of == can be 0,1,X while output of === is always 0 or1.

== doesn't compare X and if at least one bit is X output will be X.=== compare X bits and output is always 0 or 1 accordingly.

8. What is the difference between $setup and $hold?

$setup and $hold is used to monitor setup and hold time constraints for sequential logic. Setup time is the minimum time the data must arrive before active edge of clock signal and hold time is the minimum time the data cannot change after active edge of clock signal. They are specified under specify block.

Draw following diagram to explain.



9. What is casex and casez statement?

These are the types of case statement in which casex represent x and z as don't care while casez represent z as don't care. Don't cares are not allowed in case statement so casex and casez are used.

10. What is repeat loop in Verilog?

Repeat loop is used to execute loop fixed number of times. It is not used to loop expression like we see in while loop statement. It contains constant, variable or signal . For example repeat(8) .

11. What is duty cycle?

Duty Cycle is the fraction of time the signal is high or low. It represents on time of a signal. It can be represented by D=T/P where Dis the duty cycle,T is the time signal is active and P is the total time period of signal.

12. What is $random?

$random is used to generate random numbers to identify hidden bugs in program. It returns 32 bit number. It returns a new random number each time the function is called. It can also contain as argument which ensures that same random number is generated each time the test is run. The argument for can be register, integer or time variable.

$random;
$random();

13. Write a Verilog code for synchronous and asynchronous reset?

Example:- a.
always @(poseedge clk) //synchronous reset is clock dependent
begin
If (reset)
else
end

b.always @(posedge clock or posedge reset) //Asynchronous reset is clock independent
begin
if(reset)
end

14. What is the difference between blocking and nonblocking assignment?

blocking	Non blocking
The whole statement is done before Control passes to next statement	It evaluates all the RHS at current time unit and assign to LHS at the end of the time unit.
It uses = operator	It uses <= operator

15. What is the difference between task and function?

Function	Task
Function can invoke another function but not task	Task can invoke another task or function
Function doesn't contain timing constraints	Task can contain timing constraints<= operator
Function must have at least one input argument	Task can contain input, output, inout as argument
Function always return single value	Task does not return value but pass multiple values through output or inout ports.

16. What are different types of delay control?

a) Regular delay control
b) Intra assignment delay control
c) Zero delay control

Example:-

```
//Intra assignment delay control
reg a,b,c;
Initial
begin
a=0,b=0;
c= #5 a+b;
/* Takes value of a and b at time 0 evaluates a+b and then
 waits 5 time units to assign value to c */
end

//Regular delay control
a=0,b=0;
Initial
begin
temp=a+b;
#5 c=temp
/*Evaluates a+b at current time unit and stores it in temprory
variable and then assign it to c at 5 time units */
end

//Zero delay control
initial
begin
a=0;
b=0;
end
Intial
begin
   #0 a=1;  //Zero delay control
   #0 b=1;
end
/*
In above code a=0,b=0,a=1.b=1 are to be excuted at
simulation time zero but a=1 and b=1 are excuted after
a=0,b=0 as zero delay control is applied to these statements.
*/
```

17. What do you mean parallel block?

In parallel block all statements are executed concurrently(ie not sequential) They are specified by keyword fork and join. Timing constraints can be provided in parallel block. Example:-

```
module parallel;
reg a,b,c;
initial
begin
$monitor ("%g,a=%b,b=%b,c=%b",$time,a,b,c);
fork
#1 a=0;
#5 b=1;
#10 c=0;
join
 #1 $display ("%g EXIT",$time);
end
endmodule
```

In the above code a gets 0 after 1 time unit,b after 5 time unit,c after 10 time units ,EXIT after 11 time units

18. What is $time in Verilog?

$time function is invoked to get the current simulation time.A time variable is a special register data type to store simulation time.$time returns 64bit integer value. Example:-

```
 time curr_time;
      initial
      curr_time = $time;
```

19. What is defparam?

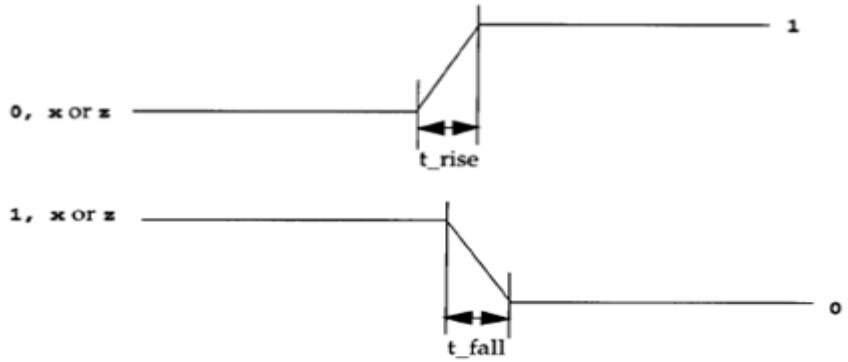
Parameter values can be overridden by use of defparam keyword at module instance.It can also change the parameter values at different time interval within the module. Example:-

```
module hello;
   parameter p1 = 0;
   initial
   $display ("hello p1 = %d",p1);
   endmodule

   module Top;
   defparam c1.p1 = 2,c2.p1 = 3; //Parameter values are overridden in module Top
    hello c1();//module instance of hello
    hello c2();
    endmodule
```

20. What is rise,fall and turnoff delays?

a. Rise delay indicates gate output transition from any value to 1
b. Fall delay indicates gate output transition from any value to 0
c. Turnoff delay indicates gate output transition from any value to z



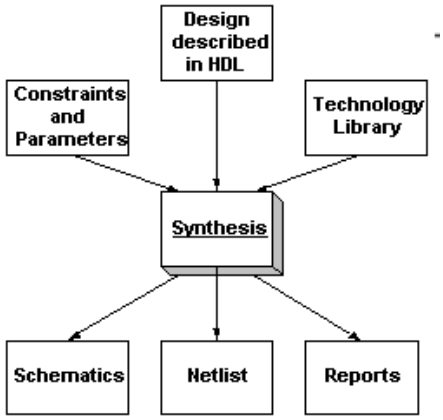
Each delay mentioned above has min\typ\max values. These values are used because of the variation in IC fabrication process.

Syntax:-

#(rise fall turnoff)

21. What do you mean by logic synthesis?

Logic synthesis is mechanism by which RTL description is converted in terms of logic gates by the use of synthesis tool. It is recommended that signal width and variable width is explicitly specified. Defining unsized variable results in large gate level netlist.



22. Show delay back annotation flow?

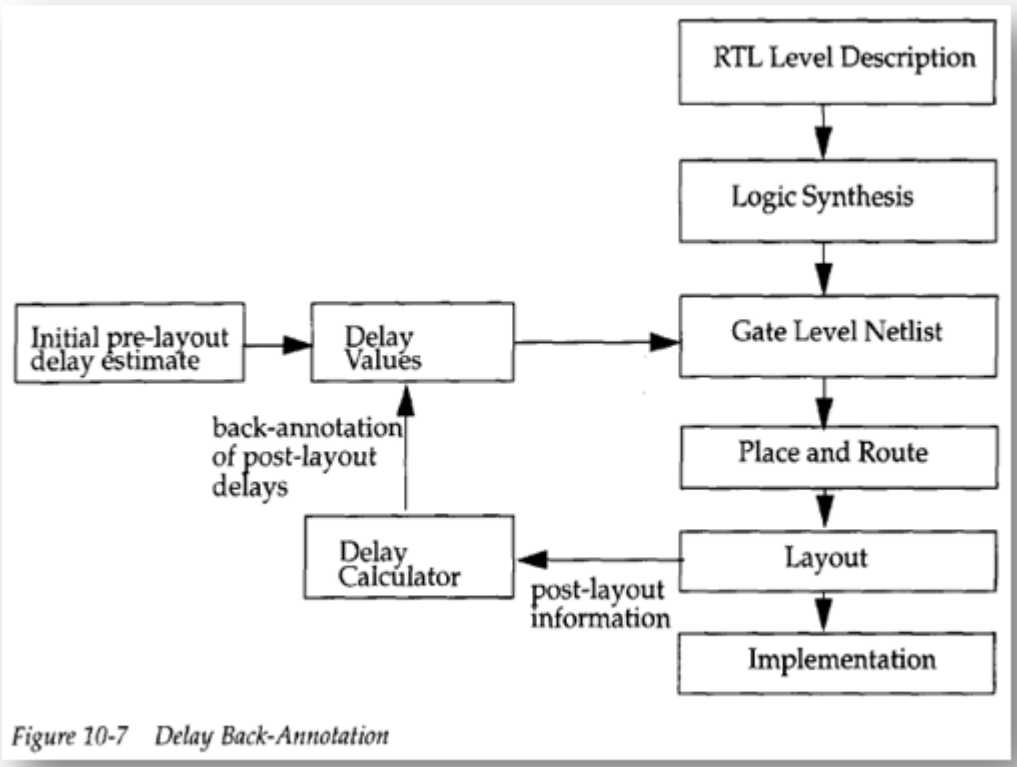


Figure 10-7 Delay Back-Annotation

The above process is used repeatedly to obtain final circuit that meets timing requirements.

23. What is specparam?
Specparam provides mechanism to specify parameters inside specify block.This is different from module parameters which is not specified in specify block.

The following example illustrates specparam declarations:

```
specify
specparam tRise_clk_q=150, tFall_clk_q=200;
specparam tRise_control=40, tFall_control=50;
endspecify
```

24. What is PLI?
PLI is programmining language interface.Essentialy it's a mechanism to invoke C function inside Verilog. It extends Verilog capabilities by allowing users to define their own utilities.Designers can write their own syatem tasks by using PLI routine.Some PLI application include

a. Application software like calculator or translator can be written using PLI.
b. Can be used to define customized system tasks and routines.
c. PLI can be used for customized output display
d. It allows users to change internal data structure.

25. What is force and release?
These are the second form of procrdural continuous assignment.They are used to force and release certain values to registers or nets.They are not used in design block and only used in stimulus block or testbench.They are active for certain period only.
The below example shows the use of force and release

```
module testbench;
//statements
Dff tdff(q,q_bar,d,clk,rst);
initial
begin
#50 force tdff.Q = 1'b1; //Force Q to 1 at time 50
#50 release tdff.Q; //relaese Q at time 100
end
//statements
endmodule
```

26. What is level sensitive timing control in Verilog?
We know that @ is used for edge sensitive timing control but Verilog also provides level sensitive timing control ie processor waits for certain expression or condition to be true to excute the statement.This is provided by wait statement.

Syntax:-

wait (expression) statement

Here expression is Boolean value ie true or false.When true statement is excuted.When false it waits for expression to become true.

27. What are types of compiler directives?
There are four types of compiler directives in Verilog

a. `define
b. `include
c. `ifdef
d. `timescale

`define is used to define text macros

For example:-

`define SIZE 16

`include is used to import entire Verilog file to another file.

For example if want to include moniter.v file into design.v

//design.v file

`include moniter.v statements

`ifdef directive checks if the macro has been defined or not if defined

It compliles the code that follows otherwise compiler compiles the code following an optional `else directive

`ifdef SIZE

`timescale directives specify time unit and time precision of module that follows it.

Syntax:

`timescale unit\precision

28. What is logic synthesis?

Logic synthesis is the mechanism to convert RTL description in terms of gate level representation.The main concern in logic synthesis are fanin,fanout etc.It can be technology dependent or technology independent.Logic synthesis tools are present for this purpose.

29. Write a Verilog code to swap contents of two register with and without temporary registers.

```
//with temporary register

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

//without temporary register
```

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

Verilog Quiz

30. Operator which precedes the operand

A . Unary **=> This is the correct answer .**
B . Binary
C . Ternary
D. None

31. Which is legal negative number

A . 4'd-3
B . 6'-d3
C . -6d'3 **=> This is the correct answer .**
D. None

32. What is the default value for reg data type

A . 0
B . 1
C . z
D. x **=> This is the correct answer .**

33. What is system task to suspend simulation

A . $finish
B . $minitor
C . $display
D. $stop **=> This is the correct answer .**

34. **Turn off delay means, gate output transition to**

A . 1
B . 0
C . z **=> This is the correct answer .**
D. x

35. **Parameter value can be overridden at module instance by**

A . Specparam
B . Defparam **=> This is the correct answer .**
C . Parameter
D. None

36. **In continuous assignment left hand side must be**

A . Net
B . Reg
C . Scalar or Vector Net **=> This is the correct answer .**
D. Scalar or Vector reg

37. **If in1 = 4'b101x and in2 = 4'b0101 then in1 + in2 equals**

A . 0111
B . 0110
C . x
D. None **=> This is the correct answer .**

38. **-10 % 3 evaluates to**

A . -1 **=> This is the correct answer .**
B . 1
C . 0
D. x

39. What are the possible values of == operator

A . 0,1
B . 0,x
C . 1,x
D. 0,1,x **=> This is the correct answer .**

40. What is the time period of clock #10 clock = ~clock

A . 10
B . 20 **=> This is the correct answer .**
C . 5
D. None

41. #40 $finish indicates

A . end of simulation time
B . end of simulation at 40 time units **=> This is the correct answer .**
C . suspend simulation at 40 time units
D. None

42. @posedge means

A . Transition from 0 to 1,x or z **=> This is the correct answer .**
B . Transition from x to 1
C . Transition from z to 1,x
D. All of Above

43. What is the width of time register

A . 16 bit
B . 32 bit
C . 64 bit
D. 1268 bit **=> This is the correct answer .**

44. **%g or %G displays**

A . Real numbers in decimal or scientific notation
B . Real numbers in decimal or scientific notation whichever shorter
C . Real numbers in scientific format
D. Real numbers in decimal format **=> This is the correct answer .**

45. **Parameter value can be overridden at module instance by**

A . `define **=> This is the correct answer .**
B . `include
C . `ifdef
D. `timescale

46. **If x=4'b1100 then x<<2 is**

A . 4'b1000
B . 4'b0000 **=> This is the correct answer .**
C . 4'b0011
D. 4'b0110

47. **If x=4'1100 then &x equals**

A . 1'b0
B . 1'b1
C . 4'b1100
D. None **=> This is the correct answer .**

48. **If A=4'b1010 and B=4'b1100 then A&B**

A . 4'b0000
B . 4'b1000 **=> This is the correct answer .**
C . 1'b1
D. 1'b0

49. **If A=1’b1,B=2’b01,C=2’b00 then y= {A,B[0],C[1]} equals**

A . 3'b110 **=> This is the correct answer .**
B . 3'b100
C . 3'b101
D. 1'b0

50. **If A=1’b1,B=2’b01,C=2’b00 y={4{A},2{B},C} equals**

A . 10’b1111010100 **=> This is the correct answer .**
B . 9’b111101010
C . 8’b11110100
D. None

51. **Case x treats**

A . x as don’t care
B . z
C . as don’t care
D. Both x and z as don’t care **=> This is the correct answer .**

52. **If two signals of strong1 and weak0 contend which prevails**

A . tri **=> This is the correct answer .**
B . triand
C . trior
D. trireg

53. **Which net has storage strength**

A . edge sensitive bit
B . level sensitive
C . both
D. none **=> This is the correct answer .**

54. wait statement is

A . edge sensitive bit
B . level sensitive **=> This is the correct answer .**
C . both
D. none

55. In continuous assignment LHS can be

A . Scalar net
B . Vector net
C . Concatenation of both **=> This is the correct answer .**
D. All of above

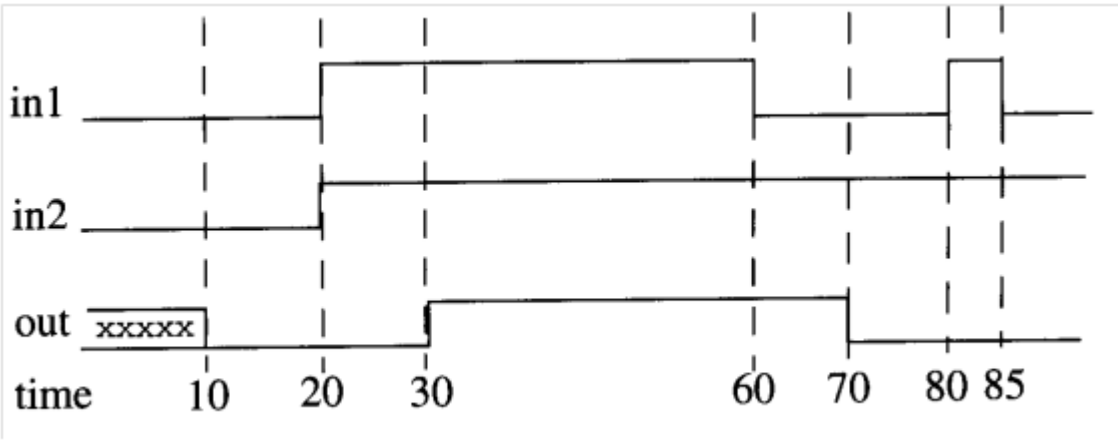
56. <= is used in

A . Blocking
B . Non Blocking **=> This is the correct answer .**
C . Both
D. None

57. Asynchronous reset is

A . Clock dependent
B . Clock independent **=> This is the correct answer .**
C . Either
D. None

58. **29. What could be the expression for below figure where in1 and in2 are input reg variables and out is output variable of wire type**



<="" b="" style="margin: 0px; padding: 0px; border: 1px solid rgb(59, 59, 59);">

A . assign out = in1&in2

B . assign #10 out = in1&in2 **=> This is the correct answer .**

C . assign #30 out = in1&in2

D. assign #40 out = in1&in2

59. **What would be simulation time in below code when rval=10**

```
`timescale 1 ns / 1 ps
module timescale_check2;
reg[31:0] rval;
initial begin
rval = 20;
#10.566601 rval = 10;
end
initial begin
$monitor("TimeScale 1ns/1ps : Time=%0t,  rval = %d
",$realtime,rval);
#100 ;
end
endmodule
```

A . 0

B . 10567 **=> This is the correct answer .**

C . 10566 **=> This is the correct answer .**

D. 1056

60. **31. What is order of execution for below code**

```
module zero;
reg a,b,c;
//initial block with zero delay
 initial begin
    #0 a = 1;
    #0 b = 1;
    #0 c = 1;
    (" Zero delay control a= %b, b= %b, c=%b
",a,b,c);
 end

 //initial block without zero delay
 initial  c = 0;

 initial begin
    a = 0;
    b = 0;
    ("Non-zero delay control a= %b, b= %b, c=%b
",a,b,c);
 end

endmodule
```

A . Zero delay control statement is executed first

. Zero delay is executed only after all other statements

B . Zero delay is executed only after all other statements

=> This is the correct answer .

