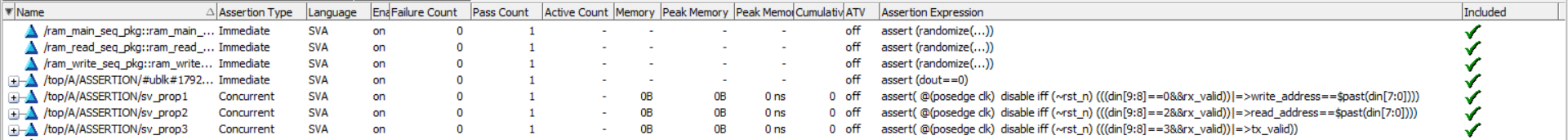


A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated



interface ram\_if (clk);

    parameter MEM\_DEPTH=256;

    parameter ADDR\_SIZE=8;

    input clk;

    logic  rst\_n, rx\_valid;

    logic [9:0] din;

    logic tx\_valid;

    logic [ADDR\_SIZE-1:0] dout;

endinterface

import uvm\_pkg::\*;

import ram\_test\_pkg::\*;

`include "uvm\_macros.svh"

module top ();

    bit clk;

    initial

    begin

        clk=0;

        forever

        #1 clk=~clk;

    end

    ram\_if R1(clk);

    SPI\_RAM A(R1.din, R1.dout ,R1.rx\_valid, R1.tx\_valid , clk, R1.rst\_n);

    bind SPI\_RAM RAM\_assertions ASSERTION (R1.din, R1.dout ,R1.rx\_valid, R1.tx\_valid ,clk, R1.rst\_n,A.write\_address,A.read\_address);

initial

begin

    uvm\_config\_db#(virtual ram\_if)::set(null, "uvm\_test\_top", "ram\_IF",R1);

    run\_test("ram\_test");

end

endmodule

package ram\_config\_pkg;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

class ram\_config\_obj extends uvm\_object;

`uvm\_object\_utils(ram\_config\_obj)

virtual ram\_if ram\_config\_vif;

function new(string name="ram\_config\_obj");

super.new(name);

endfunction

endclass

endpackage

module SPI\_RAM (din, dout ,rx\_valid, tx\_valid , clk, rst\_n);

parameter MEM\_DEPTH=256;

parameter ADDR\_SIZE=8;

input clk, rst\_n, rx\_valid;

input [9:0] din;

output  reg tx\_valid;

output reg [ADDR\_SIZE-1:0] dout;

reg [ADDR\_SIZE-1:0] write\_address, read\_address; //buses to hold write or read addresses

reg [ADDR\_SIZE-1:0] memory [MEM\_DEPTH-1:0];

always @(posedge clk or negedge rst\_n) begin

    if (~rst\_n) begin

        dout<=0;

        tx\_valid<=0;

    end

    else if (rx\_valid) begin

        case ( din [9:8])

          2'b00: begin

            write\_address<= din[ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b01: begin

            memory[write\_address] <= din [ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b10: begin

            read\_address<= din [ADDR\_SIZE-1:0];

            tx\_valid <= 0;

          end

          2'b11: begin

            dout<= memory[read\_address];

            tx\_valid<=1;

          end

        endcase

    end

end

endmodule

seq\_item

package ram\_seq\_item\_pkg;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    parameter MEM\_DEPTH=256;

parameter ADDR\_SIZE=8;

    class ram\_seq\_item extends uvm\_sequence\_item;

        `uvm\_object\_utils(ram\_seq\_item)

        //-> properties

        rand bit clk, rst\_n, rx\_valid;

        rand bit [9:0] din;

    logic tx\_valid;

        logic [ADDR\_SIZE-1:0] dout;

        logic [ADDR\_SIZE-1:0] write\_address, read\_address;

        bit WRITE\_SIG,READ\_SIG;

        rand bit WRITE,READ;

// -> Constraints

constraint reset { rst\_n dist {0:=5 , 1:=95};}

constraint RX\_VALID { rx\_valid dist {0:=20 , 1:=80}; }

constraint Din {

if(WRITE)

    {

if(WRITE\_SIG==1)

            din[9:8]==2'b01;

               else

                din[9:8]==2'b00;

               }

               else if(READ)

