1) Explain port connectivity? How is the port connectivity between Design and testbench for an AND gate?(Explain with block diagram)

Port connectivity refers to the interconnection of ports between different components or modules in a digital design. In a hardware description language (HDL) such as Verilog or VHDL, ports are the points of connection between different parts of a design, allowing signals to flow between them.Let's consider an AND gate as an example to explain port connectivity between its design and testbench. An AND gate is a fundamental digital logic gate with two inputs and one output. The output is high (1) only when both inputs are high.
Design (AND Gate):
In the design of the AND gate, you would define input and output ports.

```
module AND_gate(
    input A,
    input B,
    output Y
);
    assign Y = A & B;
endmodule
```

Testbench:-
In the testbench, you need to instantiate the AND gate module and provide stimulus to its input ports (A and B). You also need to observe or check the output (Y) to verify the correctness of the design.

```
module AND_gate_tb;
    reg A, B;
    wire Y;

    AND_gate AND_inst(
        .A(A),
        .B(B),
        .Y(Y)
    );

    initial begin
        A = 1'b0; B = 1'b0; // Example test case 1
        #10;
        A = 1'b0; B = 1'b1; // Example test case 2
```

```
        #10;
        // Add more test cases as needed
    end
endmodule
```

Here, A and B are declared as registers (reg) because they are driven by the testbench. Y is declared as a wire (wire) because it is an output from the design.

---

2) What are the two types of port connection methods during instantiation? Explain with example.

-->Port mapping in module instantiation can be done in two different ways:
*Port mapping by order
*Port mapping by name

Port mapping by order:
The order in which the ports are declared in the module instantiation statement corresponds to the order in which the ports are declared in the module definition.

Example:
Design:

```
module d_ff(d,q,clk,reset);
  input d,clk,reset;
  output reg q;
  always@(posedge clk)
    begin
      if(reset)
        q<=0;
      else
        q<=d;
    end
endmodule
```

Testbench:

```
module d_ff_tb;
  reg d,clk=0,reset;
  wire q;
  d_ff dut(d,q,clk,reset);//mapping by order
```

```
  always #1 clk=~clk;
  initial begin
    $monitor("clk=%b,d=%b,reset=%b,q=%b",clk,d,reset,q);
    #2 reset=1'b1;d=1'b0;
    #6 reset=1'b0;d=1'b1;
    #2 d=1'b0;
    #2 $finish;
  end
endmodule
```

Port mapping by name:
In named connection, each actual argument is explicitly associated with the formal parameter it corresponds to by specifying the port name.
Example:

```
module d_ff(d,q,clk,reset);
  input d,clk,reset;
  output reg q;
  always@(posedge clk)
    begin
      if(reset)
        q<=0;
      else
        q<=d;
    end
endmodule

module d_ff_tb;
  reg d,clk=0,reset;
  wire q;
  d_ff dut(.d(d),.q(q),.clk(clk),.reset(reset));//port mapping by name
  always #1 clk=~clk;
  initial begin
    $monitor("clk=%b,d=%b,reset=%b,q=%b",clk,d,reset,q);
    #2 reset=1'b1;d=1'b0;
    #6 reset=1'b0;d=1'b1;
    #2 d=1'b0;
    #2 $finish;
  end
endmodule
```

---

3)What is the difference between reg and wire?

Reg is used for procedural assignment and can store some value.
Wire elements must be continuously driven by something, and cannot store a value.
Wires are used in combinational logic while registers (reg) can be used in combinational as well as sequential logic.

---

4)What is the difference between $display and $monitor? Explain with a simple example.

-->$display is a system task used to display a message at the current simulation time. It does not continuously monitor the variables.

$monitor is a system task used to continuously monitor the variables specified in its arguments and display their values whenever they change.

