Verilog Assessment-1

1. What is the difference between reg and wire? (1)
   Ans:

reg	wire
reg can store value. It is used in procedural assignments	wire cannot store value. It is used in continuous assignments
default value is x	default value is z

2. What does timescale 1 ns/1 ps mean? (1)
   Ans: 1ns represents the reference time, and 1ps represents the precision value, up to what precision answer is considered.

3. Write the various ways to generate a clock (3)
   Ans:

```
//Method 1
`timescale 1ns/1ns
module clk_generation;
  bit clk;
  always #5 clk =~clk;
  initial begin
    clk=0;
    #50 $finish;
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

```
//Method 2
`timescale 1ns/1ns
module clk_generation;
  bit clk;
  initial forever #5 clk =~clk;
  initial begin
    clk=0;
    #50 $finish;
  end
```

```
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

```
//Method 3
`timescale 1ns/1ns
 module clk_generation;
  bit clk;
  initial begin
    clk=0;
     repeat(10) begin
      #5 clk =~clk;
    end

  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

4. State the difference between task and function in Verilog (2)
   Ans:

task	function
non-synthesizable	synthesizable
can have timing elements like @,#	cannot have timing elements
Task can give multiple output values based on output ports	Function can return only one value
Calling task inside a task and function inside a task is possible	Calling a task inside a function is not possible

5. Write a task in Verilog which takes 2 input arguments a and b of 3-bit wide, does the comparison and displays the result (perform a>b, a<b, a==b)(3)
   Ans:

```
module comparison;
  reg [3:0] a;
  reg [3:0] b;

  initial begin
    compare(3,2);
    compare(2,3);
    compare(2,2);
  end
  task compare(input reg [3:0]a, input reg [3:0]b);
    if(a>b)
      $display("a is greater than b");
    else if(a<b)
      $display("a is less than b");
    else if(a==b)
      $display("a is equal to b");
  endtask
endmodule
```

6. State the difference between blocking and non-blocking assignments with example (2)

Ans:

Blocking	**Non-Blocking**
Blocking assignment statements are assigned using (=) operator and are executed one after the other in a procedural block.	Non-blocking assignment statements are allowed to be scheduled without blocking the execution of the following statements and are specified by a (<=) symbol.
Used to design combinational circuits	Used to design sequential circuits

Example :

swapping two numbers
a=5; b=6;
non-blocking
a<=b;
b<=a;
blocking
temp=a;

a=b;
b=temp;

7. What is a sensitivity list? (1)
   Ans: It is an expression which evaluates an always block, enclosed within parentheses () followed by @ operator
   example: always @ (a or b)
8. What is the difference between initial block and always block (2)
   Ans:

Initial	always
executes only once	executes many times depending on the sensitivity list
Non-synthesizable	Synthesizable
can only be used in testbench	can be used both in design code and testbench

9. What is the difference between $display and $monitor? Explain with a simple example (2)
   Ans:

$display	$monitor
Executes only once in a simulation time slot	Executes whenever there is any value change
Works in active region	Works in postponed region

10. Write a Verilog code to implement 4:1 mux using 2:1 mux. Write the associated testbench for the same (5)
    CODE:

```
module mux_2to1(
    input wire sel,
  input a,b,
    output wire y
);

  assign y = (sel == 1'b0) ? a : b;

endmodule
```

```
module mux_4to1(
  input wire [1:0] sel,
    input wire [3:0] data_in,
    output reg y
);

wire m;
wire n;

mux_2to1 mux1(
  .sel(sel[0]),
  .a(data_in[0]),
  .b(data_in[1]),
  .y(m)
);

mux_2to1 mux2(
  .sel(sel[0]),
  .a(data_in[2]),
  .b(data_in[3]),
  .y(n)
);


mux_2to1 mux3(
  .sel(sel[1]),
  .a(m),
  .b(n),
  .y(y)
);

endmodule
```

TESTBENCH:

```
module mux_4to1_tb;

  // Inputs
  reg [1:0] sel;
  reg [3:0] data_in;

  // Output
  wire y;
```

```
  // Instantiate the mux_4to1 module
  mux_4to1 uut (
    .sel(sel),
    .data_in(data_in),
    .y(y)
  );

  // Stimulus
  initial begin
    $monitor("Time=%0t sel=%b data_in=%b y=%b", $time, sel,
data_in, y);

    // Test case 1
    sel = 2'b00;
    data_in = 4'b0001;
    #10;

    // Test case 2
    sel = 2'b01;
    data_in = 4'b0010;
    #10;

    // Test case 3
    sel = 2'b10;
    data_in = 4'b0100;
    #10;

