#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
```