**design.sv**

// Code your design here

**if\_memery.sv**

interface memory\_if (input clk);

parameter reg [15:0] ADDR\_WIDTH=8;

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

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

logic reset;

logic slv\_rsp;

logic wr;// for write wr=1;

// for read wr=0;

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

logic [DATA\_WIDTH-1:0] wdata;

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

clocking cb @(posedge clk);

//Directions are w.r.t to testbench

output wr;

output wdata;

output addr;

input rdata;

endclocking

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

input wr;

input wdata;

input addr;

endclocking

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

input rdata;

input wr;

input addr;

endclocking

//modport for specifying directions

modport tb (clocking cb,output reset,input slv\_rsp);

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

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

endinterface

**memory\_rtl.sv**

module memory\_rtl (clk,reset,wr,addr,wdata,rdata,response);

//Synchronous write read memory

parameter reg [15:0] ADDR\_WIDTH=8;

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] wdata;

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

output response;

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

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

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

//csr1\_wr\_count is a read only register

reg [31:0] csr1\_wr\_count;//addr 'h18 = 'd24

reg [7:0] csr2\_CHIP\_EN;//addr 'h20 = 'd32

reg [31:0] csr3;//addr 'h24 = 'd36

reg [31:0] csr4\_dropped;//addr 'h26 = 'd38

reg response ;//Provides response to master on successful write

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

//if wr=1 rdata should be in high impedance state

//if wr=0 rdata should be content of memory with given address

assign rdata = (out\_enable & csr2\_CHIP\_EN[0]) ? data\_out : 'bz;

//asynchronous reset and synchronous write

always @(posedge clk or posedge reset)

begin

if (reset) begin

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

mem[i] <= 'b0;

csr1\_wr\_count <='b0;

csr2\_CHIP\_EN <='b0;

csr3 <='b0;

csr4\_dropped <= 'b0;

end

else if(wr ) begin

casex (addr)

16'h0020 : csr2\_CHIP\_EN <= wdata;

16'h0024 : csr3 <= wdata;

default : begin

if(csr2\_CHIP\_EN[0] && (addr inside {[0:15]}) ) begin

mem[addr[3:0]] <= wdata ;

csr1\_wr\_count[7:0]++;

response <=1'b1;

end else

csr4\_dropped++;

end//end\_of\_default

endcase //end\_of\_case

end

else response <=1'b0;

end//end\_of\_write

//Synchronous Read

always @(posedge clk )

begin

if(wr==0) begin

casex(addr)

16'h0018 : data\_out <= csr1\_wr\_count;

16'h0020 : data\_out <= {24'h0,csr2\_CHIP\_EN};

16'h0024 : data\_out <= csr3;

16'h0026 : data\_out <= csr4\_dropped;

default : begin

if(csr2\_CHIP\_EN[0]) begin

data\_out <= mem[addr[3:0]] ;

csr1\_wr\_count[15:8]++;

end

end

endcase //end\_of\_case

out\_enable <= 1'b1;

end

else out\_enable <=1'b0;

end//end\_of\_read

endmodule

**testbench.sv**

`include "top.sv"

**base\_sequence.sv**

class base\_sequence extends uvm\_sequence#(packet);

int unsigned pkt\_count;

`uvm\_object\_utils(base\_sequence)

function new (string name="base\_sequence");

super.new(name);

set\_automatic\_phase\_objection(1);//uvm-1.2 only

endfunction

extern virtual task pre\_start();

extern virtual task body();

endclass

task base\_sequence::pre\_start();

string str;

//below one is for when setting item\_count at agent level in test

//uvm\_sequencer\_base sqr=get\_sequencer();

//if(!uvm\_config\_db #(int):: get(sqr.get\_parent(),"","item\_count",item\_count))

//below one is for when setting item\_count at sequencer level in test

//if(!uvm\_config\_db #(int):: get(this.get\_sequencer(),"","item\_count",item\_count))

