**#testbench.sv**

`include "top.sv"

**#driver.sv**

class driver;

packet pkt;

mailbox #(packet) mbx;

virtual memory\_if.tb vif;

bit [15:0] no\_of\_pkts\_recvd;

function new (input mailbox #(packet) mbx\_in,

input virtual memory\_if.tb vif\_in);

this.mbx = mbx\_in;

this.vif = vif\_in;

endfunction

task run ;

$display("@%0t [DRIVER] run started \n",$time);

while(1) //driver runs forever

begin

mbx.get(pkt);

no\_of\_pkts\_recvd++;

$display("@%0t [DRIVER] Received packet %0d from generator\n",$time,no\_of\_pkts\_recvd);

write();

read();

end//end\_of\_while

$display("@%0t [DRIVER] run ended \n",$time);

endtask

task write();

@(vif.cb);

$display("@%0t [DRIVER] write operation started\n",$time);

vif.cb.wr <= 1'b1;

vif.cb.addr <= pkt.addr;

vif.cb.data\_in <= pkt.data;

@(vif.cb);

$display("@%0t [DRIVER] write operation ended addr=%0d data=%0d\n",$time,pkt.addr,pkt.data);

endtask

task read();

$display("@%0t [DRIVER] read operation started\n",$time);

vif.cb.wr <= 1'b0;

vif.cb.addr <= pkt.addr;

@(vif.cb);

$display("@%0t [DRIVER] read operation ended addr=%0d \n",$time,pkt.addr);

endtask

function void report(input string str="Driver");

$display("@%0t [%s] Report: total\_packets\_driven=%0d \n",$time,str,no\_of\_pkts\_recvd);

endfunction

endclass

**#environment.sv**

`include "packet.sv"

`include "generator.sv"

`include "driver.sv"

`include "iMonitor.sv"

`include "oMonitor.sv"

`include "scoreboard.sv"

class environment;

bit [15:0] no\_of\_pkts;//assigned in testcase

mailbox #(packet) gen\_drv\_mbox; //will be connected to generator and driver

mailbox #(packet) mon\_in\_scb\_mbox;//will be connected to input monitor and mon\_in in scoreborad

mailbox #(packet) mon\_out\_scb\_mbox;//will be connected to output monitor and mon\_out in scoreborad

virtual memory\_if.tb vif;

virtual memory\_if.tb\_mon\_in vif\_mon\_in;

virtual memory\_if.tb\_mon\_out vif\_mon\_out;

generator gen;

driver drvr;

iMonitor mon\_in;

oMonitor mon\_out;

scoreboard scb;

function new (input virtual memory\_if.tb vif\_in,

input virtual memory\_if.tb\_mon\_in vif\_mon\_in,

input virtual memory\_if.tb\_mon\_out vif\_mon\_out,

input bit [15:0] no\_of\_pkts);

this.vif= vif\_in;

this.vif\_mon\_in=vif\_mon\_in;

this.vif\_mon\_out=vif\_mon\_out;

this.no\_of\_pkts=no\_of\_pkts;

endfunction

function void build();

$display("@%0t [Environment] build started \n",$time);

gen\_drv\_mbox = new;

mon\_in\_scb\_mbox = new;

mon\_out\_scb\_mbox = new;

gen = new(gen\_drv\_mbox,no\_of\_pkts);

drvr = new(gen\_drv\_mbox,vif);

mon\_in = new(mon\_in\_scb\_mbox,vif\_mon\_in,"iMonitor");

mon\_out = new(mon\_out\_scb\_mbox,vif\_mon\_out,"oMonitor");

scb = new(mon\_in\_scb\_mbox,mon\_out\_scb\_mbox);

$display("@%0t [Environment] build ended\n",$time);

endfunction

task run ;

$display("@%0t [Environment] run started \n",$time);

build();//contruct the components

reset();//Apply reset to design

gen.run();//start the generator

//Start all the components of environment

fork

drvr.run();

mon\_in.run();

mon\_out.run();

scb.run();

wait(no\_of\_pkts == scb.total\_pkts\_recvd);

join\_any

repeat(10) @(vif.cb);

report();

$display("@%0t [Environment] run ended \n",$time);

endtask

function void report();

$display("[Environment] \*\*\*\*\*\*\*\*\*\*\* Report Started \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

gen.report();

drvr.report();

mon\_in.report();

mon\_out.report();

scb.report();