               {

                 if(READ\_SIG==0)

               din[9:8]==2'b10;

                else

               din[9:8]==2'b11;

                }

                }

constraint write\_only {

if(WRITE\_SIG==1)

din[9:8]==2'b01;

else

din[9:8]==2'b00;

}

constraint read\_only {

if(READ\_SIG==0)

 din[9:8]==2'b10;

else

din[9:8]==2'b11;

}

        function new(string name = "ram\_seq\_item");

            super.new(name);

        endfunction

        function string convert2string();

            return $sformatf("%s rst\_n=0b%0b, rx\_valid=0b%0b,din = 0b%0b,tx\_valid = 0b%0b, dout= 0b%0b",super.convert2string(),rst\_n,rx\_valid,din,tx\_valid,dout);

        endfunction

        function string convert2string\_stimulus();

return $sformatf("rst\_n=0b%0b, rx\_valid=0b%0b,din = 0b%0b,tx\_valid = 0b%0b, dout= 0b%0b",rst\_n,rx\_valid,din,tx\_valid,dout);

        endfunction

    endclass

endpackage

reset\_seq

package ram\_reset\_seq\_pkg;

    import ram\_seq\_item\_pkg::\*;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_reset\_sequence extends  uvm\_sequence #(ram\_seq\_item);

        `uvm\_object\_utils(ram\_reset\_sequence)

        ram\_seq\_item seq\_item;

        function new(string name = "ram\_reset\_sequence");

            super.new(name);

        endfunction

        task body;

            seq\_item = ram\_seq\_item::type\_id::create("seq\_item");

            repeat(5)

            begin

            start\_item(seq\_item);

            seq\_item.rst\_n = 0;

            seq\_item.din = 0;

            seq\_item.rx\_valid = 0;

            finish\_item(seq\_item);

        end

        endtask

    endclass

endpackage

read only seq

package ram\_read\_seq\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_read\_sequence extends  uvm\_sequence #(ram\_seq\_item);

        `uvm\_object\_utils(ram\_read\_sequence)

        ram\_seq\_item seq\_item;

        function new(string name = "ram\_reset\_sequence");

            super.new(name);

        endfunction

//->  read only sequence

        task body;

    seq\_item = ram\_seq\_item::type\_id::create("seq\_item");

    seq\_item.WRITE\_SIG=0;

    seq\_item.READ\_SIG=0;

    seq\_item.write\_only.constraint\_mode(0);

    seq\_item.Din.constraint\_mode(0);

            repeat(1000)

            begin

            start\_item(seq\_item);

assert(seq\_item.randomize());

if(seq\_item.din[9:8]==2'b10)

seq\_item.READ\_SIG=1;

else if(seq\_item.din[9:8]==2'b11)

    seq\_item.READ\_SIG=0;

    finish\_item(seq\_item);

            end

        endtask

    endclass

endpackage

Write only seq

package ram\_write\_seq\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_write\_sequence extends  uvm\_sequence #(ram\_seq\_item);

        `uvm\_object\_utils(ram\_write\_sequence)

        ram\_seq\_item seq\_item;

        function new(string name = "ram\_reset\_sequence");

            super.new(name);

        endfunction

//-> write only sequence

        task body;

    seq\_item = ram\_seq\_item::type\_id::create("seq\_item");

    seq\_item.WRITE\_SIG=0;

    seq\_item.READ\_SIG=0;

    seq\_item.read\_only.constraint\_mode(0);

    seq\_item.Din.constraint\_mode(0);

            repeat(1000)

            begin

            start\_item(seq\_item);

assert(seq\_item.randomize());

if(seq\_item.din[9:8]==2'b00)

seq\_item.WRITE\_SIG=1;

else if(seq\_item.din[9:8]==2'b01)

    seq\_item.WRITE\_SIG=0;

    finish\_item(seq\_item);

    end

        endtask

    endclass

endpackage

main\_seq

package ram\_main\_seq\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_main\_sequence extends  uvm\_sequence #(ram\_seq\_item);