//below one is for when setting item\_count at sequence level in test

if(!uvm\_config\_db #(int):: get(null,this.get\_full\_name,"item\_count",pkt\_count))

begin

`uvm\_warning(get\_full\_name(),"Packet count is not set hence generating 10 transactions")

// pkt\_count=1;

end

if (uvm\_config\_db#(string)::get(null,"lucid\_vlsi","testing", str))

$display("String received is %0s",str);

else

$display("test did not set the string so you may seee");

endtask

task base\_sequence::body();

repeat(pkt\_count) begin

`uvm\_do(req);

end

endtask

**base\_test.sv**

class base\_test extends uvm\_test;

`uvm\_component\_utils(base\_test)

environment env;

virtual memory\_if.tb mvif;

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

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

function new (string name="base\_test",uvm\_component parent=null);

super.new(name,parent);

endfunction

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual task main\_phase (uvm\_phase phase);

endclass

function void base\_test::build\_phase(uvm\_phase phase);

super.build\_phase(phase);

env=environment::type\_id::create("env",this);

uvm\_config\_db#(virtual memory\_if.tb)::get(this,"","master\_if",mvif);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_in)::get(this,"","mon\_in",vif\_min);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_out)::get(this,"","mon\_out",vif\_mout);

uvm\_config\_db#(virtual memory\_if.tb)::set(this,"env.m\_agent","drvr\_if",mvif);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_in)::set(this,"env.m\_agent","iMon\_if",vif\_min);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_out)::set(this,"env.s\_agent","oMon\_if",vif\_mout);

uvm\_config\_db#(int)::set(this,"env.m\_agent.seqr.\*", "item\_count", 10);

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.reset\_phase","default\_sequence",reset\_sequence::get\_type());

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.configure\_phase","default\_sequence",config\_sequence::get\_type());

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.main\_phase","default\_sequence",base\_sequence::get\_type());

endfunction

task base\_test::main\_phase (uvm\_phase phase);

uvm\_objection objection;

super.main\_phase(phase);

objection=phase.get\_objection();

//The drain time is the amount of time to wait once all objections have been dropped

objection.set\_drain\_time(this,100ns);

endtask

**config\_sequence.sv**

class config\_sequence extends base\_sequence;

`uvm\_object\_utils(config\_sequence)

function new (string name="config\_sequence");

super.new(name);

endfunction

extern virtual task body();

endclass

task config\_sequence::body();

`uvm\_create(req);

req.addr='h20;

req.data='h1;

req.mode=CFG\_REG\_WRITE;

start\_item(req);

finish\_item(req);

`uvm\_info("CONFIG\_SEQ","Config CFG\_REG\_WRITE sequence Transaction Done \n ",UVM\_MEDIUM);

endtask

**coverage.sv**

class coverage extends uvm\_subscriber#(packet);

`uvm\_component\_utils(coverage)

real coverage\_score;

packet pkt;

covergroup cov\_mem with function sample(packet pkt) ;

coverpoint pkt.addr; // Measure coverage

endgroup

function new (string name="coverage",uvm\_component parent);

super.new(name,parent);

cov\_mem=new;

endfunction

virtual function void write( T t);

if (!$cast(pkt,t.clone)) begin

`uvm\_fatal("COV","Transaction object supplied is NULL in coverage component");

end

cov\_mem.sample(pkt);

coverage\_score=cov\_mem.get\_coverage();

`uvm\_info("COV",$sformatf("Coverage=%0f%%",coverage\_score),UVM\_MEDIUM);

endfunction

virtual function void extract\_phase(uvm\_phase phase);

uvm\_config\_db#(real)::set(null,"uvm\_test\_top.env","cov\_score",coverage\_score);

endfunction

endclass

**driver.sv**

class driver extends uvm\_driver#(packet);