    // Test case 4
    sel = 2'b11;
    data_in = 4'b1000;
    #10;
  end

endmodule
```

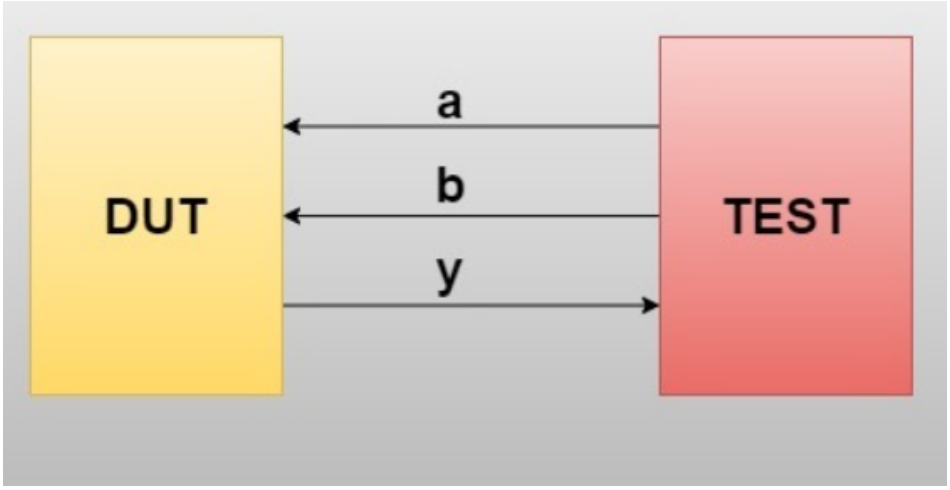
11. Difference between inter and intra delay statements with example (2)
    Inter delay : After the delay the RHS will be evaluated and assigned.
    Intra delay : RHS will be evaluated before delay and assigned to LHS after delay.
    module intra_inter;
     reg [3:0] a, b, out1;
     reg [3:0] c, d, out2;

```
//INTRA DELAY
initial begin
        a = 0;
        b = 0;
        for(integer i = 0; i<10; i++) begin
        a <= a+1;
        b <= b+1;
        out = #5(a+b);
        $display("At time %t intra delay output %d", $time, out1);
        end
end
//INTER DELAY
initial begin
        c = 0;
        d = 0;
        for(integer j = 0; j<10; j++) begin
        c <= c+1;
        d <= d+1;
        #5 out2 = (c+d);
        $display("At time %t inter delay output %d", $time, out2);
        end
end
initial begin
        #100 $finish;
end
endmodule
```

12. Difference between always and forever in Verilog (1)

Always	Forever
Is a procedural block and is executed based on the sensitivity list	Is a loop that runs continuously
Synthesizable	Non-synthesizable

13. Explain port connectivity in Verilog (2)
    The original input ports of DUT will be outputs of testbench and vice versa. Consider y = a AND b. Port connection between DUT and testbench will be as shown below:

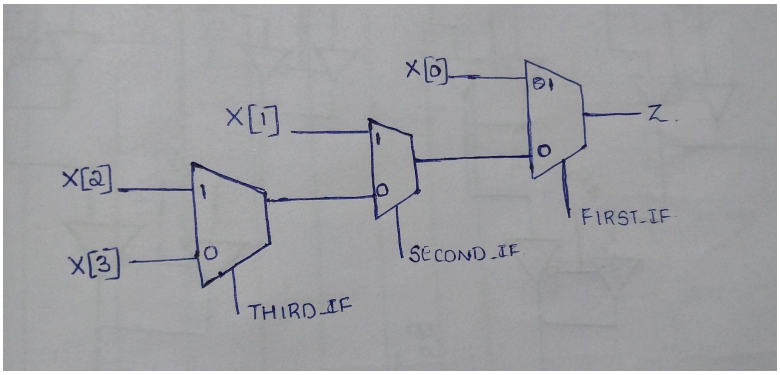


14. Write the difference between if-else and case (2)

Ans:

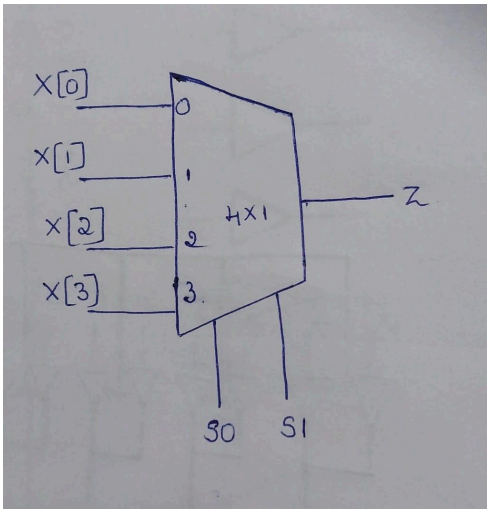
If else will result in priority 2x1 mux