$display("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

if(scb.m\_mismatches ==0)

$display("\*\*\*\*\*\*\*\*\*\*\*TEST PASSED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

else

begin

$display("\*\*\*\*\*\*\*\*\*\*\*TEST FAILED \*\*\*\*\*\*\*\*\*\* \n");

$display("\*\*\*\*\*\*\*\*\*\*\*Matches=%0d Mis\_matches=%0d \*\*\*\*\*\*\*\*\*\*\*\* \n",scb.m\_matches,scb.m\_mismatches);

end

$display("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

$display("[Environment] \*\*\*\*\*\*\*\*\*\*\* Report ended\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n");

endfunction

task reset();

$display("@%0t [Environment] \*\*\*\*\*\*\*\*\*\*\* Applying reset\*\*\*\*\*\*\*\*\*\*\* \n",$time);

vif.reset=1;

repeat(5) @(vif.cb);

vif.reset=0;

repeat(1) @(vif.cb);

$display("@%0t [Environment] \*\*\*\*\*\*\*\*\*\*\* De-asserted reset\*\*\*\*\*\*\*\*\*\*\* \n",$time);

endtask

endclass

**#generator.sv**

class generator;

bit [15:0] no\_of\_pkts;

packet pkt;

mailbox #(packet) mbx;

function new (mailbox #(packet) mbx\_in,bit [15:0] gen\_pkts\_no=1);

this.no\_of\_pkts= gen\_pkts\_no;

this.mbx = mbx\_in;

endfunction

task run ;

bit [15:0] pkt\_count;

packet ref\_pkt=new;

$display("@%0t [GENERATOR] run started \n",$time);

repeat(no\_of\_pkts)

begin

assert(ref\_pkt.randomize());

pkt=new;

pkt.copy(ref\_pkt);

mbx.put(pkt);

pkt\_count++;

//$display("@%0t [GENERATOR] Sent packet %0d to driver \n",$time,pkt\_count);

end

$display("@%0t [GENERATOR] run ended size of mailbox=%0d \n",$time,mbx.num());

endtask

function void report(input string str="GENERATOR");

$display("@%0t [%s] Report: total\_packets\_generated=%0d \n",$time,str,no\_of\_pkts);

endfunction

endclass

**#iMonitor.sv**

class iMonitor ;

string name;

bit [15:0] no\_of\_pkts\_recvd;

packet pkt;

virtual memory\_if.tb\_mon\_in vif;

mailbox #(packet) mbx;//will be connected to input of scoreboard

function new (input mailbox #(packet) mbx\_in,

input virtual memory\_if.tb\_mon\_in vif\_in,

input string name="iMonitor");

this.mbx = mbx\_in;

this.vif = vif\_in;

this.name=name;

endfunction

task run() ;

bit [15:0] addr;

$display("@%0t [%s] run started \n",$time,name);

while(1)

begin

@(vif.cb\_mon\_in.data\_in);

pkt=new;

pkt.addr = vif.cb\_mon\_in.addr;

pkt.data = vif.cb\_mon\_in.data\_in;//write data

mbx.put(pkt);

no\_of\_pkts\_recvd++;

pkt.print();

$display("@%0t [%s] Sent packet %0d to scoreboard \n",$time,name,no\_of\_pkts\_recvd);

end

$display("@%0t [%s] run ended \n",$time,name);//monitor will never end

endtask

function void report();

$display("@%0t [%s] Report: total\_packets\_received=%0d \n",$time,name,no\_of\_pkts\_recvd);

endfunction

endclass

**#oMonitor.sv**

class oMonitor ;

string name;

bit [15:0] no\_of\_pkts\_recvd;

packet pkt;

virtual memory\_if.tb\_mon\_out vif;

mailbox #(packet) mbx;//will be connected to input of scoreboard

function new (input mailbox #(packet) mbx\_in,

input virtual memory\_if.tb\_mon\_out vif\_in,

input string name="oMonitor");

this.mbx = mbx\_in;

this.vif = vif\_in;

this.name=name;

endfunction

task run() ;

bit [15:0] addr;

$display("@%0t [%s] run started \n",$time,name);

while(1)

begin

@(vif.cb\_mon\_out.data\_out);

