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_sequenc
e",reset_sequence::get_type());
uvm_config_db#(uvm_object_wrapper)::set(this,"env.m_agent.seqr.configure_phase","default_seq
uence",config_sequence::get_type());
uvm_config_db#(uvm_object_wrapper)::set(this,"env.m_agent.seqr.main_phase","default_sequenc
e",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_sequenc
e",reset_sequence::get_type());

uvm_config_db#(uvm_object_wrapper)::set(this,"env.m_agent.seqr.configure_phase","default_seq
uence",config_sequence::get_type());

uvm_config_db#(uvm_object_wrapper)::set(this,"env.m_agent.seqr.main_phase","default_sequenc
e",rw_sequence::get_type());

uvm_config_db#(uvm_object_wrapper)::set(this,"env.m_agent.seqr.shutdown_phase","default_seq
uence",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

*/
```