```
if (S0 == 0 && S1 = 0)
begin : FIRST_IF
   Z <= X[0];
end : FIRST_IF
else if (S0 == 0 && S1 == 1)
begin: SECOND_IF
   Z <= X[1];
end: SECOND_IF
else if (S0 == 1 and S1 == 0)
begin: THIRD_IF
   Z <= X[2];
end: THIRD_IF
  else
   Z <= X[3];
```



Case will result in single MUX

case ({S0,S1})
00 : Z = X[0];
01 : Z = X[1];
10 : Z = X[2];
11 : Z = X[3];
endcase



15. Write the difference between logical and case equality with example (2)
    Logical equality considers 0 and 1 as 2 states and x and z as unknown.
    Eg: 101x == 1010 results in x
    Case equality considers 0, 1, x, z as 4 different states.
    Eg: 101x === 1010 results in 0
16. Write a Verilog code to implement a D-latch with synchronous active low reset. (3)

```
module d_sync(en,reset,d,q);
  input en,reset,d;
  output reg q;
  always@(en or d)
   begin
     if(en) begin
     if(!reset) // active low because it consumes less power as we
can easily drive 1 to 0 just we need to ground
       q<=0;
     else
       q<=d;
   end
   end
endmodule
```

TESTBENCH :

```
module tb;

  reg en,reset,d;
  wire q;

  always #5 en=~en;
  d_sync dut(en,reset,d,q);
  initial begin
    reset=0;
    en=0;
    d=0;

    #10 reset=1;
    d=1;
    #10 d=0;
    #10 d=1;
    #10 reset=0;
    d=1;

    #20 $finish;
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```

17. Write a Verilog code for SR flip flop with asynchronous active high reset (4)

Ans :

CODE

```
module sr_ff(sr,clk,rst,q);
  input [1:0] sr;
  input clk,rst;
  output reg q;
  always@(posedge clk or posedge rst) begin
    if (rst) //active high reset
            q <= 1'b0;
    else
      begin
        case(sr)
```

```
          2'b00: q <= q;
          2'b01: q <= 0;
          2'b10: q <= 1;
          2'b11: q <= 1'bx;
          default: q <= q;
        endcase
      end
  end
endmodule
```

18. Write a Verilog code which performs up counting when mode input is asserted and down counting when mode input is deasserted (4)
    Ans:
    CODE:

```
module counter(input clk,rst,mode,output reg [3:0] count);
  always @(posedge clk)
    begin
      if(rst)
        count<=0;
      else begin
        if(mode)
          count<=count+1;
        else
          count<=count-1;
      end
    end
endmodule
```

TESTBENCH :

```
module tb;
  reg rst, clk,mode;
  wire[3:0] count;

  always #5clk=~clk;

  counter dut(clk,rst,mode,count);

  initial begin
    rst=1;
    clk=0;
```

```
    mode=1;

    #10 rst=0;
    #150 mode=0;
  end

  initial begin
    $dumfile("dump.vcd");
    $dumpvars;
    #300 $finish;
  end
endmodule
```

19. Write a Verilog code to generate clock by 2 logic (3)

Ans:

CODE

```
module clk2(input clk,reset,output q1);
  wire q;
  wire d;
  d_ff d1(clk,reset,d,q1);
  assign d=~q1;
endmodule


module d_ff(clk,reset,d,q);
  input clk,reset,d;
  output reg q;
  wire qbar;
  always@(posedge clk) begin
    if(reset)
      q<=0;
    else
      q<=d;
  end
  assign qbar = ~q;

endmodule
```

TESTBENCH

```
module clk_tb;
reg clk,reset;
```

```
  wire q;

  always #5 clk=~clk;

  clk2 dut(.clk(clk),.reset(reset),.q1(q));

  initial begin
    clk=0;
    reset=1;
    #6 reset=0;
    #70 $finish;
  end

  initial begin
    $dumpfile("dump.vcd");
  $dumpvars;
  end

endmodule
```

20. Find the bug in the below code: (3)

```
module 4_mux (in0, in1, in2, in3,sel, out);
input in0,in1,in2,in3;
input reg [1:0] sel;
output wire out;
always@(sel or out)
begin
case (sel)
2'b00: out = in0;
2'b01: out = in1;
2'b10: out = in2;
default: out = in3;
endcase
end
endmodule
```

Ans:

1. input [1:0] sel
2. output reg out
3. Always @ (*)

4. module mux_4 not 4_mux

21. Write a function in Verilog which takes 2 arguments n and number. Return $(number)^n$ without using ** (2)

```
module power;
  integer result;
  initial begin
  result = powerof(3,5);
    $display("result=%0d",result);
  end
endmodule
function integer powerof ( input integer n, input integer
number);
 integer x;
  x=1;
  for(integer i=0; i<n;i=i+1)begin
    x=x*number;
  end
  return x;
endfunction
```