. are executed in that simulation time and order of zero

C . delay statement is non deterministic.

D. Zero delay control statement is executed last

61. **An event is triggered by symbol**

A . =>

B . --->

C . @ **=> This is the correct answer .**

D. None

62. Which construct is used to execute loop fixed number of times

A . repeat
B . while
C . forever
D. None **=> This is the correct answer .**

63. Analyse the code below and choose the right options

```
reg clock;
initial
begin
   clock = 1'b0;
   foever #10 clock = ~clock;
end
```

. The forever loop will execute the statement infinitely

A . without advancing simulation time
. Forever loop will execute the

B . statement infinitely and rest of the design is executed in simulation time **=> This is the correct answer .**
C . Timing control construct is optional in forever
D. None

64. Function has

A . Input,output and inout port as argument
B . Have one input argument only
C . Can have more than one input arguments
D. Can have both input or output as argument. **=> This is the correct answer .**

65. Task and function can't have

A . Registers **=> This is the correct answer .**
B . Time variables
C . Local variables
D. Wires

66. Which is true about task and function

A . Task and function contain always and initial blocks

. Task and function do not contain always and initial blocks

B . and called from initial and always blocks or other tasks or function. **=> This is the correct answer .**

C . Task and function do not contain behavioral statement

D. None

67. Random number is generated by

A . $random **=> This is the correct answer .**

B . $rand

C . $strobe

D. None

68. Special parameters are defined in specify block by

A . defparam

B . specparam **=> This is the correct answer .**

C . parameter

D. none

69. Setup time is

A . Minimum time the data must arrive before active clock edge **=> This is the correct answer .**

B . Maximum time the data must arrive before active clock edge

C . Minimum time the data cannot change after active clock edge **=> This is the correct answer .**

D. None

70. **Arrays are not allowed for**

A . integer
B . register
C . time
D. real **=> This is the correct answer .**

71. **Which directive is used to include entire content of Verilog source Into another file**

A . 'include **=> This is the correct answer .**
B . 'ifdef
C . 'ifndef
D. None

72. **All ports by default are**

A . reg
B . inout
C . wires
D. Should be defined

73. **Inout ports must always be**

A . reg
B . net **=> This is the correct answer .**
C . trireg
D. none

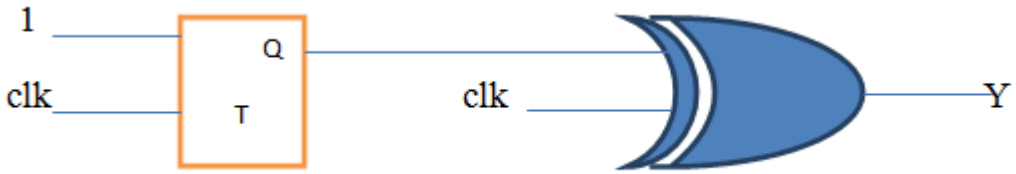
74. **Which is used to introduce delays in circuit**

A . Inverter
B . XOR gate
C . Buffer **=> This is the correct answer .**
D. Flip fops

75. Flip flop is

A . Monostable multivibrator
B . Astable multivibrator
C . Bistable multivibrator **=> This is the correct answer .**
D. None

76. For the diagram given below



A . Frequency of Y equals frequency of clk
B . Frequency of Y is double the frequency of clk
C . Frequency of Y is half the frequency of clk **=> This is the correct answer .**
D. Can't determine

77. NMOS transistor passes

A . strong0 **=> This is the correct answer .**
B . strong1
C . weak0
D. weak1

78. If a variable is not assigned in all possible execution of always statement then

A . A don't care is inferred
B . Latch is inferred **=> This is the correct answer .**
C . Variable is set to 0
D. Synthesis process will fail

79. 50. Analyse below code segment to choose correct answer

```
always @(irq)
begin
{in2,in1,in1} = 3'b0;
casez (irq)
3'b1?? : in2 = 1'b1;
3'b?1? : in1 = 1'b1;
3'b??1 : in0 = 1'b1;
end
```

A . This is non full case **=> This is the correct answer .**
B . This is full case
C . This is parallel case
D. Case which is not parallel **=> This is the correct answer .**

80. The left hand side of procedural continuous assignment can be

A . Register or concatenation of registers **=> This is the correct answer .**
B . Net or concatenation of nets
C . Can be both registers or nets
D. Array of registers

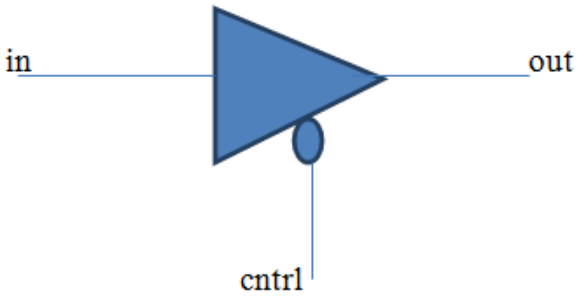
81. Which of the following about resistive switches is correct

A . Low source to drain impedance **=> This is the correct answer .**
B . High source to drain impedance
C . Retain signal strength
D. None

82. Which of the following about UDP's are correct

A . UDP can be defined inside modules
B . UDP can be instantiated inside module **=> This is the correct answer .**
C . UDP supports inout ports
D. UDP can take both scalar and vector input

83. For the diagram given below choose the correct answer



<="" b="" style="margin: 0px; padding: 0px; border: 1px solid rgb(59, 59, 59);">

A . Propagate z if cntrl is asserted
B . Propagate z if cntrl is deasserted **=> This is the correct answer .**
C . Propagate 1 if cntrl is asserted
D. None

84. For the segment given below choose the correct answer

bufif0 #(5,6,7) c1(out,in,cntrl)

A . 5=rise 6=turnoff 7=fall
B . 5=fall 6=rise 7=turnoff
C . 5=rise 6=fall 7=turnoff **=> This is the correct answer .**
D. 5=turnoff 6=rise 7=fall

85. STA(static timing analysis) is a method

A . Of computing expected timing without using simulation **=> This is the correct answer .**
B . Of computing expected timing using simulation
C. None

86. Which of the following is equivalent to logic level 1

A . 1
B . 1'b1
C . 1`b1
D . All of above
E. A,B **=> This is the correct answer .**

87. If X is a ten digit number which of the following will left extend X[9] three times

A . {(X[9], X[9], X[9]), X}
B . (X[9], X[9], X[9], X) **=> This is the correct answer .**
C . {3(X[9]), X[9]}
D. {3{X[9]}, X[9]}

88. Which of the following pertains to parameters

A . The default size of a parameter in most synthesizers is the size of an integer, 32 bits **=> This is the correct answer .**

B . Parameters enable Verilog code to be compatible with VHDL

C . Parameters cannot accept a default value

D . All of above

E. None of the above

89. Which of the following is not true about operators ?

A . A logical "or" is performed by writing "C = A || B;"

B . '!' performs logical negation while '~' performs bitwise negation

C . The two types of "or" operators are "logical" and "bitwise."

D. The "shift right" (>>) operator inserts zeros on the left end of its argument

90. Which of the following is true about always statement

A . There may be exactly one always block in a design.

B . There may be exactly one always block in a module

C . Execution of an always block occurs exactly once per simulation run

D. An always block may be used to generate a periodic signal **=> This is the correct answer .**

91. In most synthesis tools, what will happen when a signal that is needed in a sensitivity list is not included?

A . An error will be generated and the code cannot be synthesized.

B . A warning message will be generated and the code will be synthesized but the resulting netlist will not provide the desired results **=> This is the correct answer .**

C . The synthesis tool will ignore the sensitivity list since all objects that are read as part of a procedural assignment statement are considered to be sensitive

D. There will be no effect on the design and pre-synthesis simulation will be consistent with post-synthesis simulation.

92. Which of the following is used for Verilog-based synthesis tools?

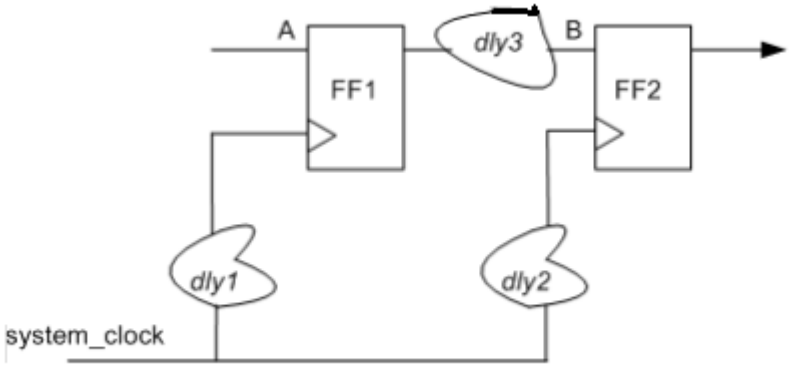
A . Intra-statement delay statements can be synthesized, but interstatement delays cannot
B . Inter-statement delay statements can be synthesized, but intrastatement delays cannot
C . Initial values on wires are almost always ignored.
D. Synthesized results are identical for "if" and " case " statementst **=> This is the correct answer .**

93. Consider the following choices below. To comment Verilog Code, one may use:

I A "Double-slash" // for a single-line comment.
II Multiple "Double-slashes" (one per line) for a multiple-line comment.
III A "Block-comment" /* */ for a single-line comment.
IV A "Block-comment" /* */ for a multiple-line comment.

A . I and II
B . I and IV **=> This is the correct answer .**
C . II and III
D . I, II, and IV
E . I, II, III and IV

94. Analyse the circuit given below with logic delays dly1,dly2,dly3



How can we fix hold and setup time violation in pinB

A . Change dly3

B . Change dly2

C . Change dly2 and dly3

D. Change dly1,dly2,dly3 **=> This is the correct answer .**

95. What is the value of a in given Verilog code

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

A . a =1

B . a=1

C . Both of above

D. A=0 in this simulation and a=1 would be in next simulation **=> This is the correct answer .**

96. Analyse the code segment given below and choose right answer

```
 module top;
-------
---------
assign a=a&b|c;
-------
-------
initial begin
#50 force out = a&b&c;
#50 release out;
end
-----------
------
endmodule
```

A . Expression a&b&c is assigned to net from time 50 to 100

1. a&b&c overrides assignment of a&b

1. a&b&c overrides assignment of a&b

D. A, C **=> This is the correct answer .**

97. **keywords "assign" "deassign" are**

A . continuous assignment
B . procedural continuous assignment **=> This is the correct answer .**
C . blocking assignment
D. nonblocking assignment

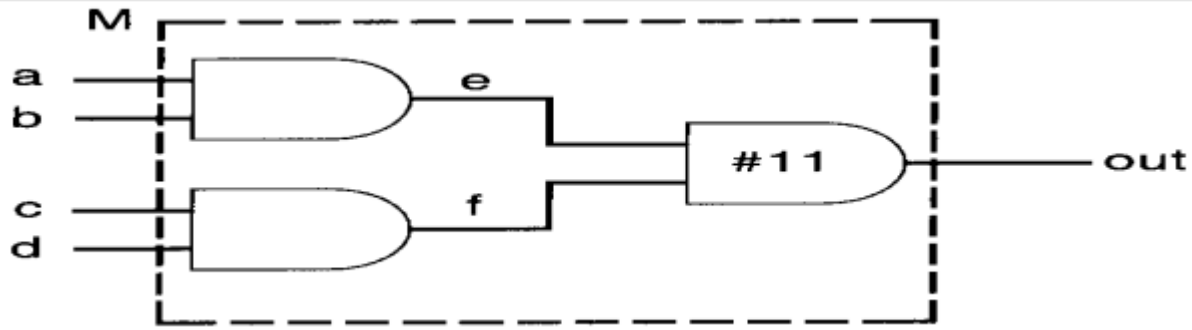
98. **What is missing in top module of below code segment**

```
module my_verilog;
parameter p1 = 5;
initial
 ("displaying my_vrilog = %d",p1);
endmodule
module top;
defparam q1.p1=6,q2.p1=4;

endmodule
```

A . Initial statement
B . One module instance of my_verilog
C . Two module instance of my_verilog **=> This is the correct answer .**
D.

99. **The below figure is an example of**



A . Pin to Pin delay
B . Distributed delay
C . Lumped **=> This is the correct answer .**
D. None

100. **Full connection describes connection between**

A . Each bit of source to each bit of destination
B . Each bit of source to every bit of destination
C . Every bit of source to each bit of destination
D . Only A
E . Only B
F. B & C **=> This is the correct answer .**

101. **For full connection given below how many parallel connections are possible**

```
//a[31:0] is a 32 bit vector and out is [10:0] bit vector
specify
(a *> out) = 9;
endspecify
```

A . 32
B . 100
C . 320
D. 352 **=> This is the correct answer .**

102. **What is the minimum FIFO depth for below specification**

Writing clock =30MHz
Reading clock=40MHz
Burst Size=B=30
8 idle clock cycles for reading

A . 29.9
B . 25
C . 25.2 **=> This is the correct answer .**
D. 37.2

103. **State table in UDP can contain**

A . 0,1
B . 0,1,z
C . 0,1,x **=> This is the correct answer .**
D. 0,1,x,z

104. In UDP

A . Multiple input and one output is permitted **=> This is the correct answer .**

B . One input and multiple output is permitted

C . Multiple input and multiple output is permitted

D. One input and one output is permitted

105. In non blocking assignment

A . Evaluates all RHS for current time unit and assign to LHS at current time

B . Evaluates all RHS for current time unit and assign to LHS at the end of time unit **=> This is the correct answer .**

C . Whole statement is done before control passes to next statement

D. None

106. For the below code segment given below

```
always @(posedge clk)
    a=b;

always @(posedge clk)
    b=a;
```

A . Values of a and b are swapped

B . Values of a and b cannot be swapped **=> This is the correct answer .**

C . Values of a and b can be swapped depending on simulator

D. Can’t determine

107. For the below code segment choose the right answer, if a is 1’bx or 1’bz

```
reg a;
casex (a)
  1'b0 : statement1;
  1'b1 : statement2;
  1'bx : statement3;
  1'bz : statement4;
endcase
```

A . statement 1 is executed

B . procedural continuous assignment **=> This is the correct answer .**

C . blocking assignment

D. nonblocking assignment

108. **Clock skew is the phenomenon of**

A . Synchronous circuit **=> This is the correct answer .**
B . Asynchronous circuit
C . Both
D. None

109. **Cycle based simulation is useful for**

A . Synchronous circuit
B . Asynchronous circuit **=> This is the correct answer .**
C . Both
D. None

110. **For tri0 what is the value when nothing is driving**

A . 1
B . 0 **=> This is the correct answer .**
C . x
D. z

111. **For time check violation is reported if**

A . Tdata_event – Treference_event) <limit=> This is the correct answer .
B . (Tdata_event – Treference_event) >limit
C . (Tdata_event – Treference_event) =limit
D. (Treference_event-Tdata_event) < limit

112. **Which is object oriented programming**

A . VHDL
B . Verilog
C . System Verilog **=> This is the correct answer .**
D. none

113. wand differs from wor in the way

A . wand forced to one if any driver to it is zero while wor forced to zero if any driver to it is zero

B . wand forced to zero if any driver to it is zero while wor forced to one if any driver to it is one **=> This is the correct answer .**

C . wand forced to zero if any driver to it is one while wor forced to one if any driver to it is zero

D. wand forced to one if any driver to it is one while wor forced to zero if any driver to it is zero

114. If there is mismatch in connecting wire such as
Y1[7:0] = Y2[15:0]

A . The end result is Y1[7:0]=Y2[15:0]

B . The end result is Y1[15:0]=Y2[7:0

C . The end result is Y1[7:0]=Y2[7:0] **=> This is the correct answer .**

D. Can't determine

115. What can be Verilog statement for 6 bit register constant C3 with decimal value of 30

A . Parameter C3=6'd30 **=> This is the correct answer .**

B . C3=6'd30

C . Parameter C3=30

D. C3=30

116. What can be the Verilog code that declares an 6-bit register, R_H36, and initially assigns it the hexadecimal value 36

A . reg[5:0] R_H36;initial R_H36=6'h36; **=> This is the correct answer .**

B . reg [5:0] R_H36;R_H36=6'h36;

C . A,B

D. None

117. What could be timescale directive so that d1 and d2 corresponds to 0.3 and 0.7 respectively and simulator time step is 1ps

A . `timescale 1ps/ps
B . `timescale 100ps/ps **=> This is the correct answer .**
C . `timescale 100ns/ps
D. `timescale 1ps/@ps

118. Choose the right answer

A . UDP's are faster to simulate and require less memory than Verilog modules **=> This is the correct answer .**
B . UDP's are faster to simulate and require more memory than Verilog modules
C . UDP's are slower to simulate and require less memory than Verilog modules
D. UDP's are slower to simulate and require more memory than Verilog modules