//skip the loop when data\_out is in high impedance state

if(vif.cb\_mon\_out.data\_out === 'z || vif.cb\_mon\_out.data\_out === 'x)

begin

//$display("@%0t [%s] DEBUG 1 data\_out=%0d \n",$time,name,vif.cb\_mon\_out.data\_out);

continue;

end

//$display("@%0t [%s] data\_out=%0d \n",$time,name,vif.cb\_mon\_out.data\_out);

pkt=new;

pkt.addr = vif.cb\_mon\_out.addr;

pkt.data = vif.cb\_mon\_out.data\_out;//read data

mbx.put(pkt);

no\_of\_pkts\_recvd++;

pkt.print();

$display("@%0t [%s] Sent packet %0d to scoreboard \n",$time,name,no\_of\_pkts\_recvd);

end

$display("@%0t [%s] run ended \n",$time,name);//monitor will never end

endtask

function void report();

$display("@%0t [%s] Report: total\_packets\_received=%0d \n",$time,name,no\_of\_pkts\_recvd);

endfunction

endclass

**#packet.sv**

class packet;

string name="Packet";

rand bit [3:0] addr;

rand bit [31:0] data;

rand bit wr;

bit [3:0] prev\_addr;

bit [31:0] prev\_data;

constraint valid {

data inside {[10:9999]};

data != prev\_data;

addr != prev\_addr;

}

function void post\_randomize();

prev\_addr=addr;

prev\_data=data;

endfunction

extern function new(string name="Packet");

extern function void print();

extern function void copy(packet rhs);

extern function bit compare(packet rhs);

endclass

function packet::new(string name="Packet");

this.name=name;

endfunction

function void packet::print ();

$display("@%0t [Packet] wr=%0d addr=%0d data=%0d \n",$time,wr,addr,data);

endfunction

function void packet::copy (packet rhs);

if(rhs==null) begin

$display("[Packet] Error Null object passed to copy method \n");

return;

end

this.addr=rhs.addr;

this.data=rhs.data;

endfunction

function bit packet::compare (packet rhs);

bit result;

if(rhs==null) begin

$display("[Packet] Error Null object passed to compare method \n");

return 0;

end

result = (this.addr == rhs.addr);

result &= (this.data == rhs.data);

result &= (this.wr == rhs.wr);

return result;

endfunction

**#program\_mem.sv**

`include "test.sv"

program program\_test (memory\_if vif);

base\_test test;

initial

begin

test=new(vif.tb,vif.tb\_mon\_in,vif.tb\_mon\_out);

test.run();

$display("@%0t [top] simulation finished \n",$time);

$finish;

end

endprogram

**#scoreboard.sv**

class scoreboard;

bit [15:0] total\_pkts\_recvd;

packet ref\_pkt;

packet got\_pkt;

mailbox #(packet) mbx\_in; //will be connected to input monitor

mailbox #(packet) mbx\_out;//will be connected to output monitor

bit [15:0] m\_matches;

bit [15:0] m\_mismatches;

function new (input mailbox #(packet) mbx\_in,

input mailbox #(packet) mbx\_out);

this.mbx\_in = mbx\_in;

this.mbx\_out = mbx\_out;

endfunction

task run ;

$display("@%0t [SCOREBOARD] run started \n",$time);

while(1)

begin

mbx\_in.get(ref\_pkt);

mbx\_out.get(got\_pkt);

total\_pkts\_recvd++;

$display("@%0t [SCOREBOARD] Packet %0d received \n",$time,total\_pkts\_recvd);

if (ref\_pkt.compare(got\_pkt) )

begin

m\_matches++;

$display("@%0t [SCOREBOARD] Packet %0d Matched \n",$time,total\_pkts\_recvd);

end

else

begin

m\_mismatches++;

$display("@%0t Error [SCOREBOARD] Packet %0d Not\_Matched \n",$time,total\_pkts\_recvd);

$display("@%0t \*\*\* Expected addr=%0d data=%0d \n Received addr=%0d data=%0d \*\*\*\*",$time,ref\_pkt.addr,ref\_pkt.data,got\_pkt.addr,got\_pkt.data);

end

end

$display("@%0t [SCOREBOARD] run ended \n",$time);

endtask

function void report();

$display("@%0t [SCOREBOARD] Report: total\_packets\_received=%0d \n",$time,total\_pkts\_recvd);