        `uvm\_object\_utils(ram\_main\_sequence)

        ram\_seq\_item seq\_item;

        function new(string name = "ram\_reset\_sequence");

            super.new(name);

        endfunction

//-> write and read sequence

        task body;

    seq\_item = ram\_seq\_item::type\_id::create("seq\_item");

    seq\_item.WRITE\_SIG=0;

    seq\_item.READ\_SIG=0;

    seq\_item.read\_only.constraint\_mode(0);

    seq\_item.write\_only.constraint\_mode(0);

            repeat(50000)

            begin

            start\_item(seq\_item);

assert(seq\_item.randomize());

if(seq\_item.din[9:8]==2'b00)

seq\_item.WRITE\_SIG=1;

else if(seq\_item.din[9:8]==2'b01)

    seq\_item.WRITE\_SIG=0;

if(seq\_item.din[9:8]==2'b10)

    seq\_item.READ\_SIG=1;

else if(seq\_item.din[9:8]==2'b11)

    seq\_item.READ\_SIG=0;

    finish\_item(seq\_item);

            end

        endtask

    endclass

endpackage

sequencer

package my\_sequencer\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class MySequencer extends  uvm\_sequencer #(ram\_seq\_item);

        `uvm\_component\_utils(MySequencer)

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

            super.new(name,parent);

        endfunction

    endclass

endpackage

monitor

package ram\_monitor\_pkg;

import uvm\_pkg::\*;

import ram\_seq\_item\_pkg::\*;

`include "uvm\_macros.svh"

class ram\_monitor extends uvm\_monitor;

`uvm\_component\_utils(ram\_monitor)

    virtual ram\_if ram\_vif;

ram\_seq\_item rsp\_seq\_item;

uvm\_analysis\_port #(ram\_seq\_item)mon\_ap;

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

    super.new(name,parent);

endfunction

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

    super.build\_phase(phase);

    mon\_ap=new("mon\_ap",this);

endfunction

task run\_phase (uvm\_phase phase);

    super.run\_phase(phase);

    forever

    begin

        rsp\_seq\_item= ram\_seq\_item::type\_id::create("rsp\_seq\_item");

        @(negedge ram\_vif.clk);

        rsp\_seq\_item.rst\_n=ram\_vif.rst\_n;

        rsp\_seq\_item.din=ram\_vif.din;

        rsp\_seq\_item.tx\_valid=ram\_vif.tx\_valid;

        rsp\_seq\_item.rx\_valid=ram\_vif.rx\_valid;

        rsp\_seq\_item.dout=ram\_vif.dout;

mon\_ap.write(rsp\_seq\_item);

`uvm\_info("run\_phase",rsp\_seq\_item.convert2string(),UVM\_HIGH)

end

endtask

endclass

endpackage

scoreboard

package ram\_scoreboard\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_scoreboard extends uvm\_scoreboard;

        `uvm\_component\_utils(ram\_scoreboard)

        uvm\_analysis\_export #(ram\_seq\_item) sb\_export;

        uvm\_tlm\_analysis\_fifo #(ram\_seq\_item) sb\_fifo;

        ram\_seq\_item seq\_item\_sb;

        logic [ADDR\_SIZE-1:0] dout\_ref;

        logic tx\_valid\_ref;

        logic [ADDR\_SIZE-1:0] write\_address,read\_address;

        logic  [ADDR\_SIZE-1:0] mem[MEM\_DEPTH-1:0];

        int error\_count = 0;

        int correct\_count = 0;

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

            super.new(name,parent);

        endfunction

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

        super.build\_phase(phase);

        sb\_export=new("sb\_export",this);

        sb\_fifo=new("sb\_fifo",this);

        endfunction

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

            super.connect\_phase(phase);

            sb\_export.connect(sb\_fifo.analysis\_export);

        endfunction

        task run\_phase(uvm\_phase phase);

            super.run\_phase(phase);

            forever

            begin

            sb\_fifo.get(seq\_item\_sb);

            ref\_model(seq\_item\_sb);

            if(seq\_item\_sb.dout != dout\_ref || seq\_item\_sb.tx\_valid != tx\_valid\_ref )

                begin

            `uvm\_error("run\_phase",$sformatf("Comparison failed, transaction received by the DUT:%s while the refrence dout:0b%0b,tx\_valid:0b%0b",seq\_item\_sb.convert2string(),dout\_ref,tx\_valid\_ref));

                    error\_count++;

            end

                else

                begin

                `uvm\_info("run\_phase",$sformatf("correct out: %s",seq\_item\_sb.convert2string()),UVM\_HIGH);

                    correct\_count++;

                end

            end

        endtask

task ref\_model(ram\_seq\_item seq\_item\_chk);

            if (~seq\_item\_chk.rst\_n) begin

        dout\_ref=0;

        tx\_valid\_ref=0;

    end

    else if (seq\_item\_chk.rx\_valid) begin

        case ( seq\_item\_chk.din [9:8])