119. Give a Verilog statement that instantiates the below RTL_circuit, with the instance name RT1, so that x has a delay of 7 time units and y has a delay of 5 time units

```
module RTL_circuit(x, y, a, b, c, d);
parameter x_delay = 3, y_delay = 7;
input a, b, c, d;
output x, y;
assign #y_delay y = (a | b) & (c | ~d);
assign #x_delay x = a ^~ b;
endmodule
```

A . RTL_circuit #(7,5) RT1(y,x,a,b,c,d);
B . RTL_circuit #(5,7) RT1(y,x,a,b,c,d);
C . #(5,7) RTL_circuit RT1(y,x,a,b,c,d);
D. RTL_circuit #(7,5) RT1(x,y,a,b,c,d); **=> This is the correct answer .**

120.Following is the Verilog code for flip-flop with a positive-edge clock.

```
module flop (clk, d, q);
input  clk, d;
output q;
reg    q;
always @(posedge clk)
begin
  q <= d;
end
    endmodule
```

121.Following is Verilog code for a flip-flop with a negative-edge clock and asynchronous clear.

```
module flop (clk, d, clr, q);
input  clk, d, clr;
output q;
reg    q;
always @(negedge clk or posedge clr)
    begin
  if (clr)
     q <= 1'b0;
  else
     q <= d;
end
    endmodule
```

122.Following is Verilog code for the flip-flop with a positive-edge clock and synchronous set.

```
    module flop (clk, d, s, q);
    input  clk, d, s;
    output q;
    reg    q;
    always @(posedge clk)
    begin
      if (s)
        q <= 1'b1;
      else
        q <= d;
    end
    endmodule
```

123.Following is Verilog code for the flip-flop with a positive-edge clock and clock enable.

```
module flop (clk, d, ce, q);
input  clk, d, ce;
output q;
reg    q;
always @(posedge clk)
    begin
  if (ce)
        q <= d;
end
    endmodule
```

124.Following is Verilog code for a 4-bit register with a positive-edge clock, asynchronous set and clock enable.

```
module flop (clk, d, ce, pre, q);
input       clk, ce, pre;
input  [3:0] d;
output [3:0] q;
reg    [3:0] q;
always @(posedge clk or posedge pre)
    begin
  if (pre)
    q <= 4'b1111;
  else if (ce)
    q <= d;
    end
    endmodule
```

125.Following is the Verilog code for a latch with a positive gate.

```
module latch (g, d, q);
    input  g, d;
    output q;
reg    q;
always @(g or d)
    begin
      if (g)
        q <= d;
    end
    endmodule
```

126.Following is the Verilog code for a latch with a positive gate and an asynchronous clear.

```
module latch (g, d, clr, q);
input  g, d, clr;
output q;
reg    q;
always @(g or d or clr)
begin
  if (clr)
    q <= 1'b0;
  else if (g)
    q <= d;
end
endmodule
```

127.Following is Verilog code for a 4-bit latch with an inverted gate and an asynchronous preset.

```
module latch (g, d, pre, q);
input       g, pre;
input  [3:0] d;
output [3:0] q;
reg    [3:0] q;
always @(g or d or pre)
begin
  if (pre)
    q <= 4'b1111;
  else if (~g)
    q <= d;
end
endmodule
```

128.Following is Verilog code for a tristate element using a combinatorial process and always block.

```
module three_st (t, i, o);
input  t, i;
output o;
reg    o;
always @(t or i)
begin
  if (~t)
    o = i;
  else
    o = 1'bZ;
end
endmodule
```

129.Following is the Verilog code for a tristate element using a concurrent assignment.

```
module three_st (t, i, o);
input  t, i;
output o;
  assign o = (~t) ? i: 1'bZ;
  endmodule
```

130.Following is the Verilog code for a 4-bit unsigned up counter with asynchronous clear.

```
module counter (clk, clr, q);
input       clk, clr;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
  if (clr)
    tmp <= 4'b0000;
  else
    tmp <= tmp + 1'b1;
end
  assign q = tmp;
endmodule
```

131.Following is the Verilog code for a 4-bit unsigned down counter with synchronous set.

```
module counter (clk, s, q);
input       clk, s;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk)
begin
  if (s)
    tmp <= 4'b1111;
  else
    tmp <= tmp - 1'b1;
end
  assign q = tmp;
endmodule
```

132.Following is the Verilog code for a 4-bit unsigned up counter with an asynchronous load from the primary input.

```
module counter (clk, load, d, q);
input       clk, load;
input  [3:0] d;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge load)
begin
  if (load)
    tmp <= d;
  else
    tmp <= tmp + 1'b1;
end
  assign q = tmp;
endmodule
```

133.Following is the Verilog code for a 4-bit unsigned up counter with a synchronous load with a constant.

```
module counter (clk, sload, q);
input       clk, sload;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk)
begin
  if (sload)
        tmp <= 4'b1010;
  else
     tmp <= tmp + 1'b1;
end
  assign q = tmp;
   endmodule
```

134.Following is the Verilog code for a 4-bit unsigned up counter with an asynchronous clear and a clock enable.

```
module counter (clk, clr, ce, q);
input       clk, clr, ce;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
  if (clr)
    tmp <= 4'b0000;
  else if (ce)
```

```
      tmp <= tmp + 1'b1;
  end
    assign q = tmp;
     endmodule
```

135.Following is the Verilog code for a 4-bit unsigned up/down counter with an asynchronous clear.

```
module counter (clk, clr, up_down, q);
input       clk, clr, up_down;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
  if (clr)
    tmp <= 4'b0000;
  else if (up_down)
    tmp <= tmp + 1'b1;
  else
    tmp <= tmp - 1'b1;
end
  assign q = tmp;
   endmodule
```

136.Following is the Verilog code for a 4-bit signed up counter with an asynchronous reset.

```
   module counter (clk, clr, q);
   input              clk, clr;
   output signed [3:0] q;
   reg    signed [3:0] tmp;
   always @ (posedge clk or posedge clr)
   begin
     if (clr)
       tmp <= 4'b0000;
     else
       tmp <= tmp + 1'b1;
   end
     assign q = tmp;
   endmodule
```

137.Following is the Verilog code for a 4-bit signed up counter with an asynchronous reset and a modulo maximum.

```
   module counter (clk, clr, q);
   parameter MAX_SQRT = 4, MAX = (MAX_SQRT*MAX_SQRT);
   input              clk, clr;
   output [MAX_SQRT-1:0] q;
   reg    [MAX_SQRT-1:0] cnt;
```

```
always @ (posedge clk or posedge clr)
begin
  if (clr)
    cnt <= 0;
  else
    cnt <= (cnt + 1) %MAX;
end
  assign q = cnt;
endmodule
```

138.Following is the Verilog code for a 4-bit unsigned up accumulator with an asynchronous clear.

```
module accum (clk, clr, d, q);
input       clk, clr;
input  [3:0] d;
output [3:0] q;
reg    [3:0] tmp;
always @(posedge clk or posedge clr)
begin
  if (clr)
    tmp <= 4'b0000;
  else
    tmp <= tmp + d;
end
  assign q = tmp;
endmodule
```

139.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, serial in and serial out.

```
module shift (clk, si, so);
input       clk,si;
output      so;
reg    [7:0] tmp;
always @(posedge clk)
begin
  tmp    <= tmp << 1;
  tmp[0] <= si;
end
  assign so = tmp[7];
   endmodule
```

140.Following is the Verilog code for an 8-bit shift-left register with a negative-edge clock, a clock enable, a serial in and a serial out.

```
module shift (clk, ce, si, so);
input       clk, si, ce;
output      so;
reg    [7:0] tmp;
always @(negedge clk)
begin
  if (ce) begin
    tmp    <= tmp << 1;
    tmp[0] <= si;
  end
end
  assign so = tmp[7];
   endmodule
```

141.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in and serial out.

```
module shift (clk, clr, si, so);
input       clk, si, clr;
output      so;
reg    [7:0] tmp;
always @(posedge clk or posedge clr)
begin
  if (clr)
    tmp <= 8'b00000000;
  else
    tmp <= {tmp[6:0], si};
end
  assign so = tmp[7];
   endmodule
```

141.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a synchronous set, a serial in and a serial out.

```
module shift (clk, s, si, so);
input       clk, si, s;
output      so;
reg    [7:0] tmp;
always @(posedge clk)
begin
  if (s)
    tmp <= 8'b11111111;
  else
    tmp <= {tmp[6:0], si};
end
```

```
assign so = tmp[7];
  endmodule
```

142.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a serial in and a parallel out.

```
module shift (clk, si, po);
input       clk, si;
output [7:0] po;
reg    [7:0] tmp;
always @(posedge clk)
begin
  tmp <= {tmp[6:0], si};
end
  assign po = tmp;
endmodule
```

143.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, an asynchronous parallel load, a serial in and a serial out.

```
module shift (clk, load, si, d, so);
input       clk, si, load;
input  [7:0] d;
output      so;
reg    [7:0] tmp;
always @(posedge clk or posedge load)
begin
  if (load)
        tmp <= d;
  else
    tmp <= {tmp[6:0], si};
end
  assign so = tmp[7];
  endmodule
```

144.Following is the Verilog code for an 8-bit shift-left register with a positive-edge clock, a synchronous parallel load, a serial in and a serial out.

```
module shift (clk, sload, si, d, so);
input       clk, si, sload;
input  [7:0] d;
output      so;
reg    [7:0] tmp;
always @(posedge clk)
begin
  if (sload)
    tmp <= d;
  else
```

```
        tmp <= {tmp[6:0], si};
    end
      assign so = tmp[7];
    endmodule
```

145.Following is the Verilog code for an 8-bit shift-left/shift-right register with a positive-edge clock, a serial in and a serial out.

```
module shift (clk, si, left_right, po);
input       clk, si, left_right;
output      po;
reg    [7:0] tmp;
always @(posedge clk)
begin
  if (left_right == 1'b0)
    tmp <= {tmp[6:0], si};
  else
    tmp <= {si, tmp[7:1]};
end
  assign po = tmp;
   endmodule
```

146.Following is the Verilog code for a 4-to-1 1-bit MUX using an If statement.

```
module mux (a, b, c, d, s, o);
input       a,b,c,d;
input  [1:0] s;
output      o;
reg         o;
always @(a or b or c or d or s)
begin
  if (s == 2'b00)
    o = a;
  else if (s == 2'b01)
    o = b;
  else if (s == 2'b10)
    o = c;
  else
    o = d;
end
    endmodule
```

147.Following is the Verilog Code for a 4-to-1 1-bit MUX using a Case statement.

```
module mux (a, b, c, d, s, o);
input       a, b, c, d;
input  [1:0] s;
output      o;
reg         o;
always @(a or b or c or d or s)
begin
  case (s)
    2'b00   : o = a;
    2'b01   : o = b;
    2'b10   : o = c;
    default : o = d;
  endcase
end
   endmodule
```

148.Following is the Verilog code for a 3-to-1 1-bit MUX with a 1-bit latch.

```
module mux (a, b, c, d, s, o);
input       a, b, c, d;
input  [1:0] s;
output      o;
reg         o;
always @(a or b or c or d or s)
begin
  if (s == 2'b00)
    o = a;
  else if (s == 2'b01)
    o = b;
  else if (s == 2'b10)
    o = c;
end
endmodule
```

149.Following is the Verilog code for a 1-of-8 decoder.

```
module mux (sel, res);
input  [2:0] sel;
output [7:0] res;
reg    [7:0] res;
always @(sel or res)
begin
```

```
      case (sel)
        3'b000  : res = 8'b00000001;
        3'b001  : res = 8'b00000010;
        3'b010  : res = 8'b00000100;
        3'b011  : res = 8'b00001000;
        3'b100  : res = 8'b00010000;
        3'b101  : res = 8'b00100000;
        3'b110  : res = 8'b01000000;
        default : res = 8'b10000000;
      endcase
    end
    endmodule
```

150.Following Verilog code leads to the inference of a 1-of-8 decoder.

```
module mux (sel, res);
input  [2:0] sel;
output [7:0] res;
reg    [7:0] res;
always @(sel or res) begin
  case (sel)
    3'b000  : res = 8'b00000001;
    3'b001  : res = 8'b00000010;
    3'b010  : res = 8'b00000100;
    3'b011  : res = 8'b00001000;
    3'b100  : res = 8'b00010000;
    3'b101  : res = 8'b00100000;
    // 110 and 111 selector values are unused
    default : res = 8'bxxxxxxxx;
  endcase
end
   endmodule
```

151.Following is the Verilog code for a 3-bit 1-of-9 Priority Encoder.

```
module priority (sel, code);
input  [7:0] sel;
output [2:0] code;
reg    [2:0] code;
always @(sel)
begin
  if (sel[0])
    code = 3'b000;
  else if (sel[1])
    code = 3'b001;
  else if (sel[2])
    code = 3'b010;
```

```
  else if (sel[3])
    code = 3'b011;
  else if (sel[4])
    code = 3'b100;
  else if (sel[5])
    code = 3'b101;
  else if (sel[6])
    code = 3'b110;
  else if (sel[7])
    code = 3'b111;
  else
    code = 3'bxxx;
end
   endmodule
```

152.Following is the Verilog code for a logical shifter.

```
module lshift (di, sel, so);
   input  [7:0] di;
input  [1:0] sel;
output [7:0] so;
reg    [7:0] so;
always @(di or sel)
begin
  case (sel)
    2'b00   : so = di;
    2'b01   : so = di << 1;
    2'b10   : so = di << 2;
    default : so = di << 3;
  endcase
end
   endmodule
```

153.Following is the Verilog code for an unsigned 8-bit adder with carry in.

```
module adder(a, b, ci, sum);
input  [7:0] a;
input  [7:0] b;
input        ci;
output [7:0] sum;

  assign sum = a + b + ci;

   endmodule
```

154.Following is the Verilog code for an unsigned 8-bit adder with carry out.

```
module adder(a, b, sum, co);
input  [7:0] a;
input  [7:0] b;
output [7:0] sum;
output       co;
wire   [8:0] tmp;

  assign tmp = a + b;
  assign sum = tmp [7:0];
  assign co  = tmp [8];

   endmodule
```

155.Following is the Verilog code for an unsigned 8-bit adder with carry in and carry out.

```
   module adder(a, b, ci, sum, co);
   input        ci;
   input  [7:0] a;
   input  [7:0] b;
   output [7:0] sum;
   output       co;
   wire   [8:0] tmp;

     assign tmp = a + b + ci;
     assign sum = tmp [7:0];
     assign co  = tmp [8];

   endmodule
```

156.Following is the Verilog code for an unsigned 8-bit adder/subtractor.

```
module addsub(a, b, oper, res);
input       oper;
input  [7:0] a;
input  [7:0] b;
output [7:0] res;
reg    [7:0] res;
always @(a or b or oper)
begin
  if (oper == 1'b0)
    res = a + b;
  else
    res = a - b;
   end
   endmodule
```

157.Following is the Verilog code for an unsigned 8-bit greater or equal comparator.

```
module compar(a, b, cmp);
input  [7:0] a;
input  [7:0] b;
output       cmp;

  assign cmp = (a >= b) ?  1'b1 : 1'b0;

  endmodule
```

158.Following is the Verilog code for an unsigned 8x4-bit multiplier.

```
module compar(a, b, res);
input  [7:0]  a;
input  [3:0]  b;
output [11:0] res;

   assign res = a * b;

endmodule
```

159.Following Verilog template shows the multiplication operation placed outside the always block and the pipeline stages represented as single registers.

```
module mult(clk, a, b, mult);
input        clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
wire   [35:0] mult_res;
reg    [35:0] pipe_1, pipe_2, pipe_3;

   assign mult_res = a_in * b_in;

always @(posedge clk)
begin
a_in   <= a;
   b_in   <= b;
   pipe_1 <= mult_res;
   pipe_2 <= pipe_1;
   pipe_3 <= pipe_2;
   mult   <= pipe_3;
end
endmodule
```

160.Following Verilog template shows the multiplication operation placed inside the always block and the pipeline stages are represented as single registers.

```
module mult(clk, a, b, mult);
input        clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
reg    [35:0] mult_res;
reg    [35:0] pipe_2, pipe_3;
always @(posedge clk)
begin
a_in     <= a;
   b_in     <= b;
   mult_res <= a_in * b_in;
   pipe_2   <= mult_res;
   pipe_3   <= pipe_2;
   mult     <= pipe_3;
end
endmodule
```

161.Following Verilog template shows the multiplication operation placed outside the always block and the pipeline stages represented as single registers.

```
module mult(clk, a, b, mult);
input        clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
wire   [35:0] mult_res;
reg    [35:0] pipe_1, pipe_2, pipe_3;

  assign mult_res = a_in * b_in;

always @(posedge clk)
begin
  a_in   <= a;
  b_in   <= b;
  pipe_1 <= mult_res;
  pipe_2 <= pipe_1;
  pipe_3 <= pipe_2;
  mult   <= pipe_3;
end
```

```
endmodule
```

162.Following Verilog template shows the multiplication operation placed inside the always block and the pipeline stages are represented as single registers.

```
module mult(clk, a, b, mult);
input        clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
reg    [35:0] mult_res;
reg    [35:0] pipe_2, pipe_3;
always @(posedge clk)
begin
  a_in     <= a;
  b_in     <= b;
  mult_res <= a_in * b_in;
  pipe_2   <= mult_res;
  pipe_3   <= pipe_2;
  mult     <= pipe_3;
end
   endmodule
```

163.Following Verilog template shows the multiplication operation placed outside the always block and the pipeline stages represented as shift registers.

```
module mult3(clk, a, b, mult);
input        clk;
input  [17:0] a;
input  [17:0] b;
output [35:0] mult;
reg    [35:0] mult;
reg    [17:0] a_in, b_in;
wire   [35:0] mult_res;
reg    [35:0] pipe_regs [3:0];

  assign mult_res = a_in * b_in;

always @(posedge clk)
begin
  a_in <= a;
  b_in <= b;
  {pipe_regs[3],pipe_regs[2],pipe_regs[1],pipe_regs[0]} <=
     {mult, pipe_regs[3],pipe_regs[2],pipe_regs[1]};
end
```

```
endmodule
```