`uvm\_component\_utils(driver)

bit [31:0] pkt\_id;

virtual memory\_if.tb vif;

bit send\_resp;

function new (string name="driver",uvm\_component parent);

super.new(name,parent);

endfunction

extern virtual task run\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

extern virtual task write(input packet pkt);

extern virtual task read(input packet pkt);

extern virtual task drive\_reset();

extern virtual task drive(packet pkt);

extern virtual task config\_reg\_write(packet pkt);

extern virtual task config\_reg\_read(packet pkt);

extern virtual task drive\_stimulus(packet pkt);

endclass

task driver::run\_phase(uvm\_phase phase);

forever begin

seq\_item\_port.get\_next\_item(req);

pkt\_id++;

drive(req);

seq\_item\_port.item\_done();

`uvm\_info(get\_type\_name(),$sformatf("Driver:: %0s Transaction %0d Done",req.mode.name(),pkt\_id),UVM\_MEDIUM);

end

endtask

task driver::drive(packet pkt);

case (req.mode)

RESET : drive\_reset();

CFG\_REG\_WRITE : config\_reg\_write(req);

CFG\_REG\_READ : config\_reg\_read(req);

default : drive\_stimulus(req);

endcase

endtask

task driver::drive\_stimulus(packet pkt);

write(req);

if(send\_resp==1'b1) begin

rsp=packet::type\_id::create("rsp",this);

rsp.slv\_rsp=vif.slv\_rsp == 1'b1 ? OK : ERROR ;

rsp.set\_id\_info(req);

seq\_item\_port.put\_response(rsp);

end

read(req);

endtask

function void driver::connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

uvm\_config\_db#(bit)::get(this,"", "set\_resp\_for\_drvr",send\_resp );

uvm\_config\_db#(virtual memory\_if.tb)::get(get\_parent(),"","drvr\_if",vif);

assert(vif != null) else

`uvm\_fatal(get\_type\_name(),"Virtual interface in driver is NULL ");

endfunction

task driver::write(input packet pkt);

@(vif.cb);

`uvm\_info(get\_type\_name()," Write transaction started...",UVM\_FULL);

vif.cb.wr <= 1'b1;

vif.cb.addr <= pkt.addr;

vif.cb.wdata <= pkt.data;

@(vif.cb);

`uvm\_info(get\_type\_name()," Write transaction ended ",UVM\_HIGH);

endtask

task driver::read(input packet pkt);

`uvm\_info(get\_type\_name()," Read transaction started...",UVM\_FULL);

vif.cb.wr <= 1'b0;

vif.cb.addr <= pkt.addr;

@(vif.cb);

`uvm\_info(get\_type\_name()," Read transaction ended ",UVM\_HIGH);

endtask

task driver::drive\_reset();

`uvm\_info(get\_type\_name(),"Reset transaction started...",UVM\_FULL);

vif.reset <= 1'b1;

repeat (2) @(vif.cb);

vif.reset <= 1'b0;

`uvm\_info(get\_type\_name(),"Reset transaction ended ",UVM\_HIGH);

endtask

task driver::config\_reg\_write(packet pkt);

`uvm\_info(get\_type\_name(),"Configuration WRITE transaction started...",UVM\_FULL);

vif.cb.addr <= pkt.addr;

vif.cb.wdata <= pkt.data;

vif.cb.wr <= 1'b1;

repeat (2) @(vif.cb);

`uvm\_info(get\_type\_name(),"Configration WRITE transaction ended ",UVM\_HIGH);

endtask

task driver::config\_reg\_read(packet pkt);

`uvm\_info(get\_type\_name(),"Configuration READ transaction started...",UVM\_FULL);

vif.cb.addr <= pkt.addr;

vif.cb.wr <= 1'b0;

repeat (2) @(vif.cb);

pkt.data=vif.cb.rdata;

`uvm\_info(get\_type\_name(),"Configration READ transaction ended ",UVM\_HIGH);

endtask

**environment.sv**

class environment extends uvm\_env;