          2'b00: begin

            write\_address= seq\_item\_chk.din[ADDR\_SIZE-1:0];

            tx\_valid\_ref = 0;

          end

          2'b01: begin

            mem[write\_address] = seq\_item\_chk.din [ADDR\_SIZE-1:0];

            tx\_valid\_ref = 0;

          end

          2'b10: begin

            read\_address= seq\_item\_chk.din [ADDR\_SIZE-1:0];

            tx\_valid\_ref = 0;

          end

          2'b11: begin

            dout\_ref= mem[read\_address];

            tx\_valid\_ref=1;

          end

        endcase

            end

        endtask

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

            super.report\_phase(phase);

            `uvm\_info("report\_phase",$sformatf("Total successful transactions:%0d",correct\_count),UVM\_MEDIUM);

            `uvm\_info("report\_phase",$sformatf("Total failed transactions:%0d",error\_count),UVM\_MEDIUM);

        endfunction

    endclass

endpackage

coverage collector

package ram\_coverage\_collector\_pkg ;

import ram\_seq\_item\_pkg::\*;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

class ram\_coverage\_collector extends  uvm\_component;

    `uvm\_component\_utils(ram\_coverage\_collector)

uvm\_analysis\_export#(ram\_seq\_item)cov\_export;

uvm\_tlm\_analysis\_fifo#(ram\_seq\_item)cov\_fifo;

ram\_seq\_item seq\_item\_cov;

covergroup RAM\_cvr();

Dout:coverpoint seq\_item\_cov.dout;

Din:coverpoint seq\_item\_cov.din[9:8]{

bins Write\_address= {2'b00};

bins Write\_data={2'b01};

bins read\_address={2'b10};

bins read\_data={2'b11};

}

DATA\_IN:coverpoint seq\_item\_cov.din;

TX:coverpoint seq\_item\_cov.tx\_valid

{

 bins ZERO={0};

 bins ONE ={1};

}

RX:coverpoint seq\_item\_cov.rx\_valid

{

 bins ZERO={0};

 bins ONE ={1};

}

/\*w\_address:coverpoint ram\_seq\_item.write\_address;

R\_address:coverpoint ram\_seq\_item.read\_address;\*/

cross\_1 : cross TX, Din

{option.cross\_auto\_bin\_max=0;

    bins tx\_valid\_din= binsof(Din.read\_data) && binsof(TX.ONE);

  }

// Cross\_2:cross w\_address,RX

// {option.cross\_auto\_bin\_max=0;

//   bins RX\_W\_ADDRESS= binsof(W\_address) && binsof(RX);

// }

/\*Cross\_3: R\_address,RX

{option.cross\_auto\_bin\_max=0;

  bins RX\_W\_ADDRESS= binsof(R\_address) && binsof(RX);

}\*/

Cross\_3: cross DATA\_IN,RX

{option.cross\_auto\_bin\_max=0;

  bins RX\_W\_ADDRESS= binsof(DATA\_IN) && binsof(RX.ONE);

}

endgroup

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

    super.new(name,parent);

    RAM\_cvr=new();

endfunction

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

            super.build\_phase(phase);

            cov\_export = new("cov\_export",this);

            cov\_fifo = new("cov\_fifo",this);

        endfunction

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

            super.connect\_phase(phase);

            cov\_export.connect(cov\_fifo.analysis\_export);

        endfunction

        task run\_phase(uvm\_phase phase);

            super.run\_phase(phase);

            forever begin

                cov\_fifo.get(seq\_item\_cov);

                RAM\_cvr.sample();

            end

        endtask

endclass

endpackage

driver

package ram\_driver\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

class ram\_driver extends uvm\_driver #(ram\_seq\_item);

    `uvm\_component\_utils(ram\_driver)

    virtual ram\_if ram\_driver\_vif;

    ram\_seq\_item stim\_seq\_item;