164.Following templates to implement Multiplier Adder with 2 Register Levels on Multiplier Inputs in Verilog.

```
module mvl_multaddsub1(clk, a, b, c, res);
input        clk;
input  [07:0] a;
input  [07:0] b;
input  [07:0] c;
output [15:0] res;
reg    [07:0] a_reg1, a_reg2, b_reg1, b_reg2;
wire   [15:0] multaddsub;
always @(posedge clk)
begin
  a_reg1 <= a;
  a_reg2 <= a_reg1;
  b_reg1 <= b;
  b_reg2 <= b_reg1;
end
  assign multaddsub = a_reg2 * b_reg2 + c;
  assign res = multaddsub;
   endmodule
```

165.Following is the Verilog code for resource sharing.

```
module addsub(a, b, c, oper, res);
input       oper;
input  [7:0] a;
input  [7:0] b;
input  [7:0] c;
output [7:0] res;
reg    [7:0] res;
always @(a or b or c or oper)
begin
  if (oper == 1'b0)
    res = a + b;
  else
    res = a - c;
end
   endmodule
```

166.Following templates show a single-port RAM in read-first mode.

```
module raminfr (clk, en, we, addr, di, do);
input       clk;
input       we;
input       en;
```

```
input  [4:0] addr;
input  [3:0] di;
output [3:0] do;
reg    [3:0] RAM [31:0];
reg    [3:0] do;
always @(posedge clk)
begin
  if (en) begin
    if (we)
          RAM[addr] <= di;

        do <= RAM[addr];
  end
end
   endmodule
```

167.Following templates show a single-port RAM in write-first mode.

```
module raminfr (clk, we, en, addr, di, do);
input       clk;
input       we;
input       en;
input  [4:0] addr;
input  [3:0] di;
output [3:0] do;
reg    [3:0] RAM [31:0];
reg    [4:0] read_addr;
always @(posedge clk)
begin
  if (en) begin
    if (we)
          RAM[addr] <= di;
       read_addr <= addr;
  end
end
  assign do = RAM[read_addr];
   endmodule
```

168.Following templates show a single-port RAM in no-change mode.

```
module raminfr (clk, we, en, addr, di, do);
input       clk;
input       we;
input       en;
input  [4:0] addr;
input  [3:0] di;
output [3:0] do;
```

```
reg    [3:0] RAM [31:0];
reg    [3:0] do;
always @(posedge clk)
begin
  if (en) begin
    if (we)
          RAM[addr] <= di;
    else
          do <= RAM[addr];
  end
end
    endmodule
```

169.Following is the Verilog code for a single-port RAM with asynchronous read.

```
    module raminfr (clk, we, a, di, do);
    input       clk;
    input       we;
    input  [4:0] a;
    input  [3:0] di;
    output [3:0] do;
    reg    [3:0] ram [31:0];
    always @(posedge clk)
    begin
  if (we)
       ram[a] <= di;
    end
      assign do = ram[a];
    endmodule
```

170.Following is the Verilog code for a single-port RAM with "false" synchronous read.

```
module raminfr (clk, we, a, di, do);
input       clk;
input       we;
input  [4:0] a;
input  [3:0] di;
output [3:0] do;
reg    [3:0] ram [31:0];
reg    [3:0] do;
always @(posedge clk)
begin
  if (we)
    ram[a] <= di;
  do <= ram[a];
end
    endmodule
```

171.Following is the Verilog code for a single-port RAM with synchronous read (read through).

```
module raminfr (clk, we, a, di, do);
input       clk;
input       we;
input  [4:0] a;
input  [3:0] di;
output [3:0] do;
reg    [3:0] ram [31:0];
reg    [4:0] read_a;
always @(posedge clk)
begin
  if (we)
    ram[a] <= di;
  read_a <= a;
end
  assign do = ram[read_a];
   endmodule
```

172.Following is the Verilog code for a single-port block RAM with enable.

```
module raminfr (clk, en, we, a, di, do);
input       clk;
input       en;
input       we;
input  [4:0] a;
input  [3:0] di;
output [3:0] do;
reg    [3:0] ram [31:0];
reg    [4:0] read_a;
always @(posedge clk)
begin
  if (en) begin
    if (we)
      ram[a] <= di;
    read_a <= a;
  end
end
  assign do = ram[read_a];
endmodule
```

173.Following is the Verilog code for a dual-port RAM with asynchronous read.

```
module raminfr (clk, we, a, dpra, di, spo, dpo);
input       clk;
input       we;
input  [4:0] a;
input  [4:0] dpra;
input  [3:0] di;
output [3:0] spo;
output [3:0] dpo;
reg    [3:0] ram [31:0];
always @(posedge clk)
begin
  if (we)
     ram[a] <= di;
end
  assign spo = ram[a];
  assign dpo = ram[dpra];
   endmodule
```

174.Following is the Verilog code for a dual-port RAM with false synchronous read.

```
    module raminfr (clk, we, a, dpra, di, spo, dpo);
    input       clk;
    input       we;
    input  [4:0] a;
    input  [4:0] dpra;
    input  [3:0] di;
    output [3:0] spo;
    output [3:0] dpo;
    reg    [3:0] ram [31:0];
    reg    [3:0] spo;
    reg    [3:0] dpo;
  always @(posedge clk)
    begin
      if (we)
        ram[a] <= di;

      spo = ram[a];
      dpo = ram[dpra];
    end
    endmodule
```

175.Following is the Verilog code for a dual-port RAM with synchronous read (read through).

```
module raminfr (clk, we, a, dpra, di, spo, dpo);
input       clk;
input       we;
input  [4:0] a;
input  [4:0] dpra;
input  [3:0] di;
output [3:0] spo;
output [3:0] dpo;
reg    [3:0] ram [31:0];
reg    [4:0] read_a;
reg    [4:0] read_dpra;
always @(posedge clk)
begin
  if (we)
    ram[a] <= di;
  read_a <= a;
  read_dpra <= dpra;
end
  assign spo = ram[read_a];
  assign dpo = ram[read_dpra];
   endmodule
```

176.Following is the Verilog code for a dual-port RAM with enable on each port.

```
module raminfr (clk, ena, enb, wea, addra, addrb, dia, doa, dob);
input       clk, ena, enb, wea;
input  [4:0] addra, addrb;
input  [3:0] dia;
output [3:0] doa, dob;
reg    [3:0] ram [31:0];
reg    [4:0] read_addra, read_addrb;
always @(posedge clk)
begin
  if (ena) begin
    if (wea) begin
          ram[addra] <= dia;
    end
  end
end

always @(posedge clk)
begin
  if (enb) begin
    read_addrb <= addrb;
  end
```

```
end
  assign doa = ram[read_addra];
  assign dob = ram[read_addrb];
   endmodule
```

177.Following is Verilog code for a ROM with registered output.

```
module rominfr (clk, en, addr, data);
input      clk;
input      en;
input [4:0] addr;
output reg [3:0] data;
always @(posedge clk)
    begin
      if (en)
        case(addr)
          4'b0000: data <= 4'b0010;
          4'b0001: data <= 4'b0010;
          4'b0010: data <= 4'b1110;
          4'b0011: data <= 4'b0010;
          4'b0100: data <= 4'b0100;
          4'b0101: data <= 4'b1010;
          4'b0110: data <= 4'b1100;
          4'b0111: data <= 4'b0000;
          4'b1000: data <= 4'b1010;
          4'b1001: data <= 4'b0010;
          4'b1010: data <= 4'b1110;
          4'b1011: data <= 4'b0010;
          4'b1100: data <= 4'b0100;
          4'b1101: data <= 4'b1010;
          4'b1110: data <= 4'b1100;
          4'b1111: data <= 4'b0000;
          default: data <= 4'bXXXX;
        endcase
    end
    endmodule
```

178.Following is Verilog code for a ROM with registered address.

```
module rominfr (clk, en, addr, data);
input      clk;
input      en;
input [4:0] addr;
output reg [3:0] data;
reg   [4:0] raddr;
always @(posedge clk)
begin
```

```
  if (en)
    raddr <= addr;
end

always @(raddr)
begin
  if (en)
    case(raddr)
        4'b0000: data = 4'b0010;
        4'b0001: data = 4'b0010;
        4'b0010: data = 4'b1110;
        4'b0011: data = 4'b0010;
        4'b0100: data = 4'b0100;
        4'b0101: data = 4'b1010;
        4'b0110: data = 4'b1100;
        4'b0111: data = 4'b0000;
        4'b1000: data = 4'b1010;
        4'b1001: data = 4'b0010;
        4'b1010: data = 4'b1110;
        4'b1011: data = 4'b0010;
        4'b1100: data = 4'b0100;
        4'b1101: data = 4'b1010;
        4'b1110: data = 4'b1100;
        4'b1111: data = 4'b0000;
        default: data = 4'bXXXX;
    endcase
end
   endmodule
```

179.Following is the Verilog code for an FSM with a single process.

```
module fsm (clk, reset, x1, outp);
input       clk, reset, x1;
output      outp;
reg         outp;
reg    [1:0] state;
parameter s1 = 2'b00; parameter s2 = 2'b01;
parameter s3 = 2'b10; parameter s4 = 2'b11;
always @(posedge clk or posedge reset)
begin
  if (reset) begin
    state <= s1; outp <= 1'b1;
  end
  else begin
    case (state)
        s1: begin
```

```
                    if (x1 == 1'b1) begin
                      state <= s2;
                    outp  <= 1'b1;
                     end
                     else begin
                       state <= s3;
                    outp  <= 1'b1;
                     end
                end
              s2: begin
                       state <= s4;
                   outp  <= 1'b0;
                end
              s3: begin
                       state <= s4;
                   outp  <= 1'b0;
                end
              s4: begin
                       state <= s1;
                   outp  <= 1'b1;
                end
       endcase
     end
   end
      endmodule
```

180.Following is the Verilog code for an FSM with two processes.

```
module fsm (clk, reset, x1, outp);
input       clk, reset, x1;
output      outp;
reg         outp;
reg    [1:0] state;
parameter s1 = 2'b00; parameter s2 = 2'b01;
parameter s3 = 2'b10; parameter s4 = 2'b11;
always @(posedge clk or posedge reset)
begin
  if (reset)
    state <= s1;
  else begin
    case (state)
          s1: if (x1 == 1'b1)
                   state <= s2;
              else
                   state <= s3;
          s2: state <= s4;
```

```
            s3: state <= s4;
            s4: state <= s1;
        endcase
      end
    end
    always @(state) begin
      case (state)
        s1: outp = 1'b1;
        s2: outp = 1'b1;
        s3: outp = 1'b0;
        s4: outp = 1'b0;
      endcase
    end
       endmodule
```

181.Following is the Verilog code for an FSM with three processes.

```
    module fsm (clk, reset, x1, outp);
    input       clk, reset, x1;
    output      outp;
    reg         outp;
    reg    [1:0] state;
    reg    [1:0] next_state;
    parameter s1 = 2'b00; parameter s2 = 2'b01;
    parameter s3 = 2'b10; parameter s4 = 2'b11;
    always @(posedge clk or posedge reset)
    begin
      if (reset)
        state <= s1;
      else
        state <= next_state;
    end

    always @(state or x1)
    begin
      case (state)
        s1: if (x1 == 1'b1)
              next_state = s2;
            else
              next_state = s3;
        s2: next_state = s4;
        s3: next_state = s4;
        s4: next_state = s1;
      endcase
       end
  endmodule
```

182.Write a Verilog code to swap contents of two registers with and without a temporary register?

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
```

183. Difference between blocking and non-blocking?

(Verilog interview questions that is most commonly asked)

The Verilog language has two forms of the procedural assignment statement: blocking and non-blocking. The two are distinguished by the = and <= assignment operators. The blocking assignment statement (= operator) acts much like in traditional programming languages. The whole statement is done before control passes on to the next statement. The non-blocking (<= operator) evaluates all the right-hand sides for the current time unit and assigns the left-hand sides at the end of the time unit. For example, the following Verilog program

```
// testing blocking and non-blocking assignment
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
```

produces the following output:
Blocking: A= 00000100 B= 00000101
Non-blocking: A= 00000100 B= 00000100

The effect is for all the non-blocking assignments to use the old values of the variables at the beginning of the current time unit and to assign the registers new values at the end of the current time unit. This reflects how register transfers occur in some hardware systems.
blocking procedural assignment is used for combinational logic and non-blocking procedural assignment for sequential

184.Tell me about verilog file I/O?

OPEN A FILE

integer file;
file = $fopenr("filename");
file = $fopenw("filename");
file = $fopena("filename");
The function $fopenr opens an existing file for reading. $fopenw opens a new file for writing, and $fopena opens a new file for writing where any data will be appended to the end of the file. The file name can be either a quoted string or a reg holding the file name. If the file was successfully opened, it returns an integer containing the file number (1..MAX_FILES) or NULL (0) if there was an error. Note that these functions are not the same as the built-in system function $fopen which opens a file for writing by $fdisplay. The files are opened in C with 'rb', 'wb', and 'ab' which allows reading and writing binary data on the PC. The 'b' is ignored on UNIX.

CLOSE A FILE

integer file, r;
r = $fcloser(file);
r = $fclosew(file);

The function $fcloser closes a file for input. $fclosew closes a file for output. It returns EOF if there was an error, otherwise 0. Note that these are not the same as $fclose which closes files for writing.

185.Difference between task and function?

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 there 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.

186. Difference between inter statement and intra statement delay?

```
//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
```

187. What is delta simulation time?

188. Difference between $monitor,$display & $strobe?
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, ...);

189.What is difference between Verilog full case and parallel case?

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."

190. What is meant by inferring latches,how to avoid it?

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.

191.Tell me how blocking and non blocking statements get executed?

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.

192. Variable and signal which will be Updated first?

Signals

193. What is sensitivity list?

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.

194. In a pure combinational circuit is it necessary to mention all the inputs in sensitivity disk? if yes, why?

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.

195. Tell me structure of Verilog code you follow?

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
```

196.Difference between Verilog and vhdl?

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.

197. What are different styles of Verilog coding I mean gate-level,continuous level and others explain in detail?

198. Can you tell me some of system tasks and their purpose?

$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

199.Can you list out some of enhancements in Verilog 2001?

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
);

200.Write a Verilog code for synchronous and asynchronous reset?

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

201.What is pli?why is it used?

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.

202.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?

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

203.What is difference between freeze deposit and force?

$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.

204.Will case infer priority register if yes how give an example?

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
```

205.Casex,z difference,which is preferable,why?

CASEZ :
Special version of the case statement which uses a Z logic value to represent don't-care bits.
CASEX :
Special version of the case statement which uses Z or X logic values to represent don't-care bits.
CASEZ should be used for case statements with wildcard don’t cares, otherwise use of CASE is required; CASEX should never be used.

This is because:
Don’t cares are not allowed in the "case" statement. Therefore casex or casez are required. Casex will automatically match any x or z with anything in the case statement. Casez will only match z’s -- x’s require an absolute match.

206.Given the following Verilog code, what value of "a" is displayed?

```
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:
1: Active Events (blocking statements)
2: Inactive Events (#0 delays, etc)
3: Non-Blocking Assign Updates (non-blocking statements)
4: 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.

207.What is the difference between the following two lines of Verilog code?
#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.

208.What is the difference between:

c = foo ? a : b;
and
if (foo) c = a;
else c = b;

The ? merges answers if the condition is "x", so for instance if foo = 1'bx, a = 'b10, and b = 'b11, you'd get c = 'b1x. On the other hand, if treats Xs or Zs as FALSE, so you'd always get c = b.
209.What are Intertial and Transport Delays ??

210.What does `timescale 1 ns/ 1 ps signify in a verilog code?

'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.

211. What is the difference between === and == ?
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.

212.How to generate sine wav using verilog coding style?

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

213.What is the difference between wire and reg?

Net types: (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.

