**Chapter -3: Control statements, function, tasks**

Q1.Demonstration with a system Verilog program any 2 types of module instantiation/port connection technique?

Ans. module adder (a, b,c\_in,sum,c\_out);

input logic [7:0] a;

input logic [7:0] b;

input logic c\_in;

output logic [7:0] sum;

output logic c\_out;

logic [8:0] result;

assign result =a+b+c\_in;

assign sum = result[7:0];

assign c\_out = result[8];

endmodule : adder

=========================testbench==============

// or browse Examples

module test\_bench ();

logic [7:0] tb\_a;

logic [7:0] tb\_b;

logic tb\_c\_in;

logic [7:0] tb\_sum;

logic tb\_c\_out;

//top module instanciation

adder adder1 (.a(tb\_a), .b(tb\_b), .c\_in(tb\_c\_in), .sum(tb\_sum), .c\_out(tb\_c\_out));

//functional part of tb

initial begin

#1;

tb\_a=1;

tb\_b=2;

tb\_c\_in=1;

#1;

tb\_a=10;

tb\_b=20;

tb\_c\_in=0;

#1;

tb\_a=55;

tb\_b=66;

tb\_c\_in=1;

#1;

tb\_a=0;

end

initial begin

$monitor("%0d",tb\_sum);

#100$finish();

end

endmodule : test\_bench

==========================output ============================

xcelium> run  
x  
4  
30  
122  
67  
Simulation complete via $finish(1) at time 100 NS + 0

===================2nd testbentch port ===========

module test\_bench ();

logic [7:0] tb\_a;

logic [7:0] tb\_b;

logic tb\_c\_in;

logic [7:0] tb\_sum;

logic tb\_c\_out;

//top module instanciation

adder adder1 (tb\_a,tb\_b,tb\_c\_in,tb\_sum,tb\_c\_out);

//functional part of tb

initial begin

#1;

tb\_a=1;

tb\_b=2;

tb\_c\_in=1;

$display(tb\_sum);

#1;

tb\_a=94;

tb\_b=54;

tb\_c\_in=1;

$display(tb\_sum);

#1;

tb\_a=0;

$display(tb\_sum);

$finish();

end

endmodule : test\_bench

==========================output ============================

xcelium> run  
 x  
 4  
149  
Simulation complete via $finish(1) at time 3 NS + 0

Q2.Demonstration with an example – fork\_join, forj\_joinany, fork\_joinnone?

Ans. // Code your testbench here

// or browse Examples

module tb();

initial begin

$display("task1 print start",$time);

fork

begin

#5 $display("task1 print 1",$time);

#15 $display("task1 print 11",$time);

end

begin

#5 $display("task1 print 2",$time);

#15 $display("task1 print 21",$time);

end

begin

#5 $display("task1 print 3",$time);

#15 $display("task1 print 31",$time);

end

#5 $display("task1 print end",$time);

join

end

initial begin

fork

$display("task2 start",$time);

begin

#5 $display("task2 print 1",$time);

#15 $display("task2 print11 ",$time);

end

begin

#5 $display("task2 print 2",$time);

#15 $display("task2 print21 ",$time);

end

begin

#5 $display("task2 print 3",$time);

#15 $display("task2 print31 ",$time);

end

#5 $display("task2 print end",$time);

join\_any

end

initial begin

fork

$display("task3 start",$time);

begin

#5 $display("task3 print 1",$time);

#15 $display("task3 print11 ",$time);

end

begin

#5 $display("task3 print 2",$time);

#15 $display("task3 print21 ",$time);

end

begin

#5 $display("task3 print 3",$time);

#15 $display("task3 print31 ",$time);

end

join\_none

#5 $display("task3 print end",$time);

end

endmodule

==========================output ============================

xcelium> run  
task1 print start 0  
task2 start 0  
task3 start 0  
task1 print 1 5  
task1 print 2 5  
task1 print 3 5  
task1 print end 5  
task2 print 1 5  
task2 print 2 5  
task2 print 3 5  
task2 print end 5  
task3 print end 5  
task3 print 1 5  
task3 print 2 5  
task3 print 3 5  
task1 print 11 20  
task1 print 21 20  
task1 print 31 20  
task2 print11 20  
task2 print21 20  
task2 print31 20  
task3 print11 20  
task3 print21 20  
task3 print31 20  
xmsim: \*W,RNQUIE: Simulation is complete.

Q3.Write a task to take inputs 16-bit address, 32-bit data\_in, operation read or write and output 32-bit data\_out and operation performed?

Ans. module tb();

task mem1();

reg [31:0] mem [1023:0];

bit [31:0] data\_in;

logic [31:0] data\_out;

bit[9:0] addr;

bit read\_write;

if(read\_write==0) begin

mem[addr]=data\_in;

data\_out = mem[addr];

$display("read operation ");

end

if(read\_write==1) begin

mem[addr]=data\_in;

#5 $display(mem[addr]);

#5 data\_out=mem[addr];

$display("write operation ");

end

#5 mem\_oper(data\_in,addr, read\_write, data\_out);

endtask

//

task mem\_oper(input bit [31:0] data\_in,input bit [9:0] addr,input bit read\_write,output logic [31:0] data\_out);

$display("data\_in=%h,data\_out=%h,addr=%h,read\_write=%h",data\_in,data\_out,addr,read\_write);

endtask

//

initial begin

mem1.data\_in=32'habcd\_abcd;

mem1.read\_write=1;

mem1.addr=10'd10;

#2 mem1;

end

endmodule

==========================output ============================

xcelium> run  
2882382797  
write operation   
data\_in=abcdabcd,data\_out=xxxxxxxx,addr=00a,read\_write=1  
xmsim: \*W,RNQUIE: Simulation is complete.

Q4. Write a function to take inputs 16-bit address, 32-bit data\_in, operation read or write and output 32-bit data\_out?

Ans.

module sv\_function;

bit [31:0] datain;

bit read\_write;

bit[16:0] addr;

//function to add two integer numbers.

function void dataout (bit [31:0] datain,bit read\_write, bit [16:0] addr) ;

logic [31:0] mem [0:1023];

bit [31:0] dataout1;

if(read\_write==0)

dataout1=mem[addr];

else

mem[addr]=datain;

endfunction

initial begin

datain=32'h5647\_6767;

addr=16'h0010;

read\_write=1;

dataout(datain,read\_write,addr);

$display("\tValue of datain = %0h",datain);

#5;

read\_write=0;

dataout(datain,read\_write,addr);

$display("\tValue of datain = %0h",dataout.dataout1);

end

endmodule

====================================output=================

xcelium> run  
 Value of datain = 56476767  
 Value of datain = 56476767  
xmsim: \*W,RNQUIE: Simulation is complete.