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

        super.new(name,parent);

    endfunction

    task run\_phase(uvm\_phase phase);

        super.run\_phase(phase);

        forever

        begin

            stim\_seq\_item = ram\_seq\_item::type\_id::create("stim\_seq\_item");

            seq\_item\_port.get\_next\_item(stim\_seq\_item);

                ram\_driver\_vif.rx\_valid = stim\_seq\_item.rx\_valid;

                ram\_driver\_vif.rst\_n = stim\_seq\_item.rst\_n;

                ram\_driver\_vif.din = stim\_seq\_item.din;

                @(negedge ram\_driver\_vif.clk);

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

                `uvm\_info("run\_phase",stim\_seq\_item.convert2string\_stimulus(),UVM\_HIGH)

        end

      endtask

    endclass

endpackage

agent

package ram\_agent\_pkg;

import uvm\_pkg::\*;

import ram\_config\_pkg::\*;

import ram\_driver\_pkg::\*;

import my\_sequencer\_pkg::\*;

import ram\_monitor\_pkg::\*;

import ram\_seq\_item\_pkg::\*;

`include "uvm\_macros.svh"

class ram\_agent extends uvm\_agent;

 `uvm\_component\_utils(ram\_agent)

 MySequencer sqr;

 ram\_config\_obj ram\_cfg;

 uvm\_analysis\_port#(ram\_seq\_item) agt\_ap;

 ram\_driver drv;

 ram\_monitor mon;

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

    super.new(name,parent);

endfunction

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

    super.build\_phase(phase);

    uvm\_config\_db #(ram\_config\_obj)::get(this,"","ram\_IF1",ram\_cfg);

sqr= MySequencer::type\_id::create("sqr",this);

drv= ram\_driver::type\_id::create("drv",this);

mon= ram\_monitor::type\_id::create("mon",this);

agt\_ap=new("agt\_ap",this);

endfunction

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

    drv.ram\_driver\_vif=ram\_cfg.ram\_config\_vif;

    mon.ram\_vif=ram\_cfg.ram\_config\_vif;

    drv.seq\_item\_port.connect(sqr.seq\_item\_export);

    mon.mon\_ap.connect(agt\_ap);

endfunction

endclass

endpackage

driver

package ram\_driver\_pkg;

    import uvm\_pkg::\*;

    import ram\_seq\_item\_pkg::\*;

    `include "uvm\_macros.svh"

class ram\_driver extends uvm\_driver #(ram\_seq\_item);

    `uvm\_component\_utils(ram\_driver)

    virtual ram\_if ram\_driver\_vif;

    ram\_seq\_item stim\_seq\_item;

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

        super.new(name,parent);

    endfunction

    task run\_phase(uvm\_phase phase);

        super.run\_phase(phase);

        forever

        begin

            stim\_seq\_item = ram\_seq\_item::type\_id::create("stim\_seq\_item");

            seq\_item\_port.get\_next\_item(stim\_seq\_item);

                ram\_driver\_vif.rx\_valid = stim\_seq\_item.rx\_valid;

                ram\_driver\_vif.rst\_n = stim\_seq\_item.rst\_n;

                ram\_driver\_vif.din = stim\_seq\_item.din;

                @(negedge ram\_driver\_vif.clk);

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

                `uvm\_info("run\_phase",stim\_seq\_item.convert2string\_stimulus(),UVM\_HIGH)

        end

      endtask

    endclass

endpackage

environmentenvy

package ram\_env\_pkg;

    import uvm\_pkg::\*;

    import ram\_agent\_pkg::\*;

    import ram\_scoreboard\_pkg::\*;

    import ram\_coverage\_collector\_pkg::\*;

    `include "uvm\_macros.svh"

    class ram\_env extends uvm\_env;

        `uvm\_component\_utils(ram\_env)

        ram\_agent agt;

        ram\_scoreboard sb;

        ram\_coverage\_collector cov;