214.How do you implement the bi-directional ports in Verilog HDL?

```
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
```

215.what is verilog case (1) ?

```
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).

216.Why is it that "if (2'b01 & 2'b10)..." doesn't run the true case?

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

217.What are Different types of Verilog Simulators ?

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:

1.) Results are only examined at the end of every clock cycle; and
2.) 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.

218.What is Constrained-Random Verification ?

Introduction

As ASIC and system-on-chip (SoC) designs continue to increase in size and complexity, there is an equal or greater increase in the size of the verification effort required to achieve functional coverage goals. This has created a trend in RTL verification techniques to employ constrained-random verification, which shifts the emphasis from hand-authored tests to utilization of compute resources. With the corresponding emergence of faster, more complex

bus standards to handle the massive volume of data traffic there has also been a renewed significance for verification IP to speed the time taken to develop advanced testbench environments that include randomization of bus traffic.

Directed-Test Methodology

Building a directed verification environment with a comprehensive set of directed tests is extremely time-consuming and difficult. Since directed tests only cover conditions that have been anticipated by the verification team, they do a poor job of covering corner cases. This can lead to costly re-spins or, worse still, missed market windows.

Traditionally verification IP works in a directed-test environment by acting on specific testbench commands such as read, write or burst to generate transactions for whichever protocol is being tested. This directed traffic is used to verify that an interface behaves as expected in response to valid transactions and error conditions. The drawback is that, in this directed methodology, the task of writing the command code and checking the responses across the full breadth of a protocol is an overwhelming task. The verification team frequently runs out of time before a mandated tape-out date, leading to poorly tested interfaces. However, the bigger issue is that directed tests only test for predicted behavior and it is typically the unforeseen that trips up design teams and leads to extremely costly bugs found in silicon.

Constrained-Random Verification Methodology

The advent of constrained-random verification gives verification engineers an effective method to achieve coverage goals faster and also help find corner-case problems. It shifts the emphasis from writing an enormous number of directed tests to writing a smaller set of constrained-random scenarios that let the compute resources do the work. Coverage goals are achieved not by the sheer weight of manual labor required to hand-write directed tests but by the number of processors that can be utilized to run random seeds. This significantly reduces the time required to achieve the coverage goals.

Scoreboards are used to verify that data has successfully reached its destination, while monitors snoop the interfaces to provide coverage information. New or revised constraints focus verification on the uncovered parts of the design under test. As verification progresses, the simulation tool identifies the best seeds, which are then retained as regression tests to create a set of scenarios, constraints, and seeds that provide high coverage of the design.

219.What are the rules governing usage of a Verilog function?

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.

220.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?

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.

221.What are the differences between blocking and nonblocking assignments?

While both blocking and nonblocking assignments are procedural assignments, they differ in behaviour with respect to simulation and logic synthesis as follows:

Table 1-3. Differences between blocking and nonblocking assignments

Blocking assignments	**Nonblocking assignments**
In a blocking assignment, the evaluation of the expression on the RHS is updated to the LHS variable autonomously based on the delay value (either 0 if no delay specified, or scheduled as a future event if a non-0 value is specified)	Nonblocking assignment to LHS is *scheduled* to occur when the next evaluation cycle occurs in simulation and not immediately. Updates are not available immediately within the same time unit
When multiple blocking assignments are present in a process, the trailing assignments are blocked from occurring until the current assignment is completed	Multiple nonblocking assignments can be scheduled to occur concurrently on the next evaluation cycle in simulation
There is a possibility of race conditions on the variables of blocking assignments if assignments happen to it from two processes concurrently	The race conditions are avoided as the updated value is assigned after evaluation
Recommended to use within combinatorial **always** blocks	Recommended to use within the sequential **always** blocks
Can be used in procedural assignments like ***initial***, ***always*** and continuous assignments to nets like ***assign*** statements	Can be used only in the procedural blocks like ***initial*** and ***always***; Continuous assignment to nets like the ***assign*** statement is not permitted
Represented by "=" operator sign between LHS and RHS	Represented by "<=" operator sign between LHS and RHS

222.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. For example, in the following statement, changes to the rhs net will update the lhs net, but not vice versa.

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 lh s , and vice versa.

```
wire rhs , lhs
assign lhs=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 lh s , and vice versa.

```
module test ();
wire rhs,lhs;

alias lhs=rhs;
```

In the above example, any change to either side of the net gets reflected on the other side.

223.Are tasks and functions re-entrant, and how are they different from static task and function calls?

In Verilog-95, tasks and functions were not re-entrant. From Verilog version 2001 onwards, the tasks and functions are reentrant. The reentrant tasks have a keyword automatic between the keyword task and the name of the task. The presence of the keyword automatic replicates and allocates the variables within a task dynamically for each task entry during concurrent task calls, i.e., the values don’t get overwritten for each task call. Without the keyword, the variables are allocated statically, which means these variables are shared across different task calls, and can hence get overwritten by each task call.

Reentrant task	Static task
Has the keyword ***automatic*** between the ***task*** keyword and identifier	*Doesn't* have the keyword ***automatic*** between the ***task*** keyword and the identifier
Variables declared within the task are allocated dynamically for each concurrent task call	Variable declarations within the task are allocated statically
All variables will be replicated in each concurrent call to store state specific to that invocation	Each concurrent call to the task will OVERWRITE the statically allocated local variables of the task from all other concurrent calls to the task
Variables declared are de-allocated at the end of task invocation	Variables retain their values between invocations
Task items cannot be accessed by hierarchical inferences	Task items can be accessed by hierarchical inferences
Task items shall be allocated new across all uses of the task executing concurrently	Task items can be shared across all uses of the task executing concurrently

224.How can I override variables in an automatic task?

By default, all variables in a module are static, i.e., these variables will be replicated for all instances of a module. However, in the case of task and function, either the task/function itself or the variables within them can be defined as static or automatic. The following explains the inferences through different combinations of the task/function and/or its variables, declared either as static or automatic:

No automatic definition of task/function or its variables This is the Verilog-1995 format, wherein the task/function and its variables were implicitly static. The variables are allocated only once. Without the mention of the automatic keyword, multiple calls to task/function will override their variables.

static task/function definition

System Verilog introduced the keyword static. When a task/function is explicitly defined as static, then its variables are allocated only once, and can be overridden. This scenario is exactly the same scenario as before.

automatic task/function definition

From Verilog-2001 onwards, and included within SystemVerilog, when the task/function is declared as automatic, its variables are also implicitly automatic. Hence, during multiple calls of the task/function, the variables are allocated each time and replicated without any overwrites.

static task/function and automatic variables

SystemVerilog also allows the use of automatic variables in a static task/function. Those without any changes to automatic variables will remain implicitly static. This will be useful in scenarios wherein the implicit static variables need to be initialised before the task call, and the automatic variables can be allocated each time.

automatic task/function and static variables

SystemVerilog also allows the use of static variables in an automatic task/function. Those without any changes to static variables will remain implicitly automatic. This will be useful in scenarios wherein the static variables need to be updated for each call, whereas the rest can be allocated each time.

225.How do I prevent selected parameters of a module from being overridden during instantiation?

If a particular parameter within a module should be prevented from being overridden, then it should be declared using the localparam construct, rather than the parameter construct. The localparam construct has been introduced from Verilog-2001. Note that a localparam variable is fully identical to being defined as a parameter, too. In the following example, the localparam construct is used to specify num_bits, and hence trying to override it directly gives an error message.

```
module localparam_list (addr, data);
parameter width = 32;
parameter depth = 64;
localparam num_bits = width * depth;
input  [width-1 : 0] addr;
input  [depth-1 : 0] data;
...
endmodule
```

Note, however, that, since the width and depth are specified using the parameter construct, they can be overridden during instantiation or using defparam, and hence will indirectly override the num_bits values. In general, localparam constructs are useful in defining new and localized identifiers whose values are derived from regular parameters.

226.What are the pros and cons of specifying the parameters using the defparam construct vs. specifying during instantiation?

The advantages of specifying parameters during instantiation method are:

All the values to all the parameters don't need to be specified. Only those parameters that are assigned the new values need to be specified. The unspecified parameters will retain their default values specified within its module definition.

The order of specifying the parameter is not relevant anymore, since the parameters are directly specified and linked by their name.

The disadvantage of specifying parameter during instantiation are:

This has a lower precedence when compared to assigning using defparam.

The advantages of specifying parameter assignments using defparam are:

This method always has precedence over specifying parameters during instantiation.

All the parameter value override assignments can be grouped inside one module and together in one place, typically in the top-level testbench itself.

When multiple defparams for a single parameter are specified, the parameter takes the value of the last defparam statement encountered in the source if, and only if, the multiple defparam's are in the same file. If there are defparam's in different files that override the same parameter, the final value of the parameter is indeterminate.

The disadvantages of specifying parameter assignments using defparam are:

The parameter is typically specified by the scope of the hierarchies underneath which it exists. If a particular module gets ungrouped in its hierarchy, [sometimes necessary during synthesis], then the scope to specify the parameter is lost, and is unspecified. B

For example, if a module is instantiated in a simulation testbench, and its internal parameters are then overridden using hierarchical defparam constructs (For example, defparam U1.U_fifo.width = 32;). Later, when this module is synthesized, the internal hierarchy within U1 may no longer exist in the gate-level netlist, depending upon the synthesis strategy chosen. Therefore post-synthesis simulation will fail on the hierarchical defparam override.

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

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

228.How is the connectivity established in Verilog when connecting wires of different widths?

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.

229.Can I use a Verilog function to define the width of a multi-bit port, wire, or reg type?

The width elements of ports, wire or reg declarations require a constant in both MSB and LSB. Before Verilog 2001, it is a syntax error to specify a function call to evaluate the value of these widths.

For example, the following code is erroneous before Verilog 2001 version.
reg [port1(val1:vla2) : port2 (val3:val4)] reg1;

In the above example, get_high and get_low are both function calls of evaluating a constant result for MSB and LSB respectively. However, Verilog-2001 allows the use of a function call to evaluate the MSB or LSB of a width declaration

230.What is the implication of a combinatorial feedback loops in design testability?

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.

231.What are the various methods to contain power during RTL coding?

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.

-Verilog gate level expected questions.

232.Tell something about why we do gate level simulations?

a. Since scan and other test structures are added during and after synthesis, they are not checked by the rtl simulations and therefore need to be verified by gate level simulation.
b. Static timing analysis tools do not check asynchronous interfaces, so gate level simulation is required to look at the timing of these interfaces.
c. Careless wildcards in the static timing constraints set false path or mutlicycle path constraints where they don't belong.

d. Design changes, typos, or misunderstanding of the design can lead to incorrect false paths or multicycle paths in the static timing constraints.
e. Using create_clock instead of create_generated_clock leads to incorrect static timing between clock domains.
f. Gate level simulation can be used to collect switching factor data for power estimation.
g. X's in RTL simulation can be optimistic or pessimistic. The best way to verify that the design does not have any unintended dependence on initial conditions is to run gate level simulation.
f. It's a nice "warm fuzzy" that the design has been implemented correctly.

233. Say if I perform Formal Verification say Logical Equivalence across Gatelevel netlists(Synthesis and post routed netlist). Do you still see a reason behind GLS.?

If we have verified the Synthesized netlist functionality is correct when compared to RTL and when we compare the Synthesized netlist versus Post route netlist logical Equivalence then I think we may not require GLS after P & R. But how do we ensure on Timing . To my knowledge Formal Verification Logical Equivalence Check does not perform Timing checks and dont ensure that the design will work on the operating frequency , so still I would go for GLS after post route database.

234.An AND gate and OR gate are given inputs X & 1 , what is expected output?

AND Gate output will be X

OR Gate output will be 1.

235.What is difference between NMOS & RNMOS?

RNMOS is resistive nmos that is in simulation strength will decrease by one unit , please refer to below Diagram.

Input Strength	Output Strength
supply	pull
strong	pull
pull	weak
weak	medium
large	medium
medium	small
small	small
highz	highz

236.Tell something about modeling delays in verilog?

Verilog can model delay types within its specification for gates and buffers. Parameters that can be modelled are T_rise, T_fall and T_turnoff. To add further detail, each of the three values can have minimum, typical and maximum values

T_rise, t_fall and t_off

Delay modelling syntax follows a specific discipline;
gate_type #(t_rise, t_fall, t_off) gate_name (paramters);
When specifiying the delays it is not necessary to have all of the delay values specified. However, certain rules are followed.

and #(3) gate1 (out1, in1, in2);
When only 1 delay is specified, the value is used to represent all of the delay types, i.e. in this example, t_rise = t_fall = t_off = 3.

or #(2,3) gate2 (out2, in3, in4);
When two delays are specified, the first value represents the rise time, the second value represents the fall time. Turn off time is presumed to be 0.

buf #(1,2,3) gate3 (out3, enable, in5);
When three delays are specified, the first value represents t_rise, the second value represents t_fall and the last value the turn off time.

Min, typ and max values

The general syntax for min, typ and max delay modelling is;
gate_type #(t_rise_min:t_ris_typ:t_rise_max, t_fall_min:t_fall_typ:t_fall_max, t_off_min:t_off_typ:t_off_max) gate_name (paramteters);

Similar rules apply for th especifying order as above. If only one t_rise value is specified then this value is applied to min, typ and max. If specifying more than one number, then all 3 MUST be scpecified. It is incorrect to specify two values as the compiler does not know which of the parameters the value represents.

An example of specifying two delays;
and #(1:2:3, 4:5:6) gate1 (out1, in1, in2);
This shows all values necessary for rise and fall times and gives values for min, typ and max for both delay types.

Another acceptable alternative would be;
or #(6:3:9, 5) gate2 (out2, in3, in4);
Here, 5 represents min, typ and max for the fall time.

N.B. T_off is only applicable to tri-state logic devices, it does not apply to primitive logic gates because they cannot be turned off.

237.With a specify block how to defining pin-to-pin delays for the module ?

```
 module A( q, a, b, c, d )
input a, b, c, d;
output q;
wire e, f;
// specify block containing delay statements
specify
( a => q ) = 6;   // delay from a to q
( b => q ) = 7;   // delay from b to q
( c => q ) = 7;   // delay form c to q
( d => q ) = 6;   // delay from d to q
endspecify
// module definition
or o1( e, a, b );
or o2( f, c, d );
exor ex1( q, e, f );
endmodule
 module A( q, a, b, c, d )
input a, b, c, d;
```

```
output q;
wire e, f;
// specify block containing full connection statements
specify
( a, d *> q ) = 6;     // delay from a and d to q
( b, c *> q ) = 7;     // delay from b and c to q
endspecify
// module definition
or o1( e, a, b );
or o2( f, c, d );
exor ex1( q, e, f );
endmodule
```

238.What are conditional path delays?

Conditional path delays, sometimes called *state dependent path delays*, are used to model delays which are dependent on the values of the signals in the circuit. This type of delay is expressed with an **if** conditional statement. The operands can be scalar or vector module input or inout ports, locally defined registers or nets, compile time constants (constant numbers or specify block parameters), or any bit-select or part-select of these. The conditional statement can contain any bitwise, logical, concatenation, conditional, or reduction operator.
The **else** construct cannot be used.

```
//Conditional path delays
Module A( q, a, b, c, d );
output q;
input a, b, c, d;
wire e, f;
// specify block with conditional timing statements
specify
// different timing set by level of input a
if (a) ( a => q ) = 12;
if ~(a) ( a => q ) = 14;
// delay conditional on b and c
// if b & c is true then delay is 7 else delay is 9
if ( b & c ) ( b => q ) = 7;
if ( ~( b & c )) ( b => q ) = 9;
// using the concatenation operator and full connections
```

```
if ( {c, d} = 2'b10 ) ( c, d *> q ) = 15;
if ( {c, d} != 2'b10 ) ( c, d *> q ) = 12;
endspecify
or o1( e, a, b );
or o2( f, c, d );
exor ex1( q, e, f );
endmodule
```

239.Tell something about Rise, fall, and turn-off delays?

Timing delays between pins can be expressed in greater detail by specifying rise, fall, and turn-off delay values. One, two, three, six, or twelve delay values can be specified for any path. The order in which the delay values are specified must be strictly followed.

```
// One delay used for all transitions
specparam delay = 15;
( a => q ) = delay;
// Two delays gives rise and fall times
specparam rise = 10, fall = 11;
( a => q ) = ( rise, fall );
// Three delays gives rise, fall and turn-off
// rise is used for 0-1, and z-1, fall for 1-0, and z-0, and turn-off for 0-z, and 1-z.
specparam rise = 10, fall = 11, toff = 8;
( a => q ) = ( rise, fall, toff );
// Six delays specifies transitions 0-1, 1-0, 0-z, z-1, 1-z, z-0
// strictly in that order
specparam t01 = 8, t10 = 9, t0z = 10, tz1 = 11, t1z = 12, tz0 = 13;
( a => q ) = ( t01, t10, t0z, tz1, t1z, tz0 );
// Twelve delays specifies transitions:
// 0-1, 1-0, 0-z, z-1, 1-z, z-0, 0-x, x-1, 1-x, x-0, x-z, z-x
// again strictly in that order
specparam t01 = 8, t10 = 9, t0z = 10, tz1 = 11, t1z = 12, tz0 = 13;
specparam t0x = 11, tx1 = 14, t1x = 12, tx0 = 10, txz = 8, tzx = 9;
( a => q ) = ( t01, t10, t0z, tz1, t1z, tz0, t0x, tx1, t1x, tx0, txz, tzx );
```

7)Tell me about In verilog delay modeling?

Distributed Delay

Distributed delay is delay assigned to each gate in a module. An example circuit is shown below.

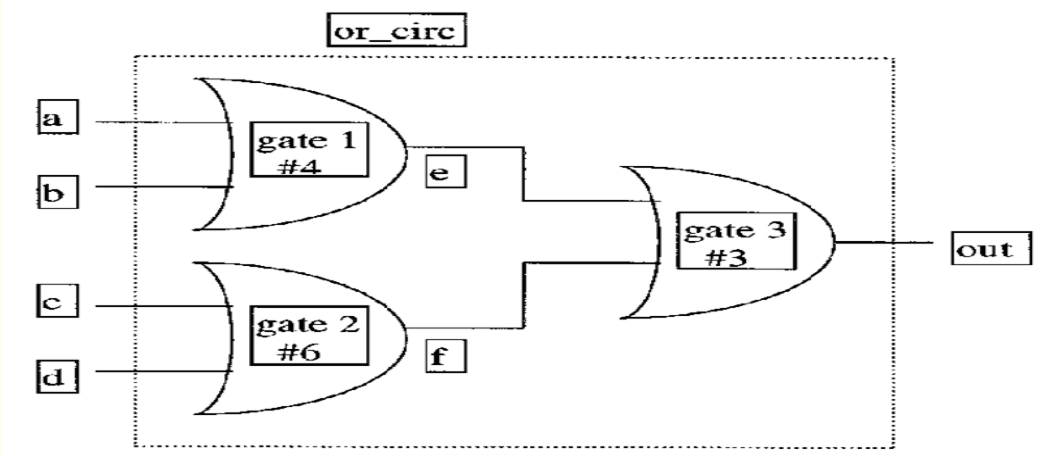


Figure 1: Distributed delay

As can be seen from Figure 1, each of the or-gates in the circuit above has a delay assigned to it:

- gate 1 has a delay of 4
- gate 2 has a delay of 6
- gate 3 has a delay of 3

When the input of any gate change, the output of the gate changes after the delay value specified.

The gate function and delay, for example for gate 1, can be described in the following manner:

or #4 a1 (e, a, b);

A delay of 4 is assigned to the or-gate. This means that the output of the gate, e, is delayed by 4 from the inputs a and b.