`uvm\_component\_utils(environment)

bit [31:0] exp\_pkt\_count;

real tot\_cov\_score;

bit [31:0] m\_matches,mis\_matches,csr4\_dropped;

bit cov\_enable;

master\_agent m\_agent;

slave\_agent s\_agent;

scoreboard scb;

coverage cov\_comp;

function new (string name="environment",uvm\_component parent=null);

super.new(name,parent);

endfunction

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

extern virtual function void report\_phase(uvm\_phase phase);

extern virtual function void extract\_phase(uvm\_phase phase);

endclass

function void environment::build\_phase(uvm\_phase phase);

super.build\_phase(phase);

m\_agent=master\_agent::type\_id::create("m\_agent",this);

s\_agent=slave\_agent::type\_id::create("s\_agent",this);

scb=scoreboard::type\_id::create("scb",this);

uvm\_config\_db#(bit)::get(this,"","enable\_coverage",cov\_enable);

if(cov\_enable)

cov\_comp=coverage::type\_id::create("cov\_comp",this);

endfunction

function void environment::connect\_phase(uvm\_phase phase);

m\_agent.ap.connect(scb.mon\_in);

s\_agent.ap.connect(scb.mon\_out);

if(cov\_enable) m\_agent.ap.connect(cov\_comp.analysis\_export);

endfunction

function void environment::extract\_phase(uvm\_phase phase);

uvm\_config\_db#(int)::get(this,"m\_agent.seqr.\*","item\_count",exp\_pkt\_count);

uvm\_config\_db#(real)::get(this,"","cov\_score",tot\_cov\_score);

uvm\_config\_db#(int)::get(this,"","matches",m\_matches);

uvm\_config\_db#(int)::get(this,"","mis\_matches",mis\_matches);

uvm\_config\_db#(bit [31:0])::get(this,"m\_agent.seqr","dropped\_count",csr4\_dropped);

endfunction

function void environment::report\_phase(uvm\_phase phase);

bit [31:0] tot\_scb\_cnt;

tot\_scb\_cnt= m\_matches + mis\_matches;

if(exp\_pkt\_count != tot\_scb\_cnt) begin

`uvm\_info("","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",UVM\_NONE);

`uvm\_info("FAIL","Test Failed due to packet count MIS\_MATCH",UVM\_NONE);

`uvm\_info("FAIL",$sformatf("exp\_pkt\_count=%0d Received\_in\_scb=%0d ",exp\_pkt\_count,tot\_scb\_cnt),UVM\_NONE);

`uvm\_fatal("FAIL","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Test FAILED \*\*\*\*\*\*\*\*\*\*\*\*");

end

else if(mis\_matches != 0) begin

`uvm\_info("","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",UVM\_NONE);

`uvm\_info("FAIL","Test Failed due to mis\_matched packets in scoreboard",UVM\_NONE);

`uvm\_info("FAIL",$sformatf("matched\_pkt\_count=%0d mis\_matched\_pkt\_count=%0d ",m\_matches,mis\_matches),UVM\_NONE);

`uvm\_fatal("FAIL","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Test FAILED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

end

else begin

`uvm\_info("PASS","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Test PASSED \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",UVM\_NONE);

`uvm\_info("PASS",$sformatf("exp\_pkt\_count=%0d Received\_in\_scb=%0d ",exp\_pkt\_count,tot\_scb\_cnt),UVM\_NONE);

`uvm\_info("PASS",$sformatf("matched\_pkt\_count=%0d mis\_matched\_pkt\_count=%0d ",m\_matches,mis\_matches),UVM\_NONE);

`uvm\_info("PASS",$sformatf("Coverage=%0f%%",tot\_cov\_score),UVM\_NONE);

`uvm\_info("","\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*",UVM\_NONE);

end

endfunction

**iMonitor.sv**

class iMonitor extends uvm\_monitor;

`uvm\_component\_utils(iMonitor)

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

// This TLM port is used to connect the monitor to the scoreboard

uvm\_analysis\_port #(packet) analysis\_port;

// Current monitored transaction

packet pkt;

function new (string name="iMonitor",uvm\_component parent);

super.new(name,parent);

endfunction

extern virtual task run\_phase(uvm\_phase phase);

