**Chapter -7: IPCs, Interface, Program**

Q1. Declare a semaphore with 1 key in a bucket and demonstrate with a program how two processes access the common resources?

Ans. module semophore();

semaphore sem = new(1);

initial begin

fork

begin

$display("watting task 1 for the key ",$time);

sem.get(1);

$display(" task1 getting key",$time);

#10 sem.put(1);

$display("put back the key",$time);

end

begin

$display("watting task 2 for the key ",$time);

sem.get(1);

$display(" task2 getting key",$time);

#10 sem.put(1);

$display("put back task2 the key",$time);

end

join

$display("both process end",$time);

end

endmodule

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

xcelium> run

watting task 1 for the key 0

task1 getting key 0

watting task 2 for the key 0

put back the key 10

task2 getting key 10

put back task2 the key 20

both process end 20

xmsim: \*W,RNQUIE: Simulation is complete.

Q2. Declare a mailbox and demonstrate the put(), get(), try\_get(),try\_put() methods with a program?

Ans. class packet ;

rand bit [15:0] data;

rand bit [7:0] addr;

//displaying randmize values

function void pre\_randomize ();

$display("Packet::Packet Generated");

$display("Packet::Addr=%0d,Data=%0d",addr,data);

endfunction

function void post\_randomize ();

$display("Packet::Packet Generated");

$display("Packet::Addr=%0d,Data=%0d",addr,data);

endfunction

endclass

// ---------------------------------------------

//generator

//----------------------------------------------

class generator;

packet pkt;

mailbox mbox;//creating handle of mailbox

//constructor

function new(mailbox mbox);

this.mbox=mbox;

endfunction

task run;

repeat(1) begin

pkt = new();

pkt.randomize();//generating packet

mbox.put(pkt);

$display("pkt is putted");

#5;

end

endtask

task run1;

repeat(1) begin

pkt = new();

pkt.randomize;//generating packet

mbox.try\_put(pkt);

$display("try\_pkt is putted");

#5;

end

endtask

endclass

//----------------------------------

//driver

//--------------------------------

class driver;

packet pkt;

mailbox mbox;//creating handle of mailbox

//constructor

function new(mailbox mbox);

this.mbox=mbox;

endfunction

task run;

repeat(1) begin

mbox.get(pkt);

$display("getting pkt");

$display("data=%d addr=%d",pkt.data,pkt.addr);

end

endtask

task run1;

repeat(1) begin

mbox.try\_get(pkt);

$display("try\_getting pkt");

$display("data=%d addr=%d",pkt.data,pkt.addr);

end

endtask

endclass

//==============================

//--------------------------

//module

//------------------------

module mail\_box;

generator gen;

driver dri;

mailbox mbox;

initial begin

mbox=new();

gen=new(mbox);

dri=new(mbox);

fork

gen.run();

dri.run();

#5gen.run1();

#5dri.run1();

join

end

endmodule

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

xcelium> run

Packet::Packet Generated

Packet::Addr=0,Data=0

Packet::Packet Generated

Packet::Addr=161,Data=44212

pkt is putted

getting pkt

data=44212 addr=161

Packet::Packet Generated

Packet::Addr=0,Data=0

Packet::Packet Generated

Packet::Addr=180,Data=40847

try\_pkt is putted

try\_getting pkt

data=40847 addr=180

xmsim: \*W,RNQUIE: Simulation is complet

=======================================================

Q3.write a SV for the following memory design with mod ports?

Ans. // Code your testbench here

// or browse Examples

interface port;

logic [7:0] datain;

logic [7:0] dataout;

logic[7:0] addr;

logic write;

logic read;

logic clk;

logic [7:0] datain1;

logic [7:0] dataout1;

logic[7:0] addr1;

logic write1;

logic read1;

logic clk1;

modport mem(input datain,addr,read,write,clk, output dataout);

modport mem2(input datain1,addr1,read1,write1,clk1, output dataout1);

endinterface

//testbench

module test();

port tb\_m1();

mem1 tb\_mem(.m1(tb\_m1.mem),.m2(tb\_m1.mem2));

initial begin

tb\_m1.addr=0;

tb\_m1.datain=0;

tb\_m1.dataout=0;

tb\_m1.clk=0;

tb\_m1.read=0;

tb\_m1.write=0;

//mam2

tb\_m1.addr1=0;

tb\_m1.datain1=0;

tb\_m1.dataout1=0;

tb\_m1.clk1=0;

tb\_m1.read1=0;

tb\_m1.write1=0;

forever begin

#5 tb\_m1.clk= ~tb\_m1.clk;

#5 tb\_m1.clk1= ~tb\_m1.clk1;

end

end

initial begin

#15;

tb\_m1.write=1;

tb\_m1.addr=2;

tb\_m1.datain=8'h22;

#5;

tb\_m1.write1=1;

tb\_m1.addr1=5;

tb\_m1.datain1=8'h24;

#15;

tb\_m1.write=0;

tb\_m1.read=1;

#5;

tb\_m1.write1=0;

tb\_m1.read1=1;

end

initial begin

$monitor("dataout:%0h addr:%0h 2 dataout1:%0h addr1:%0h ",tb\_m1.dataout,tb\_m1.addr,tb\_m1.dataout1,tb\_m1.clk1);