The module explaining Figure 1 can be of two forms:

1)

Module or_circ (out, a, b, c, d);

output out;

input a, b, c, d;

wire e, f;

//Delay distributed to each gate

or #4 a1 (e, a, b);

or #6 a2 (f, c, d);

or #3 a3 (out, e, f);

endmodule

2)

Module or_circ (out, a, b, c, d);

output out;

input a, b, c, d;

wire e, f;

//Delay distributed to each expression

assign #4 e = a & b;

assign #6 e = c & d;

assign #3 e = e & f;

endmodule

Version 1 models the circuit by assigning delay values to individual gates, while version 2 use delay values in individual assign statements. (An assign statement allows us to describe a combinational logic function without regard to its actual structural implementation. This means that the assign statement does not contain any modules with port connections). The above or_circ modules results in delays of (4+3) = 7 and (6+3) = 9 for the 4 connections part from the input to the output of the circuit.

Lumped Delay

Lumped delay is delay assigned as a single delay in each module, mostly to the output gate of the module. The cumulative delay of all paths is lumped at one location. The figure below is an example of lumped delay. This figure is similar as the figure of the distributed delay, but with the sum delay of the longest path assigned to the output gate: (delay of gate 2 + delay of gate 3) = 9.

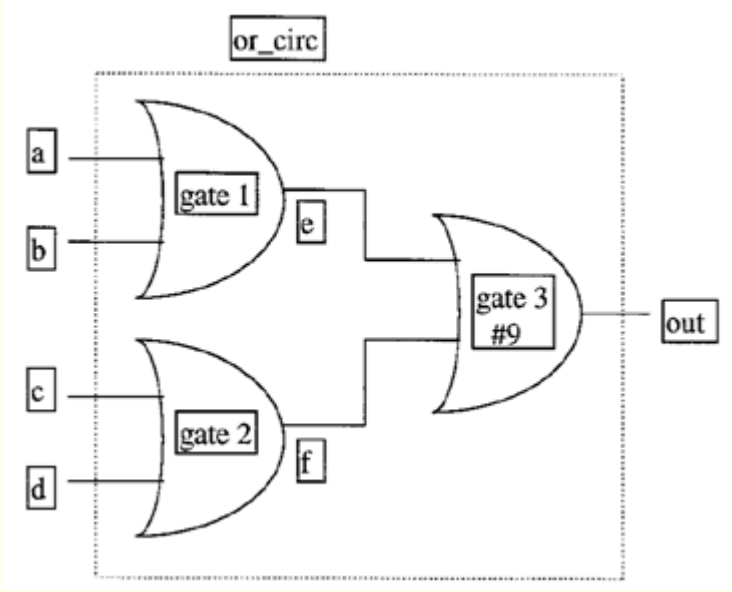


Figure 2: Lumped delay

As can be seen from Figure 2, gate 3 has got a delay of 9. When the input of this gate changes, the output of the gate changes after the delay value specified.

The program corresponding to Figure 2, is very similar to the one for distributed delay. The difference is that only or - gate 3 has got a delay assigned to it:

1)

Module or_circ (out, a, b, c, d);

output out;

input a, b, c, d;

wire e, f;

or a1 (e, a, b);

or a2 (f, c, d);

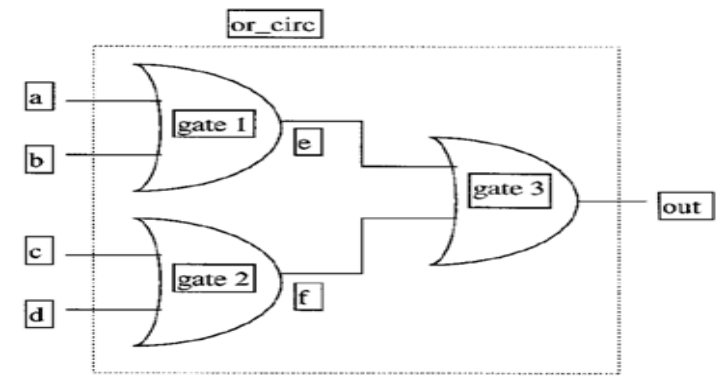
or #9 a3 (out, e, f); //delay only on the output gate

endmodule

This model can be used if delay between different inputs is not required.

Pin - to Pin Delay

Pin - to - Pin delay, also called path delay, is delay assigned to paths from each input to each output. An example circuit is shown below.



path a - e - out, delay = 7

path b - e - out, delay = 7

path c - f - out, delay = 9

path d - f - out, delay = 9

Figure 3: Pin - to Pin delay

The total delay from each input to each output is given. The same example circuit as for the distributed and lumped delay model is used. This means that the sum delay from each input to each output is the same.

The module for the above circuit is shown beneath:

Module or_circ (out, a, b, c, d);

output out;

input a, b, c, d;

wire e, f;

//Blocks specified with path delay

specify

(a => out) = 7;

(b => out) = 7;

(c => out) = 9;

(d => out) = 9;

endspecify

//gate calculations

or a1(e, a, b);

or a2(f, c, d);

or a3(out, e, f);

endmodule

Path delays of a module are specified incide a specify block, as seen from the example above. An example of delay from the input, a, to the output, out, is written as (a => out) = delay, where delay sets the delay between the two ports. The gate calculations are done after the path delays are defined.

For larger circuits, the pin - to - pin delay can be easier to model than distributed delay. This is because the designer writing delay models, needs to know only the input / output pins of the module, rather than the internals of the module. The path delays for digital circuits can be found through different simulation programs, for instance SPICE. Pin - to - Pin delays for standard parts can be found from data books. By using the path delay model, the program speed will increase.

240.Tell something about delay modeling timing checks?

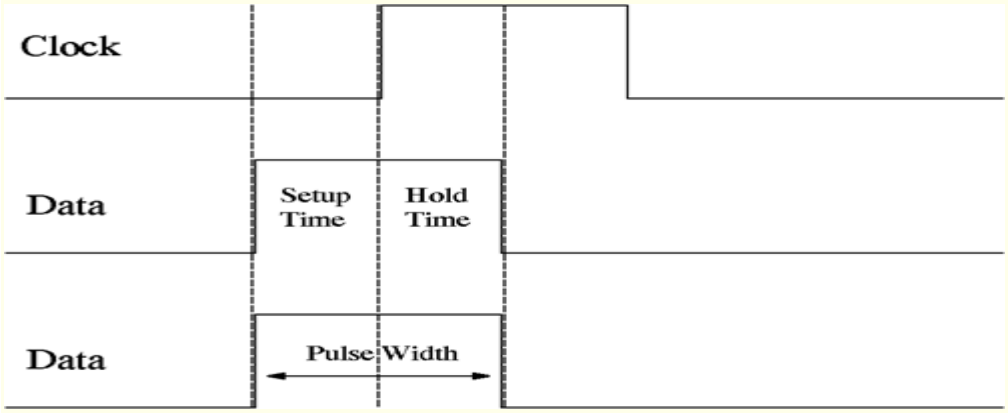
Delay Modeling: Timing Checks.

Keywords: $setup, $hold, $width

This section, the final part of the delay modeling chapter, discusses some of the various system tasks that exist for the purposes of timing checks. Verilog contains many timing-check system tasks, but only the three most common tasks are discussed here: **$setup**, **$hold** and **$width**. Timing checks are used to verify that timing constraints are upheld, and are especially important in the simulation of high-speed sequential circuits such as microprocessors. All timing checks must be contained within **specify** blocks as shown in the example below.

The **$setup** and **$hold** tasks are used to monitor the *setup* and *hold* constraints during the simulation of a sequential circuit element. In the example, the setup time is the minimum allowed time between a change in the input *d* and a positive clock edge. Similarly, the hold time is the minimum allowed time between a positive clock edge and a change in the input *d*.

The **$width** task is used to check the minimum width of a positive or negative-going pulse. In the example, this is the time between a negative transition and the transition back to 1.



Syntax:

NB: *data_change*, *reference* and *reference1* must be declared *wire*s.

$setup(*data_change*, *reference*, *time_limit*);

data_change: signal that is checked against the *reference*

reference: signal used as reference

time_limit: minimum time required between the two events.

Violation if: Treference - Tdata_change < time_limit.

$hold(*reference*, *data_change*, *time_limit*);

reference: signal used as reference

data_change: signal that is checked against the *reference*

time_limit: minimum time required between the two events.

Violation if: Tdata_change - Treference < time_limit

$width(*reference1*, *time_limit*);

reference1: first transition of signal

time_limit: minimum time required between *transition1* and *transition2*.

Violation if: Treference2 - Treference1 < time_limit

Example:

```
module d_type(q, clk, d);
  output q;
  input  clk, d;

  reg    q;

  always @(posedge clk)
```

```
    q = d;

endmodule // d_type

module stimulus;

  reg  clk, d;
  wire q, clk2, d2;
   d_type dt_test(q, clk, d);

  assign d2=d;
  assign clk2=clk;

  initial
    begin
      $display ("\t\t     clock d q");
      $display ($time,"   %b   %b %b", clk, d, q);
      clk=0;
      d=1;
      #7 d=0;
      #7 d=1; // causes setup violation
      #3 d=0;
      #5 d=1; // causes hold violation
      #2 d=0;
      #1 d=1; // causes width violation
    end // initial begin

  initial
    #26 $finish;

  always
    #3 clk = ~clk;
    always
```

```
    #1 $display ($time,"   %b   %b %b", clk, d, q);
  specify
    $setup(d2, posedge clk2, 2);
    $hold(posedge clk2, d2, 2);
    $width(negedge d2, 2);
  endspecify
endmodule // stimulus
```

Output:

```
                clock d q
               0   x   x x
               1   0   1 x
               2   0   1 x
               3   1   1 x
               4   1   1 1
               5   1   1 1
               6   0   1 1
               7   0   0 1
               8   0   0 1
               9   1   0 1
              10   1   0 0
              11   1   0 0
              12   0   0 0
              13   0   0 0
              14   0   1 0
              15   1   1 0

"timechecks.v", 46: Timing violation in stimulus
   $setup( d2:14, posedge clk2:15, 2 );
               16   1   1 1
              17   1   0 1
              18   0   0 1
              19   0   0 1
              20   0   0 1
```

21 1 0 1

22 1 1 0

"timechecks.v", 47: Timing violation in stimulus

$hold(posedge clk2:21, d2:22, 2);

23 1 1 0

24 0 0 0

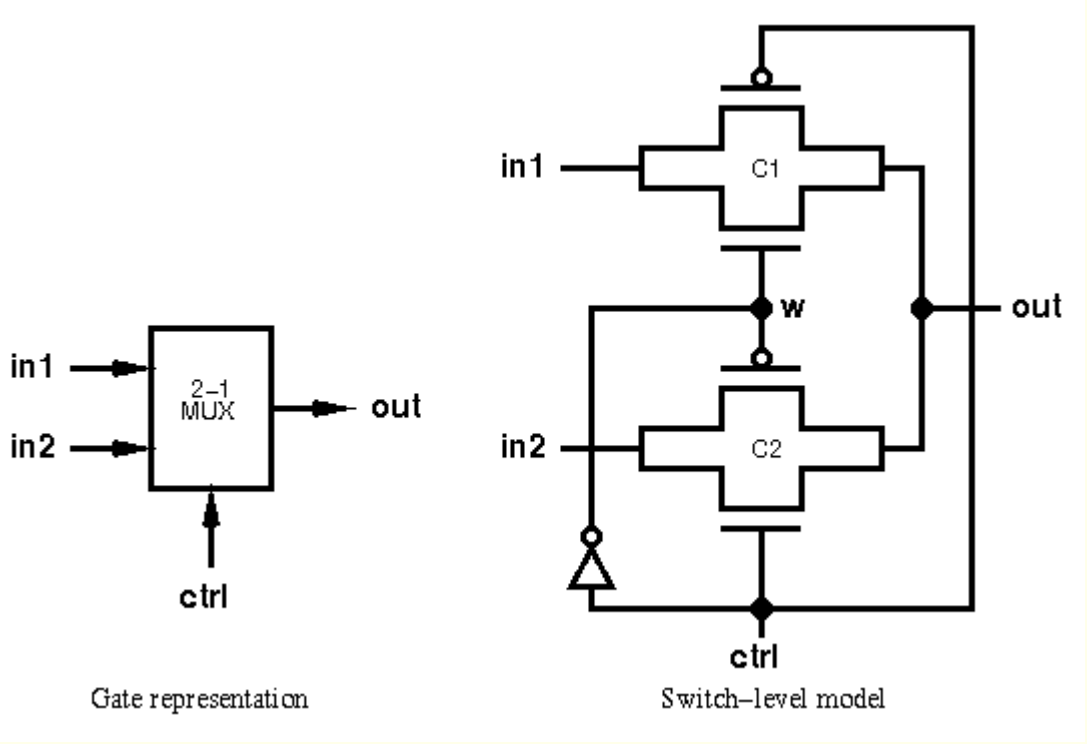
25 0 1 0

"timechecks.v", 48: Timing violation in stimulus

$width(negedge d2:24, : 25, 2);

241.Draw a 2:1 mux using switches and verilog code for it?

1-bit 2-1 Multiplexer



This circuit assigns the output **out** to either inputs **in1** or **in2** depending on the low or high values of **ctrl** respectively.

// Switch-level description of a 1-bit 2-1 multiplexer

// ctrl=0, out=in1; ctrl=1, out=in2

```
module mux21_sw (out, ctrl, in1, in2);

  output out;                 // mux output
  input  ctrl, in1, in2;      // mux inputs
  wire       w;               // internal wire

  inv_sw I1 (w, ctrl);        // instantiate inverter module

  cmos C1 (out, in1, w, ctrl);  // instantiate cmos switches
  cmos C2 (out, in2, ctrl, w);

endmodule
```

An inverter is required in the multiplexer circuit, which is instantiated from the previously defined module.

Two transmission gates, of instance names C1 and C2, are implemented with the *cmos* statement, in the format cmos [instancename]([output],[input],[nmosgate],[pmosgate]). Again, the instance name is optional.

242.What are the synthesizable gate level constructs?

The <GATETYPE> Keywords				
and	buf	nmos	tran	pullup
nand	not	pmos	tranif0	pulldown
nor	bufif0	cmos	tranif1	
or	bufif1	rnmos	rtran	
xor	notif0	rpmos	rtranif0	
xnor	notif1	rcmos	rtranif1	

The above table gives all the gate level constructs of only the constructs in first two columns are synthesizable.

243.Explain about setup time and hold time, what will happen if there is setup time and hold tine violation, how to overcome this?

Set up time is the amount of time before the clock edge that the input signal needs to be stable to guarantee it is accepted properly on the clock edge.

Hold time is the amount of time after the clock edge that same input signal has to be held

before changing it to make sure it is sensed properly at the clock edge.
Whenever there are setup and hold time violations in any flip-flop, it enters a state where its output is unpredictable: this state is known as metastable state (quasi stable state); at the end of metastable state, the flip-flop settles down to either '1' or '0'. This whole process is known as metastability

244. What is skew, what are problems associated with it and how to minimize it?

In circuit design, clock skew is a phenomenon in synchronous circuits in which the clock signal (sent from the clock circuit) arrives at different components at different times.
This is typically due to two causes. The first is a material flaw, which causes a signal to travel faster or slower than expected. The second is distance: if the signal has to travel the entire length of a circuit, it will likely (depending on the circuit's size) arrive at different parts of the circuit at different times. Clock skew can cause harm in two ways. Suppose that a logic path travels through combinational logic from a source flip-flop to a destination flip-flop. If the destination flip-flop receives the clock tick later than the source flip-flop, and if the logic path delay is short enough, then the data signal might arrive at the destination flip-flop before the clock tick, destroying there the previous data that should have been clocked through. This is called a hold violation because the previous data is not held long enough at the destination flip-flop to be properly clocked through. If the destination flip-flop receives the clock tick earlier than the source flip-flop, then the data signal has that much less time to reach the destination flip-flop before the next clock tick. If it fails to do so, a setup violation occurs, so-called because the new data was not set up and stable before the next clock tick arrived. A hold violation is more serious than a setup violation because it cannot be fixed by increasing the clock period.
Clock skew, if done right, can also benefit a circuit. It can be intentionally introduced to decrease the clock period at which the circuit will operate correctly, and/or to increase the setup or hold safety margins. The optimal set of clock delays is determined by a linear program, in which a setup and a hold constraint appears for each logic path. In this linear program, zero clock skew is merely a feasible point.
Clock skew can be minimized by proper routing of clock signal (clock distribution tree) or putting variable delay buffer so that all clock inputs arrive at the same time

245.What is slack?

'Slack' is the amount of time you have that is measured from when an event 'actually happens' and when it 'must happen'.. The term 'actually happens' can also be taken as being a predicted time for when the event will 'actually happen'.
When something 'must happen' can also be called a 'deadline' so another definition of slack would be the time from when something 'actually happens' (call this Tact) until the deadline (call this Tdead).
Slack = Tdead - Tact.
Negative slack implies that the 'actually happen' time is later than the 'deadline' time...in other words it's too late and a timing violation....you have a timing problem that needs some attention.

246. What is glitch? What causes it (explain with waveform)? How to overcome it?

The following figure shows a gated clock. The gated clock's corresponding timing diagram shows that this implementation can lead to clock glitches, which can cause the flip-flop to clock at the wrong time.