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

            super.new(name,parent);

        endfunction

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

            super.build\_phase(phase);

            agt=ram\_agent::type\_id::create("agt",this);

            sb=ram\_scoreboard::type\_id::create("sb",this);

            cov=ram\_coverage\_collector::type\_id::create("cov",this);

    endfunction

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

            agt.agt\_ap.connect(sb.sb\_export);

            agt.agt\_ap.connect(cov.cov\_export);

    endfunction

endclass

    endpackage

assertions

module RAM\_assertions (din, dout ,rx\_valid, tx\_valid , clk, rst\_n,write\_address,read\_address);

parameter MEM\_DEPTH=256;

parameter ADDR\_SIZE=8;

input clk, rst\_n, rx\_valid;

input [9:0] din;

input tx\_valid;

input [ADDR\_SIZE-1:0] dout;

input [ADDR\_SIZE-1:0] write\_address, read\_address;

property prop\_1;

@(posedge clk) disable iff(!rst\_n) (din[9:8]==2'b00 &&rx\_valid)|=> write\_address==$past(din[7:0]);

endproperty

property prop\_2;

@(posedge clk) disable iff(!rst\_n) (din[9:8]==2'b10 && rx\_valid) |=>read\_address==$past(din[7:0]);

    endproperty

property prop\_3;

        @(posedge clk) disable iff(!rst\_n) (din[9:8]==2'b11 && rx\_valid) |=>  (tx\_valid==1);

    endproperty

    always\_comb

begin

if(!rst\_n)

assert final(dout==0);

end

sv\_prop1:assert property(prop\_1);

sv\_prop2:assert property(prop\_2);

sv\_prop3:assert property(prop\_3);

cvr\_prop1:cover property(prop\_1);

cvr\_prop2:cover property(prop\_2);

cvr\_prop3:cover property(prop\_3);

endmodule

test

package ram\_test\_pkg;

import uvm\_pkg::\*;

 import ram\_env\_pkg::\*;

 import ram\_config\_pkg::\*;

 import ram\_main\_seq\_pkg::\*;

 import ram\_reset\_seq\_pkg::\*;

import ram\_write\_seq\_pkg::\*;

import ram\_read\_seq\_pkg::\*;

 `include "uvm\_macros.svh"

class ram\_test extends uvm\_test;

    `uvm\_component\_utils(ram\_test)

    ram\_env env;

    ram\_config\_obj ram\_config\_obj\_test;

    ram\_main\_sequence main\_seq;

    ram\_reset\_sequence reset\_seq;

    ram\_write\_sequence write\_seq;

    ram\_read\_sequence read\_seq;

    virtual ram\_if ram\_vif;

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

        super.new(name,parent);

    endfunction

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

        super.build\_phase (phase);

        main\_seq=ram\_main\_sequence::type\_id:: create("main\_seq",this);

        reset\_seq=ram\_reset\_sequence ::type\_id:: create("reset\_seq",this);

        write\_seq=ram\_write\_sequence::type\_id:: create("write\_seq",this);

        read\_seq=ram\_read\_sequence ::type\_id:: create("read\_seq",this);

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

        ram\_config\_obj\_test=ram\_config\_obj::type\_id:: create("ram\_config\_obj\_test",this);

        uvm\_config\_db#(virtual ram\_if)::get(this, "", "ram\_IF",ram\_config\_obj\_test.ram\_config\_vif);

        uvm\_config\_db#(ram\_config\_obj)::set(this, "\*", "ram\_IF1",ram\_config\_obj\_test);

    endfunction

    task run\_phase(uvm\_phase phase);

        super.run\_phase(phase);

        phase.raise\_objection(this);

        `uvm\_info("run\_phase","reset\_asserted",UVM\_LOW)

        reset\_seq.start(env.agt.sqr);

    `uvm\_info("run\_phase","reset\_desserted",UVM\_LOW)

   `uvm\_info("run\_phase","write\_only\_asserted",UVM\_LOW)

    write\_seq.start(env.agt.sqr);

    `uvm\_info("run\_phase","write\_only\_desserted",UVM\_LOW)

`uvm\_info("run\_phase","read\_only\_asserted",UVM\_LOW)

 read\_seq.start(env.agt.sqr);

 `uvm\_info("run\_phase","read\_only\_desserted",UVM\_LOW)

 `uvm\_info("run\_phase","read\_and\_write\_asserted",UVM\_LOW)

 main\_seq.start(env.agt.sqr);

 `uvm\_info("run\_phase","read\_and\_write\_desserted",UVM\_LOW)

        phase.drop\_objection(this);

    endtask

endclass

endpackage