Example for $diplay:

```
module d_ff_tb;
  reg d,clk=0,reset;
  wire q;
  d_ff dut(.d(d),.q(q),.clk(clk),.reset(reset));
  always #1 clk=~clk;
  initial begin

    #2 reset=1'b1;d=1'b0;
    #6 reset=1'b0;d=1'b1;
    #2 d=1'b0;
    #2 $display("clk=%b,d=%b,reset=%b,q=%b",clk,d,reset,q);
    #2 $finish;

  end
endmodule
```

Output:

```
# run -all
# clk=1,d=0,reset=0,q=0
# ** Note: $finish : testbench.sv(12)
```

example for $monitor:

```
module d_ff_tb;
  reg d,clk=0,reset;
  wire q;
  d_ff dut(.d(d),.q(q),.clk(clk),.reset(reset));
  always #1 clk=~clk;
  initial begin
    $monitor("clk=%b,d=%b,reset=%b,q=%b",clk,d,reset,q);
    #2 reset=1'b1;d=1'b0;
    #6 reset=1'b0;d=1'b1;
    #2 d=1'b0;
    #2 $finish;
  end
endmodule
```

Output:

```
# run -all
# clk=0,d=x,reset=x,q=x
# clk=1,d=x,reset=x,q=x
# clk=0,d=0,reset=1,q=x
# clk=1,d=0,reset=1,q=0
# clk=0,d=0,reset=1,q=0
# clk=1,d=0,reset=1,q=0
# clk=0,d=0,reset=1,q=0
# clk=1,d=0,reset=1,q=0
# clk=0,d=1,reset=0,q=0
# clk=1,d=1,reset=0,q=1
# clk=0,d=0,reset=0,q=1
# clk=1,d=0,reset=0,q=0
```

5) Design a clk/2 circuit and write a Verilog code for the same.

Design:

```
module d_ff(d,q,clk,reset);
  input d,clk,reset;
  output reg q;
  always@(posedge clk)
```

```
    begin
      if(reset)
        q<=0;
      else
        q<=~q;
    end
endmodule
```

Testbench:

```
module d_ff_tb;
  reg d,clk=0,reset;
  wire q;
  d_ff dut(.d(d),.q(q),.clk(clk),.reset(reset));
  always #1 clk=~clk;
  initial begin
    $monitor("clk=%b,reset=%b,q=%b",clk,reset,q);
    #2 reset=1'b1;
    #2 reset=1'b0;
    #25 $finish;
  end
endmodule
```

Output:

```
# clk=0,reset=x,q=x
# clk=1,reset=x,q=x
# clk=0,reset=1,q=x
# clk=1,reset=1,q=0
# clk=0,reset=0,q=0
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
```

```
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
# clk=1,reset=0,q=1
# clk=0,reset=0,q=1
# clk=1,reset=0,q=0
# clk=0,reset=0,q=0
```

---

6)Implement a Exclusive-or gate using only nand gate and write the verilog code for the same.

Design:

```
module ex_or_nand(a,b,y);
  input a,b;
  output y;
  wire c,d,e;
  nand(c,a,b);
  nand(d,a,c);
  nand(e,c,b);
  nand(y,d,e);
endmodule
```

Testbench:

```
module ex_or_nand_tb;
  reg a,b;
  wire y;
  ex_or_nand dut(a,b,y);
  initial begin
    $monitor("a=%b,b=%b,y=%b",a,b,y);
    #1 a=1'b0;b=1'b0;
    #1 a=1'b0;b=1'b1;
    #1 a=1'b1;b=1'b0;
    #1 a=1'b1;b=1'b1;
    #1 $finish;
```

```
  end
endmodule
```

Output:

```
# a=0,b=0,y=0
# a=0,b=1,y=1
# a=1,b=0,y=1
# a=1,b=1,y=0
```

---

7. Explain the behaviour of the circuit and write the verilog code for the same.

Design:

```
module sr_ff(s,r,q,q_bar,clk,reset);
  input s,r,clk,reset;
  output reg q,q_bar;
  wire in1,in2;
  and(in1,s,q_bar);
  and(in2,r,q);
  always@(posedge clk)
    begin
      if(reset)q<=0;
      else
        begin
          case({in1,in2})
            2'b00: q<=q;
            2'b01: q<=0;
            2'b10: q<=1;
            default:q<=q;
          endcase
        end
    end
      assign q_bar=~q;
endmodule
```

Testbench:

```
module sr_ff_tb;
  reg s,r,clk=0,reset;
  wire q,q_bar;
```

```
  sr_ff dut(s,r,q,q_bar,clk,reset);
  always #1 clk=~clk;
  initial begin
    $monitor("clk=%b,reset=%b,s=%b,r=%b,q=%b,q_bar=%b",clk,reset,s,r,q,q_bar);
    reset=1'b1;
    #2 reset=1'b0;s=1'b0;r=1'b1;//reset condition
    #4 s=1'b0;r=1'b0;//previous state
    #4 s=1'b1;r=1'b0;//set condition
    #2 s=1'b0;r=1'b0;//previous state
    #4 s=1'b1;r=1'b1;//invalid state is removed
    #4 s=1'b0;r=1'b0;
    #4 $finish;
  end
endmodule
```