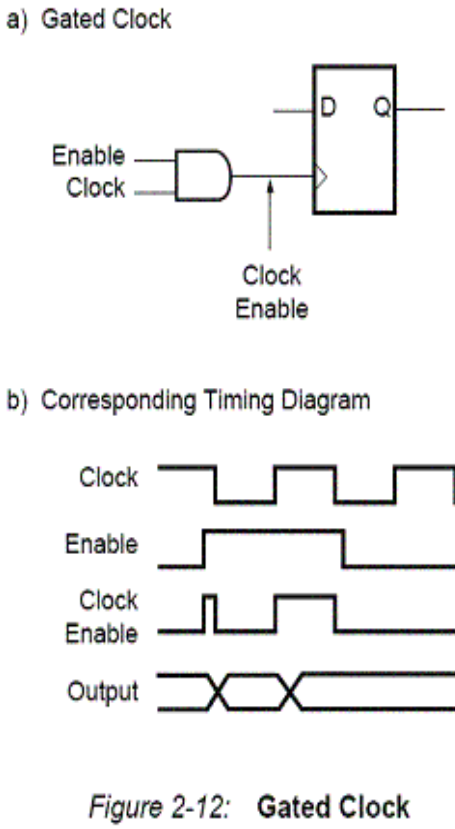


Figure 2-12: **Gated Clock**

The following figure shows a synchronous alternative to the gated clock using a data path. The flip-flop is clocked at every clock cycle and the data path is controlled by an enable. When the enable is Low, the multiplexer feeds the output of the register back on itself. When the enable is High, new data is fed to the flip-flop and the register changes its state

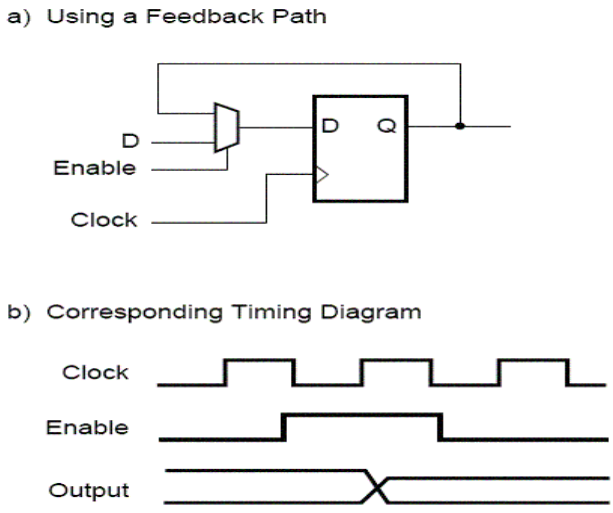


Figure 2-13: **Synchronous Design Using Data Feedback**

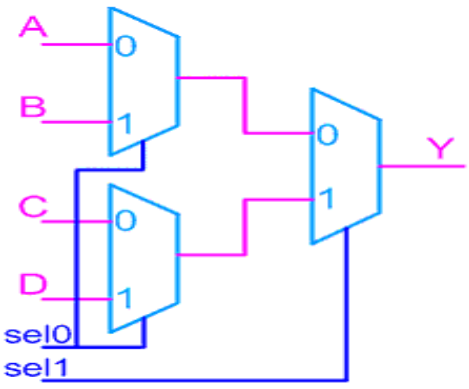
247.Given only two xor gates one must function as buffer and another as inverter?

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.

248. What is difference between latch and flipflop?

The main difference between latch and FF is that latches are level sensitive while FF are 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.

249.Build a 4:1 mux using only 2:1 mux?



250.Difference between heap and stack?

The Stack is more or less responsible for keeping track of what's executing in our code (or what's been "called"). The Heap is more or less responsible for keeping track of our objects (our data, well... most of it - we'll get to that later.).

Think of the Stack as a series of boxes stacked one on top of the next. We keep track of what's going on in our application by stacking another box on top every time we call a method (called a Frame). We can only use what's in the top box on the stack. When we're done with the top box (the method is done executing) we throw it away and proceed to use the stuff in the previous box on the top of the stack. The Heap is similar except that its purpose is to hold information (not keep track of execution most of the time) so anything in our Heap can be accessed at any time. With the Heap, there are no constraints as to what can be accessed like in the stack. The Heap is like the heap of clean laundry on our bed that we have not taken the time to put away yet - we can grab what we need quickly. The Stack is like the stack of shoe boxes in the closet where we have to take off the top one to get to the one underneath it.

251.Difference between mealy and moore state machine?

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.

Adv and Disadv

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.

252.Difference between onehot and binary encoding?

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.

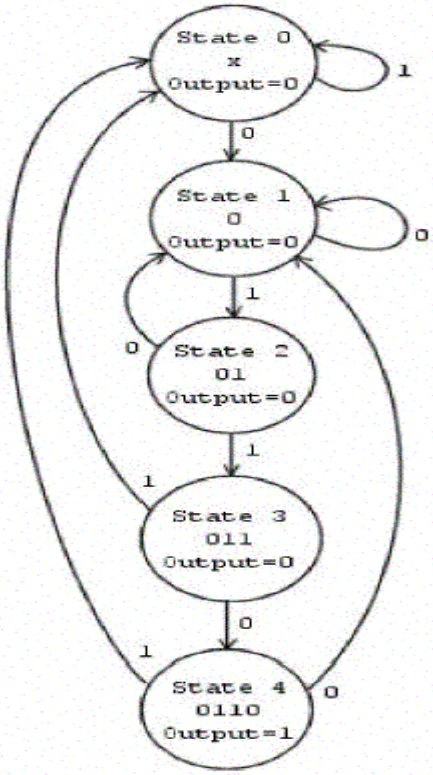
A onehot FSM design requires a flip-flop for each state in the design and only one flip-flop (the flip-flop representing the current or "hot" state) is set at a time in a one hot FSM design. For a state machine with 9- 16 states, a binary FSM only requires 4 flip-flops while a onehot FSM requires a flip-flop for each state in the design
FPGA vendors frequently recommend using a onehot state encoding style because flip-flops are plentiful in an FPGA and the combinational logic required to implement a onehot 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, onehot FSMs typically run faster than a binary encoded FSM with larger combinational logic blocks

253. How to calculate maximum operating frequency?

254.How to find out longest path?

You can find answer to this in timing.ppt of presentations section on this site

255.Draw the state diagram to output a "1" for one cycle if the sequence "0110" shows up (the leading 0s cannot be used in more than one sequence)?



256.How to achieve 180 degree exact phase shift?

Never tell using inverter
a) dcm's 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.

257.What is significance of ras and cas in SDRAM?

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.

258.Tell some of applications of buffer?

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 fanout,eg:bufg

259.Implement an AND gate using mux?

This is the basic question that many interviewers ask. for and gate, give one input as select line,incase if u r giving b as select line, connect one input to logic '0' and other input to a.

260.What will happen if contents of register are shifter left, right?

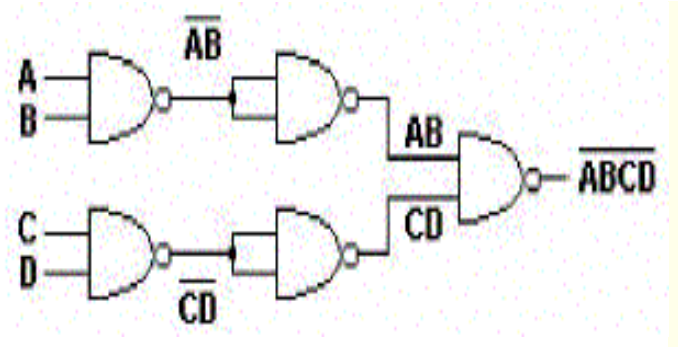
It is well known that in left shift all bits will be shifted left and LSB will be appended with 0 and in right shift all bits will be shifted right and MSB will be appended with 0 this is a straightforward answer
What is expected is in a left shift value gets Multiplied by 2 eg:consider 0000_1110=14 a left shift will make it 0001_110=28, it the same fashion right shift will Divide the value by 2.

261.Given the following FIFO and rules, how deep does the FIFO need to be to prevent underflow or overflow?

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.

262. Design a four-input NAND gate using only two-input NAND gates ?
A:Basically, you can tie the inputs of a NAND gate together to get an inverter, so...



263.Difference between Synchronous and Asynchronous reset.?

Synchronous reset logic will synthesize to smaller flip-flops, particularly if the reset is gated with the logic generating the d-input. 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 clock.

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, can not 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.

264.Why are most interrupts active low?

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.

265.Give two ways of converting a two input NAND gate to an inverter?

(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.

266.What are set up time & hold time constraints? What do they signify? Which one is critical for estimating maximum clock frequency of a circuit?

set up time: - 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.

267.Differences between D-Latch and D flip-flop?
D-latch is level sensitive where as flip-flop is edge sensitive. Flip-flops are made up of latches.

268.What is a multiplexer?
Is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. (2n =>n).

269.How can you convert an SR Flip-flop to a JK Flip-flop?

By giving the feed back we can convert, i.e !Q=>S and Q=>R.Hence the S and R inputs will act as J and K respectively.

270.How can you convert the JK Flip-flop to a D Flip-flop?

By connecting the J input to the K through the inverter.

271.What is Race-around problem?How can you rectify it?

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.

272.How do you detect if two 8-bit signals are same?

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 are then given as i/p to an 8-i/p nor gate. if o/p is 1 then A=B.

273.7 bit ring counter's initial state is 0100010. After how many clock cycles will it return to the initial state?

6 cycles

274.Convert D-FF into divide by 2. (not latch) What is the max clock frequency the circuit can handle, given the following information?

T_setup= 6nS T_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

275.Guys this is the basic question asked most frequently. Design all the basic gates(NOT,AND,OR,NAND,NOR,XOR,XNOR) using 2:1 Multiplexer?

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

276.N number of XNOR gates are 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?

If N=Odd, the circuit acts as even parity detector, ie 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.

277.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:

(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.
278.Design a circuit that calculates the square of a number? It should not use any multiplier circuits. It should use Multiplexers and other logic?
This is interesting....
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
and so on
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.

279.How will you implement a Full subtractor from a Full adder?

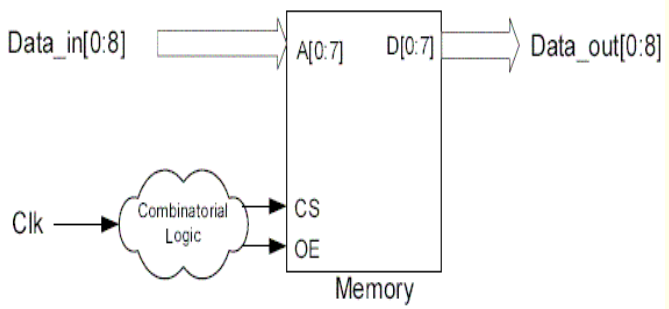
all the bits of subtrahend should be connected to the xor gate. Other input to the xor being one.The input carry bit to the full adder should be made 1. Then the full adder works like a full subtractor

280.A very good interview question... 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...?

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 does not basically depend on clock frequency.
281.In a 3-bit Johnson's counter what are the unused states?

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

282.The question is to design minimal hardware system, which encrypts 8-bit parallel data. A synchronized clock is provided to this system as well. The output encrypted data should be at the same rate as the input data but no necessarily with the same phase?



The encryption system is centered around a memory device that perform a LUT (Look-Up

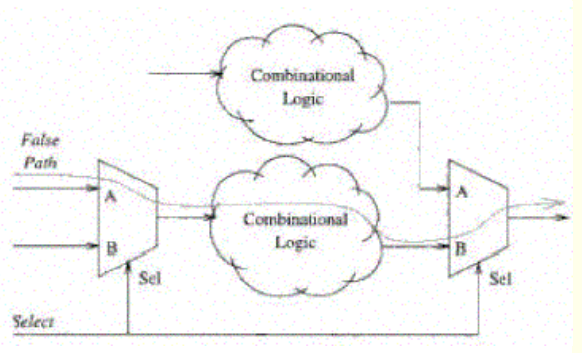
Table) conversion. This memory functionality can be achieved by using a PROM, EPROM, FLASH and etc. The device contains an encryption code, which may be burned into the device with an external programmer. In encryption operation, the data_in is an address pointer into a memory cell and the combinatorial logic generates the control signals. This creates a read access from the memory. Then the memory device goes to the appropriate address and outputs the associate data. This data represent the data_in after encryption.

283.What is an LFSR .List a few of its industry applications.?

LFSR is a linear feedback shift register where the input bit is driven by a linear function of the overall shift register value. coming to industrial applications, as far as I know, it is used for encryption and decryption and in BIST(built-in-self-test) based applications..

284.what is false path?how it determine in ckt? what the effect of false path in ckt?

By timing all the paths in the circuit the timing analyzer can determine all the critical paths in the circuit. However, the circuit may have false paths, which are the paths in the circuit which are never exercised during normal circuit operation for any set of inputs.
An example of a false path is shown in figure below. The path going from the input A of the first MUX through the combinational logic out through the B input of the second MUS is a false path. This path can never be activated since if the A input of the first MUX is activated, then Sel line will also select the A input of the second MUX.
STA (Static Timing Analysis) tools are able to identify simple false paths; however they are not able to identify all the false paths and sometimes report false paths as critical paths. Removal of false paths makes circuit testable and its timing performance predictable (sometimes faster)



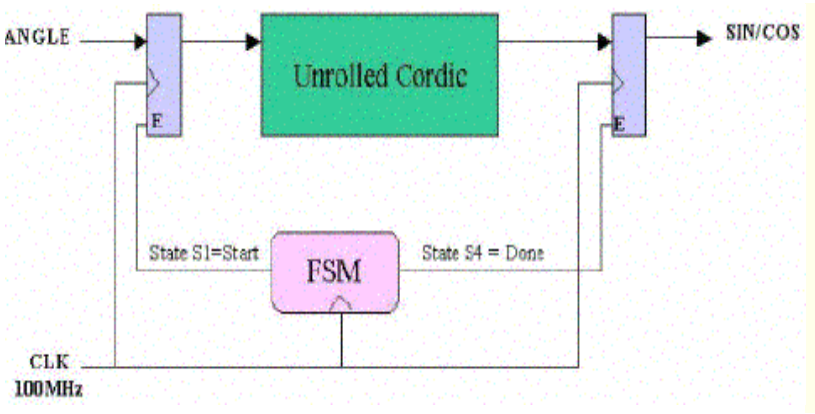
285.Consider two similar processors, one with a clock skew of 100ps and other with a clock skew of 50ps. Which one is likely to have more power? Why?

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.

286.What are multi-cycle paths?

Multi-cycle paths are paths between registers that take more than one clock cycle to become

stable.
For ex. Analyzing the design shown in fig below shows that the output SIN/COS requires 4 clock-cycles after the input ANGLE is latched in. This means that the combinatorial block (the Unrolled Cordic) can take up to 4 clock periods (25MHz) to propagate its result. Place and Route tools are capable of fixing multi-cycle paths problem.



287.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?

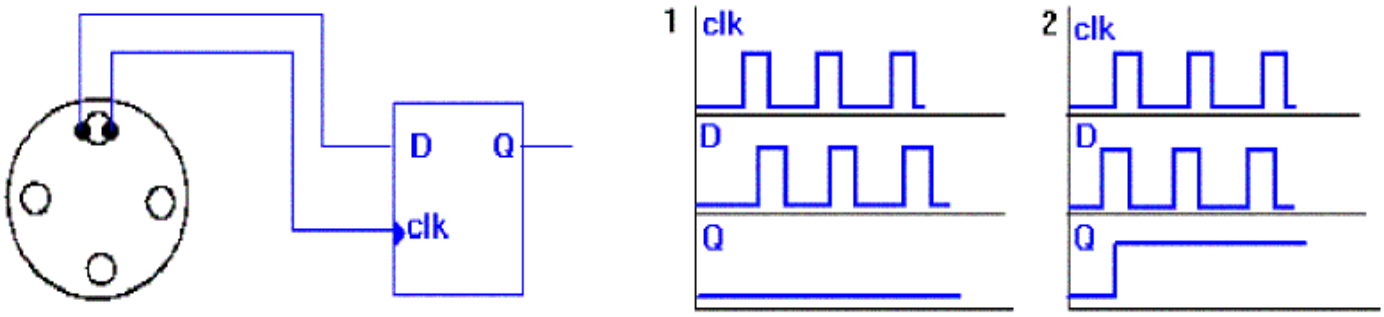
The synchronous counter will have lesser delay as the input to each flop is readily available before the clock edge. Whereas the cascade counter will take long time as the output of one flop is used as clock to the other. So the delay will be propagating. For Eg: 16 state counter = 4 bit counter = 4 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)

288.what is difference between RAM and FIFO?

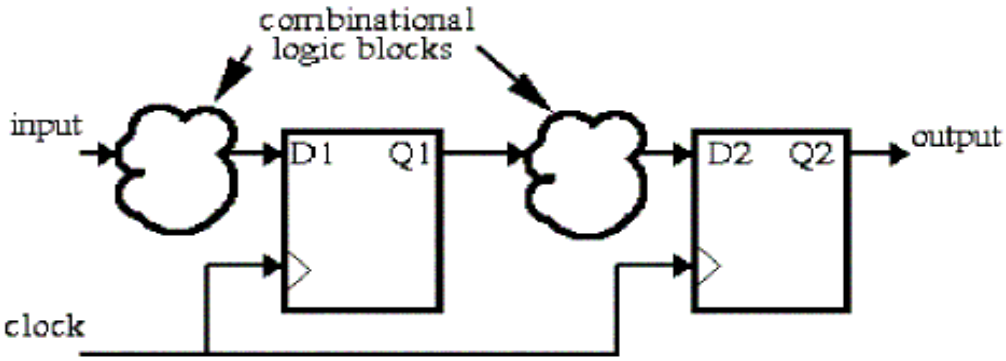
FIFO does not have address lines
Ram is used for storage purpose where as fifo is used for synchronization purpose i.e. when two peripherals are working in different clock domains then we will go for fifo.

289.The circle can rotate clockwise and back. Use minimum hardware to build a circuit to indicate the direction of rotating.?

2 sensors are required to find out the direction of rotating. They are placed like at the drawing. One of them 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.



290.Draw timing diagrams for following circuit.?