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

endclass

function void iMonitor::build\_phase(uvm\_phase phase) ;

super.build\_phase(phase);

analysis\_port=new("analysis\_port",this);

endfunction

function void iMonitor::connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

if (!uvm\_config\_db#(virtual memory\_if.tb\_mon\_in)::get(get\_parent(), "", "iMon\_if", vif)) begin

`uvm\_fatal(get\_type\_name(), "iMonitor DUT interface not set");

end

endfunction

task iMonitor::run\_phase(uvm\_phase phase);

// The job of the iMonitor is to passively monitor the physical signals,

// interprete and report the activities that it sees. In this case, to

// re-construct the packet that it sees on the DUT's input port as specified

forever begin

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

if(vif.cb\_mon\_in.addr=='h20) continue;

pkt = packet::type\_id::create("pkt",this);

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

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

`uvm\_info(get\_type\_name(),pkt.convert2string(),UVM\_MEDIUM);

analysis\_port.write(pkt);

end

endtask

**master\_agent.sv**

class master\_agent extends uvm\_agent;

`uvm\_component\_utils(master\_agent)

driver drvr;

iMonitor iMon;

sequencer seqr;

// pass through port

uvm\_analysis\_port#(packet) ap;

function new (string name="master\_agent",uvm\_component parent);

super.new(name,parent);

endfunction

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

endclass

function void master\_agent::build\_phase(uvm\_phase phase);

super.build\_phase(phase);

ap=new("ap",this);

if(is\_active==UVM\_ACTIVE) begin

seqr=sequencer::type\_id::create("seqr",this);

drvr=driver::type\_id::create("drvr",this);

end

iMon=iMonitor::type\_id::create("iMon",this);

endfunction

function void master\_agent::connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

if(is\_active==UVM\_ACTIVE) begin

drvr.seq\_item\_port.connect(seqr.seq\_item\_export);

end

iMon.analysis\_port.connect(this.ap);

endfunction

**mem\_env\_pkg.pkg**

package mem\_env\_pkg;

typedef enum {NORMAL,RESET,WRITE,READ,CFG\_REG\_WRITE,CFG\_REG\_READ} op\_type;

typedef enum {OK,ERROR} slv\_response\_type;

// UVM class library compiled in a package

import uvm\_pkg::\*;

`include "packet.sv"

`include "base\_sequence.sv"

`include "reset\_sequence.sv"

`include "config\_sequence.sv"

`include "shutdown\_sequence.sv"

`include "rw\_sequence.sv"

`include "sequencer.sv"

`include "driver.sv"

`include "iMonitor.sv"

`include "oMonitor.sv"

`include "coverage.sv"

`include "master\_agent.sv"

`include "slave\_agent.sv"

`include "scoreboard.sv"

`include "environment.sv"

endpackage

**mem\_test1.sv**

class mem\_test1 extends base\_test;

`uvm\_component\_utils(mem\_test1)

function new (string name="mem\_test1",uvm\_component parent=null);

super.new(name,parent);

endfunction

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void end\_of\_elaboration\_phase(uvm\_phase phase);

extern virtual function void start\_of\_simulation\_phase(uvm\_phase phase);

endclass

function void mem\_test1::build\_phase(uvm\_phase phase);

super.build\_phase(phase);

uvm\_config\_db#(int)::set(this,"env.m\_agent.seqr.\*", "item\_count", 30);

uvm\_config\_db#(bit)::set(this,"env.m\_agent.drvr", "set\_resp\_for\_drvr", 1'b1);

uvm\_config\_db#(bit)::set(this,"env","enable\_coverage",1'b1);

uvm\_config\_db#(string)::set(null,"lucid\_vlsi","testing", "hello srinivas");

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.reset\_phase","default\_sequence",reset\_sequence::get\_type());

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.configure\_phase","default\_sequence",config\_sequence::get\_type());