#100 $finish();

end

endmodule

////sstemverilog design\

// Code your design here

module mem1(port.mem m1, port.mem2 m2);

logic [7:0] mam [0:255];

always@(posedge m1.clk)

begin

if(m1.write && ! m1.read)

mam[m1.addr] <= m1.datain;

else if (m1.read && ! m1.write)

m1.dataout<=mam[m1.addr];

else

m1.dataout<=8'h00;

end

always@(posedge m2.clk1)

begin

if(m2.write1 && ! m2.read1)

mam[m2.addr1] <= m2.datain1;

else if (m2.read1 && ! m2.write1)

m2.dataout1<=mam[m2.addr1];

else

m2.dataout1<=8'h00;

end

endmodule

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

xcelium> run

dataout:0 addr:0 2 dataout1:0 addr1:0

dataout:0 addr:0 2 dataout1:0 addr1:1

dataout:0 addr:2 2 dataout1:0 addr1:1

dataout:0 addr:2 2 dataout1:0 addr1:0

dataout:0 addr:2 2 dataout1:0 addr1:1

dataout:0 addr:2 2 dataout1:0 addr1:0

dataout:22 addr:2 2 dataout1:0 addr1:0

dataout:22 addr:2 2 dataout1:24 addr1:1

dataout:22 addr:2 2 dataout1:24 addr1:0

dataout:22 addr:2 2 dataout1:24 addr1:1

dataout:22 addr:2 2 dataout1:24 addr1:0

dataout:22 addr:2 2 dataout1:24 addr1:1

Simulation complete via $finish(1) at time 100 NS + 0

**Chapter -8:Clocking blocks with interface**

Q1.Modify the interface declared in session 07 Q.3 by using clocking blocks?

Ans. module mem1(port.mem m1, port.mem2 m2);

logic [7:0] mam [0:255];

always@(posedge m1.cb.clk)

begin

if(m1.cb.write && ! m1.cb.read)

mam[m1.cb.addr] <= m1.cb.datain;

else if (m1.cb.read && ! m1.cb.write)

m1.cb.dataout<=mam[m1.cb.addr];

else

m1.cb.dataout<=8'h00;

end

always@(posedge m2.cb1.clk1)

begin

if(m2.cb1.write1 && ! m2.cb1.read1)

mam[m2.cb1.addr1] <= m2.cb1.datain1;

else if (m2.cb1.read1 && ! m2.cb1.write1)

m2.cb1.dataout1<=mam[m2.cb1.addr1];

else

m2.cb1.dataout1<=8'h00;

end

endmodule

// Code your testbench here

// or browse Examples

interface port(input bit clk2);

logic [7:0] datain;

logic [7:0] dataout;

logic[7:0] addr;

logic write;

logic read;

logic clk;

logic [7:0] datain1;

logic [7:0] dataout1;

logic[7:0] addr1;

logic write1;

logic read1;

logic clk1;

clocking cb @(posedge clk2);

input datain,addr,read,write,clk;

output dataout;

endclocking

clocking cb1@(negedge clk2);

input datain1,addr1,read1,write1,clk1;

output dataout1;

endclocking

modport mem(clocking cb,input datain,addr,read,write,clk,output dataout);

modport mem2(clocking cb1,input datain1,addr1,read1,write1,clk1,output dataout1);

endinterface

//testbench

module test();

logic clk2;

port tb\_m1(clk2);

initial begin

clk2=0;

forever #2 clk2=~clk2;

end

mem1 tb\_mem(.m1(tb\_m1.mem),.m2(tb\_m1.mem2));

initial begin

tb\_m1.addr=0;

tb\_m1.datain=0;

tb\_m1.dataout=0;

tb\_m1.clk=0;

tb\_m1.read=0;

tb\_m1.write=0;

//mam2

tb\_m1.addr1=0;

tb\_m1.datain1=0;

tb\_m1.dataout1=0;

tb\_m1.clk1=0;

tb\_m1.read1=0;

tb\_m1.write1=0;

forever begin

#5 tb\_m1.clk= ~tb\_m1.clk;

#5 tb\_m1.clk1= ~tb\_m1.clk1;

end

end

initial begin

#15;

tb\_m1.write=1;

tb\_m1.addr=2;

tb\_m1.datain=8'h22;

#5;

tb\_m1.write1=1;

tb\_m1.addr1=5;

tb\_m1.datain1=8'h24;

#15;

tb\_m1.write=0;

tb\_m1.read=1;

tb\_m1.write1=0;

tb\_m1.read1=1;

end

initial begin

$monitor("dataout:%0h addr:%0h 2 dataout1:%0h time:%0d ",tb\_m1.dataout,tb\_m1.addr,tb\_m1.dataout1,$time);

#100 $finish();

end

endmodule

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

xcelium> run

dataout:0 addr:0 2 dataout1:0 time:0

dataout:0 addr:2 2 dataout1:0 time:15

dataout:22 addr:2 2 dataout1:0 time:46

dataout:22 addr:2 2 dataout1:24 time:52

Simulation complete via $finish(1) at time 100 NS + 0