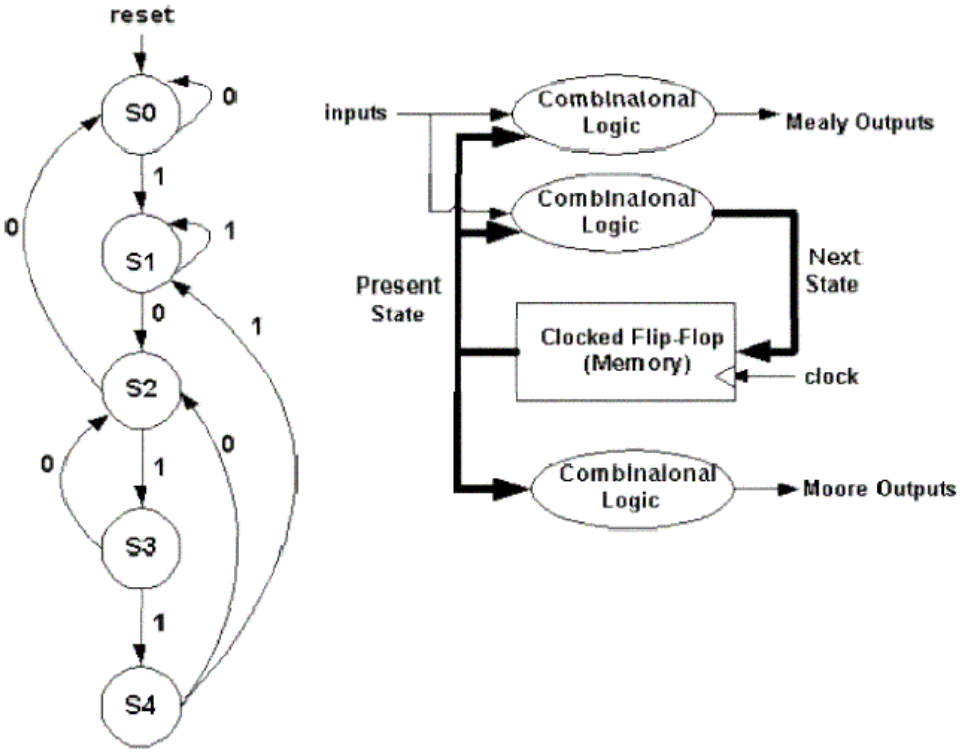
291.Implement the following circuits:
(a) 3 input NAND gate using min no of 2 input NAND Gates
(b) 3 input NOR gate using min no of 2 inpur NOR Gates
(c) 3 input XNOR gate using min no of 2 inpur XNOR Gates
Assuming 3 inputs A,B,C?

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)
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.
3 input NOR:
Same as above just interchange NAND with NOR ((A NOR B) NOR (A NOR B)) NOR C
3 input XNOR:
Same as above except the inputs for the second XNOR gate, Output of the first XNOR gate is one of the inputs and connect the second input to ground or logical '0'
((A XNOR B) XNOR 0)) XNOR C

292.Is it possible to reduce clock skew to zero? Explain your 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).

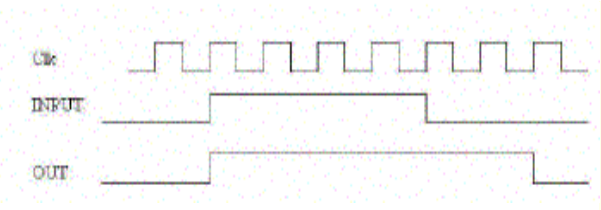
293.Design a FSM (Finite State Machine) to detect a sequence 10110?



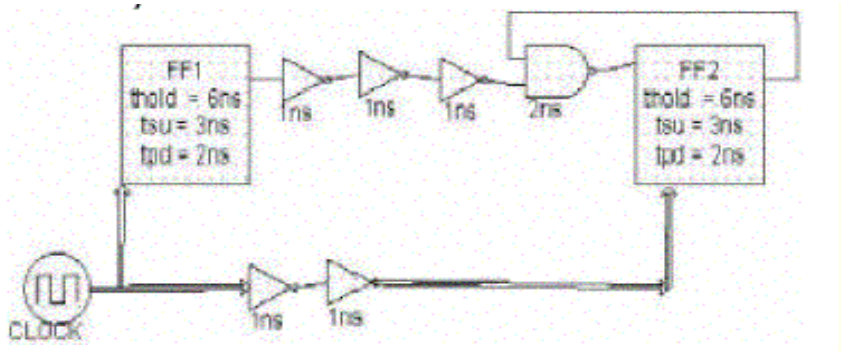
294.Convert D-FF into divide by 2. (not latch)? What is the max clock frequency of the circuit , given the following information?
T_setup= 6nS
T_hold = 2nS
T_propagation = 10nS

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

295.Give the circuit to extend the falling edge of the input by 2 clock pulses?The waveforms are shown in the following figure.

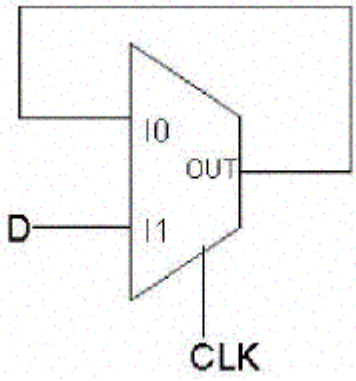


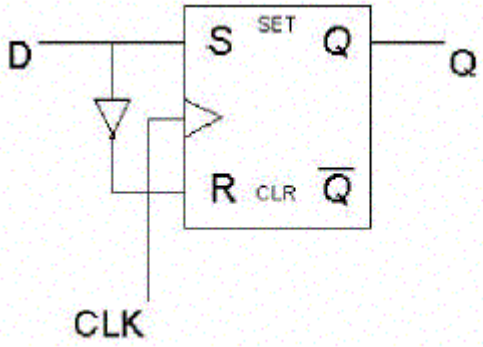
296.For the Circuit Shown below, What is the Maximum Frequency of Operation?Are there any hold time violations for FF2? If yes, how do you modify the circuit to avoid them?



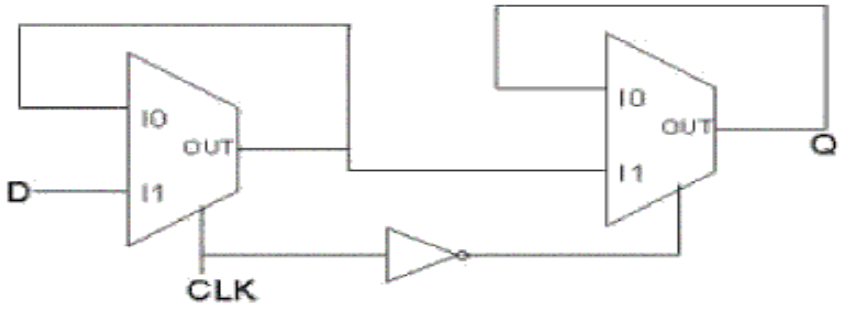
The minumum time period = 3+2+(1+1+1) = 8ns Maximum Frequency = 1/8n= 125MHz. And there is a hold time violation in the circuit,because of feedback, if you observe, tcq2+AND gate delay is less than thold2,To avoid this we need to use even number of inverters(buffers). Here we need to use 2 inverters each with a delay of 1ns. then the hold time value exactly meets.

297.Design a D-latch using (a) using 2:1 Mux (b) from S-R Latch ?





298.How to implement a Master Slave flip flop using a 2 to 1 mux?



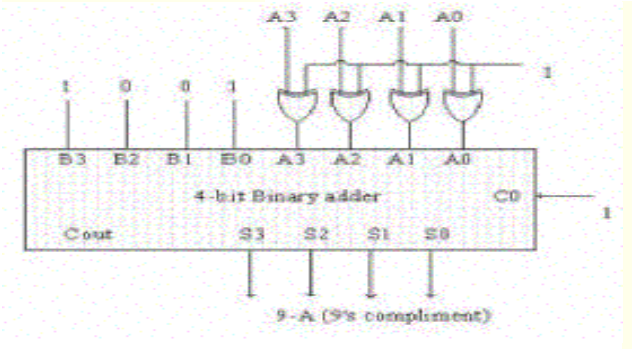
299.how many 2 input xor's are needed to inplement 16 input parity generator ?

It is always n-1 Where n is number of inputs.So 16 input parity generator will require 15 two input xor's .

300.Design a circuit for finding the 9's compliment of a BCD number using 4-bit binary adder and some external logic gates?

9's compliment is nothing but subracting the given no from 9.So using a 4 bit binary adder we can just subract the given binary no from 1001(i.e. 9).Here we can use the 2's compliment

method addition.



301.what is Difference between writeback and write through cache?

A caching method in which modifications to data in the cache aren't copied to the cache source until absolutely necessary. Write-back caching is available on many microprocessors , including all Intel processors since the 80486. With these microprocessors, data modifications to data stored in the L1 cache aren't copied to main memory until absolutely necessary. In contrast, a write-through cache performs all write operations in parallel -- data is written to main memory and the L1 cache simultaneously. Write-back caching yields somewhat better performance than write-through caching because it reduces the number of write operations to main memory. With this performance improvement comes a slight risk that data may be lost if the system crashes.
A write-back cache is also called a copy-back cache.

302.Difference between Synchronous,Asynchronous & Isynchronous communication?

Sending data encoded into your signal requires that the sender and receiver are both using the same enconding/decoding method, and know where to look in the signal to find data. Asynchronous systems do not send separate information to indicate the encoding or clocking information. The receiver must decide the clocking of the signal on it's own. This means that the receiver must decide where to look in the signal stream to find ones and zeroes, and decide for itself where each individual bit stops and starts. This information is not in the data in the signal sent from transmitting unit.

Synchronous systems negotiate the connection at the data-link level before communication begins. Basic synchronous systems will synchronize two clocks before transmission, and reset their numeric counters for errors etc. More advanced systems may negotiate things like error correction and compression.

Time-dependent. it refers to processes where data must be delivered within certain time constraints. For example, Multimedia stream require an isochronous transport mechanism to ensure that data is delivered as fast as it is displayed and to ensure that the audio is synchronized with the video.

303. What are different ways Multiply & Divide?

304.What is a SoC (System On Chip), ASIC, “full custom chip”, and an FPGA?

There are no precise definitions. Here is my sense of it all. First, 15 years ago, people were unclear on exactly what VLSI meant. Was it 50000 gates? 100000 gates? was is just anything

bigger than LSI? My professor simply told me that; VLSI is a level of complexity and integration in a chip that demands Electronic Design Automation tools in order to succeed. In other words, big enough that manually drawing lots of little blue, red and green lines is too much for a human to reasonably do. I think that, likewise, SoC is that level of integration onto a chip that demands more expertise beyond traditional skills of electronics. In other words, pulling off a SoC demands Hardware, Software, and Systems Engineering talent. So, trivially, SoCs aggressively combine HW/SW on a single chip. Maybe more pragmatically, SoC just means that ASIC and Software folks are learning a little bit more about each other's techniques and tools than they did before. Two other interpretations of SoC are 1) a chip that integrates various IP (Intellectual Property) blocks on it and is thus highly centered with issues like Reuse, and 2) a chip integrating multiple classes of electronic circuitry such as Digital CMOS, mixed-signal digital and analog (e.g. sensors, modulators, A/Ds), DRAM memory, high voltage power, etc.

ASIC stands for "Application Specific Integrated Circuit". A chip designed for a specific application. Usually, I think people associate ASICs with the Standard Cell design methodology. Standard Cell design and the typical "ASIC flow" usually means that designers are using Hardware Description Languages, Synthesis and a library of primitive cells (e.g. libraries containing AND, NAND, OR, NOR, NOT, FLIP-FLOP, LATCH, ADDER, BUFFER, PAD cells that are wired together (real libraries are not this simple, but you get the idea..). Design usually is NOT done at a transistor level. There is a high reliance on automated tools because the assumption is that the chip is being made for a SPECIFIC APPLICATION where time is of the essence. But, the chip is manufactured from scratch in that no pre-made circuitry is being programmed or reused. ASIC designer may, or may not, even be aware of the locations of various pieces of circuitry on the chip since the tools do much of the construction, placement and wiring of all the little pieces.

Full Custom, in contrast to ASIC (or Standard Cell), means that every geometric feature going onto the chip being designed (think of those pretty chip pictures we have all seen) is controlled, more or less, by the human design. Automated tools are certainly used to wire up different parts of the circuit and maybe even manipulate (repeat, rotate, etc.) sections of the chip. But, the human designer is actively engaged with the physical features of the circuitry. Higher human crafting and less reliance on standard cells takes more time and implies higher NRE costs, but lowers RE costs for standard parts like memories, processors, uarts, etc. FPGAs, or Field Programmable Gate Arrays are completely designed chips that designers load a programming pattern into to achieve a specific digital function. A bit pattern (almost like a software program) is loaded into the already manufactured device which essentially interconnects lots of available gates to meet the designers purposes. FPGAs are sometimes thought of as a "Sea of Gates" where the designer specifies how they are connected. FPGA designers often use many of the same tools that ASIC designers use, even though the FPGA is inherently more flexible. All these things can be intermixed in hybrid sorts of ways. For example, FPGAs are now available that have microprocessor embedded within them which were designed in a full custom manner, all of which now demands "SoC" types of HW/SW integration skills from the designer.

305.What is "Scan" ?

Scan Insertion and ATPG helps test ASICs (e.g. chips) during manufacture. If you know what JTAG boundary scan is, then Scan is the same idea except that it is done inside the chip instead of on the entire board. Scan tests for defects in the chip's circuitry after it is manufactured (e.g. Scan does not help you test whether your Design *functions* as intended). ASIC designers usually implement the scan themselves and occurs just after synthesis. ATPG (Automated Test Pattern Generation) refers to the creation of "Test Vectors" that the Scan circuitry enables to be introduced into the chip. Here's a brief summary:

- Scan Insertion is done by a tool and results in all (or most) of your design's flip-flops to be replaced by special "Scan Flip-flops". Scan flops have additional inputs/outputs that allow them to be configured into a "chain" (e.g. a big shift register) when the chip is put into a test mode.
- The Scan flip-flops are connected up into a chain (perhaps multiple chains)
- The ATPG tool, which knows about the scan chain you've created, generates a series of test vectors.
- The ATPG test vectors include both "Stimulus" and "Expected" bit patterns. These bit vectors are shifted into the chip on the scan chains, and the chips reaction to the stimulus is shifted back out again.
- The ATE (Automated Test Equipment) at the chip factory can put the chip into the scan test mode, and apply the test vectors. If any vectors do not match, then the chip is defective and it is thrown away.
- Scan/ATPG tools will strive to maximize the "coverage" of the ATPG vectors. In other words, given some measure of the total number of nodes in the chip that could be faulty (shorted, grounded, "stuck at 1", "stuck at 0"), what percentage of them can be detected with the ATPG vectors? Scan is a good technology and can achive high coverage in the 90% range.
- Scan testing does not solve all test problems. Scan testing typically does not test memories (no flip-flops!), needs a gate-level netlist to work with, and can take a long time to run on the ATE.
- FPGA designers may be unfamiliar with scan since FPGA testing has already been done by the FPGA manufacturer. ASIC designers do not have this luxury and must handle all the manufacturing test details themselves.
- Check out the Synopsys WWW site for more info.

306. 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
```

307. 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.

308. 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
```

309. 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, ... );
```

310. 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."

311. 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.

312. 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.

313. Variable And Signal Which Will Be Updated First?

Answer :

Signals

314. 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.

315. 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.

316. 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
```

317. 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.

318. 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

319. 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
);

320. 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

321. 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.

322. 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

323. 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.

324. 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.

325. 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
```

326. 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.

327. 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.

328. 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.

329. 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.

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

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

330. 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.

331. 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
```

332. 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).

333. 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 (&&).

334. 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 Behavioural 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.

Verilog Interiew Quetions Collection :

- What is the difference between $display and $monitor and $write and $strobe?
- What is the difference between code-compiled simulator and normal simulator?
- What is the difference between wire and reg?
- What is the difference between blocking and non-blocking assignments?
- What is the significance Timescale directivbe?
- What is the difference between bit wise, unary and logical operators?
- What is the difference between task and function?
- What is the difference between casex, casez and case statements?
- Which one preferred-casex or casez?
- For what is defparam used?
- What is the difference between “= =” and “= = =” ?
- What is a compiler directive like ‘include’ and ‘ifdef’?
- Write a verilog code to swap contents of two registers with and without a temporary register?
- What is the difference between inter statement and intra statement delay?
- What is delta simulation time?
- What is difference between Verilog full case and parallel case?
- What you mean by inferring latches?
- How to avoid latches in your design?
- Why latches are not preferred in synthesized design?
- How blocking and non blocking statements get executed?
- Which will be updated first: is it variable or signal?

- What is sensitivity list?
- If you miss sensitivity list what happens?
- In a pure combinational circuit is it necessary to mention all the inputs in sensitivity disk? If yes, why? If not, why?
- In a pure sequential circuit is it necessary to mention all the inputs in sensitivity disk? If yes, why? If not, why?
- What is general structure of Verilog code you follow?
- What are the difference between Verilog and VHDL?
- What are system tasks?
- List some of system tasks and what are their purposes?
- What are the enhancements in Verilog 2001?
- Write a Verilog code for synchronous and asynchronous reset?
- What is pli? why is it used?
- What is file I/O?
- What is difference between freeze deposit and force?
- Will case always infer priority register? If yes how? Give an example.
- What are inertial and transport delays ?
- What does `timescale 1 ns/ 1 ps' signify in a verilog code?
- How to generate sine wav using verilog coding style?
- How do you implement the bi-directional ports in Verilog HDL?
- How to write FSM is verilog?
- What is verilog case (1)?
- What are Different types of Verilog simulators available?
- What is Constrained-Random Verification ?

How can you model a SRAM at RTL Level?http://www.asic.co.in/Index_files/verilog_interview_questions3.html