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.main\_phase","default\_sequence",rw\_sequence::get\_type());

uvm\_config\_db#(uvm\_object\_wrapper)::set(this,"env.m\_agent.seqr.shutdown\_phase","default\_sequence",shutdown\_sequence::get\_type());

endfunction

function void mem\_test1::end\_of\_elaboration\_phase(uvm\_phase phase);

super.end\_of\_elaboration\_phase(phase);

uvm\_root::get().print\_topology();

endfunction

function void mem\_test1::start\_of\_simulation\_phase(uvm\_phase phase);

`uvm\_info ("STUDY\_INFO","Original uvm\_info message STUDY\_INFO 1 ",UVM\_MEDIUM)

`uvm\_error("STUDY\_ERR","Original uvm\_error message 1")

`uvm\_error("STUDY\_ERR","Original uvm\_error message 2")

`uvm\_error("STUDY\_ERR\_M1","Original uvm\_error message 3")

`uvm\_error("STUDY\_ERR\_M2","Original uvm\_error message 4")

`uvm\_error("STUDY\_ERR\_M3","Original uvm\_error message 5")

`uvm\_warning("STUDY\_WARNING","Original uvm\_warning message 1")

`uvm\_warning("STUDY\_WARNING","Original uvm\_warning message 2")

`uvm\_warning("STUDY\_WARNING2","Original uvm\_warning message 3")

`uvm\_warning("STUDY\_WARNING2","Original uvm\_warning message 4")

`uvm\_warning("STUDY\_WARNING3","Original uvm\_warning message 5")

`uvm\_info ("STUDY\_INFO\_END","Original uvm\_info message STUDY\_INFO\_END ",UVM\_MEDIUM)

endfunction

**oMonitor.sv**

class oMonitor extends uvm\_monitor;

`uvm\_component\_utils(oMonitor)

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

// This TLM port is used to connect the monitor to the scoreboard

uvm\_analysis\_port #(packet) analysis\_port;

// Current monitored transaction

packet pkt;

function new (string name="oMonitor",uvm\_component parent);

super.new(name,parent);

endfunction

extern virtual task run\_phase(uvm\_phase phase);

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

endclass

function void oMonitor::build\_phase(uvm\_phase phase) ;

super.build\_phase(phase);

//create TLM port

analysis\_port=new("analysis\_port",this);

endfunction

function void oMonitor::connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

if (!uvm\_config\_db#(virtual memory\_if.tb\_mon\_out)::get(get\_parent(), "", "oMon\_if", vif)) begin

`uvm\_fatal(get\_type\_name(), "oMonitor DUT interface not set");

end

endfunction

task oMonitor::run\_phase(uvm\_phase phase);

// The job of the oMonitor is to passively monitor the physical signals,

// interprete and report the activities that it sees. In this case, to

// re-construct the packet that it sees on the DUT's output port as specified

forever begin

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

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

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

continue;

pkt = packet::type\_id::create("pkt",this);

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

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

`uvm\_info(get\_type\_name(),pkt.convert2string(),UVM\_MEDIUM);

analysis\_port.write(pkt);

end

endtask

**packet.sv**

`define AWIDTH 8

`define DWIDTH 32

class packet extends uvm\_sequence\_item;

rand logic [`AWIDTH - 1:0] addr;

rand logic [`DWIDTH - 1:0] data;

op\_type mode;

slv\_response\_type slv\_rsp;

bit [`AWIDTH - 1:0] prev\_addr;

bit [`DWIDTH - 1:0] prev\_data;

constraint valid {

addr inside {[0:15]};

data inside {[10:9999]};

data != prev\_data;

addr != prev\_addr;

}

function void post\_randomize();

prev\_addr=addr;

prev\_data=data;

endfunction

`uvm\_object\_utils\_begin(packet)

`uvm\_field\_int(addr,UVM\_ALL\_ON)

`uvm\_field\_int(data,UVM\_ALL\_ON)

`uvm\_object\_utils\_end

virtual function string convert2string();

return $sformatf("[%0s] addr=%0d data=%0d ",get\_type\_name(),this.addr,this.data);

endfunction

function new(string name="packet");

super.new(name);

endfunction

endclass

**reset\_sequence.sv**

class reset\_sequence extends base\_sequence;