$display("@%0t [SCOREBOARD] Report: Matches=%0d Mis\_Matches=%0d \n",$time,m\_matches,m\_mismatches);

endfunction

endclass

**#test.sv**

`include "environment.sv"

class base\_test;

bit [15:0] no\_of\_pkts;

virtual memory\_if.tb vif;

virtual memory\_if.tb\_mon\_in vif\_mon\_in;

virtual memory\_if.tb\_mon\_out vif\_mon\_out;

environment env;

function new (input virtual memory\_if.tb vif\_in,

input virtual memory\_if.tb\_mon\_in vif\_mon\_in,

input virtual memory\_if.tb\_mon\_out vif\_mon\_out

);

this.vif= vif\_in;

this.vif\_mon\_in=vif\_mon\_in;

this.vif\_mon\_out=vif\_mon\_out;

endfunction

task setup();

no\_of\_pkts=50;

env = new(vif,vif\_mon\_in,vif\_mon\_out,no\_of\_pkts);

endtask

task run ();

$display("@%0t [TESTCASE] run started \n",$time);

setup();

env.run();

$display("@%0t [TESTCASE] run ended\n",$time);

endtask

endclass**#top.sv**

`include "program\_mem.sv"

module top;

parameter reg [15:0] ADDR\_WIDTH=4;

parameter reg [15:0] DATA\_WIDTH=31;

parameter reg [15:0] MEM\_SIZE=16;

bit clk;

always #10 clk=!clk;

memory\_if #(ADDR\_WIDTH,DATA\_WIDTH,MEM\_SIZE) mem\_if (clk);

memory\_rtl #(ADDR\_WIDTH,DATA\_WIDTH,MEM\_SIZE) dut (.clk(clk),.reset(mem\_if.reset),

.wr(mem\_if.wr),.addr(mem\_if.addr),

.data\_in(mem\_if.data\_in),

.data\_out(mem\_if.data\_out)

);

program\_test ptest (mem\_if);

endmodule

**#design.sv**

`include "if\_memory.sv"

module memory\_rtl (clk,reset,wr,addr,data\_in,data\_out);

//Synchronous write read memory

parameter reg [15:0] ADDR\_WIDTH=4;

parameter reg [15:0] DATA\_WIDTH=31;

parameter reg [15:0] MEM\_SIZE=16;

input clk,reset;

input wr;// for write wr=1;

// for read wr=0;

input [ADDR\_WIDTH-1:0] addr;

input [DATA\_WIDTH-1:0] data\_in;

output [DATA\_WIDTH-1:0] data\_out;

wire [DATA\_WIDTH-1:0] data\_out;

reg [DATA\_WIDTH-1:0] mem [MEM\_SIZE];

reg [DATA\_WIDTH-1:0] rdata;

reg out\_enable;//controls when to pass read data on data\_out pin

//if wr=1 data\_out should be in high impedance state

//if wr=0 data\_out should be content of memory with given address

assign data\_out = out\_enable ? rdata : 'bz;

//asynchronous reset and synchronous write

always @(posedge clk or posedge reset)

begin

if (reset)

for(int i=0;i<MEM\_SIZE;i++)

mem[i] <= 'b0;

else if(wr)

mem[addr] <= data\_in;//write to mem location at given addr

end//end\_of\_write

//Synchronous Read

always @(posedge clk )

begin

if(wr==0)

begin

rdata <= mem[addr]; //read from memory

out\_enable <= 1'b1;

end

else out\_enable <=1'b0;

end//end\_of\_read

endmodule

**#if\_memory.sv**

interface memory\_if (input clk);

parameter reg [15:0] ADDR\_WIDTH=4;

parameter reg [15:0] DATA\_WIDTH=31;

parameter reg [15:0] MEM\_SIZE=16;

logic reset;

logic wr;// for write wr=1;

// for read wr=0;

logic [ADDR\_WIDTH-1:0] addr;

logic [DATA\_WIDTH-1:0] data\_in;

logic [DATA\_WIDTH-1:0] data\_out;

clocking cb @(posedge clk);

//Directions are w.r.t to testbench

output wr;

output data\_in;

output addr;

endclocking

clocking cb\_mon\_in @(posedge clk);

input wr;

input data\_in;

input addr;

endclocking

clocking cb\_mon\_out @(posedge clk);

input data\_out;

input wr;

input addr;

endclocking

//modport for specifying directions

modport tb (clocking cb,output reset);

modport tb\_mon\_in (clocking cb\_mon\_in);

modport tb\_mon\_out (clocking cb\_mon\_out);

//modport dut (input addr,data\_in,wr,reset, output data\_out);

endinterface