Output:

```
# run -all
# clk=0,reset=1,s=x,r=x,q=x,q_bar=x
# clk=1,reset=1,s=x,r=x,q=0,q_bar=1
# clk=0,reset=0,s=0,r=1,q=0,q_bar=1
# clk=1,reset=0,s=0,r=1,q=0,q_bar=1
# clk=0,reset=0,s=0,r=1,q=0,q_bar=1
# clk=1,reset=0,s=0,r=1,q=0,q_bar=1
# clk=0,reset=0,s=0,r=0,q=0,q_bar=1
# clk=1,reset=0,s=0,r=0,q=0,q_bar=1
# clk=0,reset=0,s=0,r=0,q=0,q_bar=1
# clk=1,reset=0,s=0,r=0,q=0,q_bar=1
# clk=0,reset=0,s=1,r=0,q=0,q_bar=1
# clk=1,reset=0,s=1,r=0,q=1,q_bar=0
# clk=0,reset=0,s=0,r=0,q=1,q_bar=0
# clk=1,reset=0,s=0,r=0,q=1,q_bar=0
# clk=0,reset=0,s=0,r=0,q=1,q_bar=0
# clk=1,reset=0,s=0,r=0,q=1,q_bar=0
# clk=0,reset=0,s=1,r=1,q=1,q_bar=0
# clk=1,reset=0,s=1,r=1,q=0,q_bar=1
# clk=0,reset=0,s=1,r=1,q=0,q_bar=1
# clk=1,reset=0,s=1,r=1,q=1,q_bar=0
# clk=0,reset=0,s=0,r=0,q=1,q_bar=0
# clk=1,reset=0,s=0,r=0,q=1,q_bar=0
```

```
# clk=0,reset=0,s=0,r=0,q=1,q_bar=0
# clk=1,reset=0,s=0,r=0,q=1,q_bar=0
```

---

8)What is a procedural block? Difference between initial and always block.

-->Procedural blocks are executed sequentially and are primarily used for describing how data flows and operations are performed within a module.

Initial blocks are used to specify the initial values of signals and variables when a simulation starts. An initial block is executed only once,
at the beginning of the simulation, and is used to initialize the state of the system.

Always blocks, on the other hand, are used to specify the behavior of a system over time. An always block is executed repeatedly, whenever a signal or signals
in its sensitivity list change. An always block can be used to model combinational logic or sequential logic, depending on the type of statements inside the block.

---

9) What does `timescale 1ns/1 ps mean?
-->timescale directive specifies the time units used for simulation. The format of the timescale directive is timescale <time_unit>/<time_precision>.
<time_unit> is 1 nanosecond (ns), which is the basic time unit for simulation.
<time_precision> is 1 picosecond (ps), which means the simulation can distinguish time differences down to 1 picosecond.

---

10) Write a verilog code depicting different ways to generate a clock.

```
`timescale 1ps/1ps

module clk;
  reg  clk1, clk2, clk3;

  always #5clk1 = ~clk1;
```

```
 initial begin
   forever #5clk2 = ~clk2;
 end

  initial begin
   repeat(10) #5clk3 = ~clk3;
  end

 initial begin
   $dumpfile("dump.vcd");
   $dumpvars;
   clk1 = 1'b0;
   clk2 = 1'b0;
   clk3 = 1'b0;
   $monitor(" time = %0t clk1 = %b clk = %b clk3 = %b", $time, clk1, clk2, clk3);

   #50$finish;
 end
endmodule: clk
```

Output:-

```
time = 0 clk1 = 0 clk = 0 clk3 = 0
 time = 5 clk1 = 1 clk = 1 clk3 = 1
 time = 10 clk1 = 0 clk = 0 clk3 = 0
 time = 15 clk1 = 1 clk = 1 clk3 = 1
 time = 20 clk1 = 0 clk = 0 clk3 = 0
 time = 25 clk1 = 1 clk = 1 clk3 = 1
 time = 30 clk1 = 0 clk = 0 clk3 = 0
 time = 35 clk1 = 1 clk = 1 clk3 = 1
 time = 40 clk1 = 0 clk = 0 clk3 = 0
 time = 45 clk1 = 1 clk = 1 clk3 = 1
```