`uvm\_object\_utils(reset\_sequence)

function new (string name="reset\_sequence");

super.new(name);

endfunction

extern virtual task body();

endclass

task reset\_sequence::body();

begin

`uvm\_create(req);

req.mode=RESET;

start\_item(req);

finish\_item(req);

`uvm\_info("RST\_SEQ","Reset Transaction Done \n",UVM\_MEDIUM);

end

endtask

**rw\_sequence.sv**

class rw\_sequence extends base\_sequence;

`uvm\_object\_utils(rw\_sequence)

function new (string name="rw\_sequence");

super.new(name);

endfunction

extern virtual task body();

endclass

task rw\_sequence::body();

bit [31:0] count;

REQ ref\_pkt;

ref\_pkt=packet::type\_id::create("ref\_pkt",,get\_full\_name());

repeat(pkt\_count) begin

`uvm\_create(req);

assert(ref\_pkt.randomize());

req.copy(ref\_pkt);

start\_item(req);

finish\_item(req);

get\_response(rsp);

count++;

`uvm\_info("RW\_SEQ",$sformatf("Transaction %0d Done with resp=%0s \n ",count,rsp.slv\_rsp.name()),UVM\_MEDIUM);

end

endtask

**scoreboard.sv**

class scoreboard extends uvm\_scoreboard;

`uvm\_component\_utils(scoreboard)

uvm\_analysis\_port #(packet) mon\_in;

uvm\_analysis\_port #(packet) mon\_out;

uvm\_in\_order\_class\_comparator #(packet) m\_comp;

function new(string name="scoreboard",uvm\_component parent=null);

super.new(name,parent);

endfunction

virtual function void build\_phase(uvm\_phase phase);

super.build\_phase(phase);

m\_comp=uvm\_in\_order\_class\_comparator#(packet)::type\_id::create("m\_comp",this);

mon\_in=new("mon\_in",this);

mon\_out=new("mon\_out",this);

endfunction

virtual function void connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

mon\_in.connect(m\_comp.before\_export);

mon\_out.connect(m\_comp.after\_export);

endfunction

virtual function void extract\_phase(uvm\_phase phase);

uvm\_config\_db#(int)::set(null,"uvm\_test\_top.env","matches",m\_comp.m\_matches);

uvm\_config\_db#(int)::set(null,"uvm\_test\_top.env","mis\_matches",m\_comp.m\_mismatches);

endfunction

function void report\_phase(uvm\_phase phase);

`uvm\_info("SCB",$sformatf("Scoreboard completed with matches=%0d mismatches=%0d ",m\_comp.m\_matches,m\_comp.m\_mismatches),UVM\_NONE);

endfunction

endclass

**sequencer.sv**

//typedef uvm\_sequencer #(packet) sequencer;

//OR use like below shown

class sequencer extends uvm\_sequencer#(packet);

`uvm\_component\_utils(sequencer);

function new (string name="sequencer",uvm\_component parent);

super.new(name,parent);

endfunction

endclass

**shutdown\_sequence.sv**

class shutdown\_sequence extends base\_sequence;

`uvm\_object\_utils(shutdown\_sequence)

bit [31:0] csr\_dropped;

function new (string name="shutdown\_sequence");

super.new(name);

endfunction

extern virtual task body();

endclass

task shutdown\_sequence::body();

`uvm\_create(req);

req.addr='h26;//addr of csr4\_dropped register in DUT

req.data='h0;

req.mode=CFG\_REG\_READ;

start\_item(req);

finish\_item(req);

csr\_dropped=req.data;

uvm\_config\_db#(bit [31:0])::set(get\_sequencer(),"","dropped\_count",csr\_dropped);

`uvm\_info("CONFIG\_SEQ",$sformatf("csr\_dropped\_count=%0d",req.data),UVM\_MEDIUM);

`uvm\_info("CONFIG\_SEQ","Shutdown sequence Transaction Done ",UVM\_MEDIUM);

endtask

**slave\_agent.sv**

class slave\_agent extends uvm\_agent;

`uvm\_component\_utils(slave\_agent)

oMonitor oMon;

uvm\_analysis\_port#(packet) ap;

function new (string name="slave\_agent",uvm\_component parent);

super.new(name,parent);

endfunction

extern virtual function void build\_phase(uvm\_phase phase);

extern virtual function void connect\_phase(uvm\_phase phase);

endclass

function void slave\_agent::build\_phase(uvm\_phase phase);

super.build\_phase(phase);

ap=new("slave\_ap",this);

oMon=oMonitor::type\_id::create("oMon",this);

endfunction

function void slave\_agent::connect\_phase(uvm\_phase phase);

super.connect\_phase(phase);

oMon.analysis\_port.connect(this.ap);

endfunction

**program\_mem.sv**

**`**include "mem\_env\_pkg.pkg"

program mem\_program(memory\_if pif);

import uvm\_pkg::\*;

import mem\_env\_pkg::\*;

`include "base\_test.sv"

`include "mem\_test1.sv"

initial begin

$timeformat(-9, 1, "ns", 10);

uvm\_config\_db#(virtual memory\_if.tb)::set(null,"uvm\_test\_top","master\_if",pif.tb);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_in)::set(null,"uvm\_test\_top","mon\_in",pif.tb\_mon\_in);

uvm\_config\_db#(virtual memory\_if.tb\_mon\_out)::set(null,"uvm\_test\_top","mon\_out",pif.tb\_mon\_out);

run\_test();

end

endprogram

**top.sv**

`include "if\_memory.sv"

`include "memory\_rtl.sv"

`include "program\_mem.sv"

module top;

bit clk;

always #10 clk=!clk;

memory\_if mem\_if (clk);

memory\_rtl dut\_inst (

.clk(clk),

.reset(mem\_if.reset),

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

.response(mem\_if.slv\_rsp),

.wdata(mem\_if.wdata),

.rdata(mem\_if.rdata));

mem\_program pgm\_inst (mem\_if);

endmodule

**runtime\_command\_lines**

/\*

./simv +UVM\_TESTNAME=my\_test -l log

#./simv +UVM\_TESTNAME=my\_test -l log +UVM\_NO\_RELNOTES +UVM\_VERBOSITY=UVM\_MEDIUM

#+uvm\_set\_verbosity=\*,\_ALL\_,UVM\_LOW

#+uvm\_set\_verbosity="\*.seq\*,\_ALL\_,UVM\_LOW,time,0"

#./simv +UVM\_TESTNAME=my\_test +uvm\_set\_verbosity=\*.iMon\*,\_ALL\_,UVM\_LOW,time,0

#./simv +UVM\_TESTNAME=my\_test +uvm\_set\_verbosity=\*.drv\*\*,\_ALL\_,UVM\_LOW,time,0

#./simv +UVM\_TESTNAME=my\_test +uvm\_set\_verbosity=\*.seq\*\*,\_ALL\_,UVM\_LOW,time,0

#./simv +UVM\_TESTNAME=my\_test +uvm\_set\_verbosity=\*.oMon\*,\_ALL\_,UVM\_LOW,time,0

#below setting will make uvm\_info inside the sequence not to display messages on screen

# ./simv +UVM\_TESTNAME=my\_test -l log +uvm\_set\_action=\*.seq\*,\_ALL\_,UVM\_INFO,UVM\_NO\_ACTION

#below one changes UVM\_ERROR to UVM\_WARNING for all ID's

#./simv +UVM\_TESTNAME=my\_test -l log +uvm\_set\_severity=\*,\_ALL\_,UVM\_ERROR,UVM\_WARNING

#below one changes UVM\_ERROR to UVM\_INFO for all ID's

#./simv +UVM\_TESTNAME=my\_test -l log +uvm\_set\_severity="\*,\_ALL\_,UVM\_ERROR,UVM\_INFO"

+UVM\_TESTNAME=mem\_test1 +UVM\_MAX\_QUIT\_COUNT=3

\*/