VERIFCATION MODEL FOR MEMORY

```
//------------------------------------------------------------------------
//                                      design.sv
//------------------------------------------------------------------------
module memory #( parameter ADDR_WIDTH = 3,
                 parameter DATA_WIDTH = 8 ) (
   input clk,
   input reset,

   //control signals
   input [ADDR_WIDTH-1:0]  addr,
   input                   wr_en,
   input                   rd_en,

   //data signals
   input  [DATA_WIDTH-1:0] wdata,
   output [DATA_WIDTH-1:0] rdata
  );

  reg [DATA_WIDTH-1:0] r_data;

  //Memory
  reg [DATA_WIDTH-1:0] mem [2**ADDR_WIDTH];

  //Reset
  always @(posedge reset)
    for(int i=0;i<2**ADDR_WIDTH;i++) mem[i]=8'hFF;

  // Write data to Memory
  always @(posedge clk)
```

```
  if (wr_en)    mem[addr] <= wdata;

 // Read data from memory
 always @(posedge clk)
  if (rd_en) r_data <= mem[addr];

endmodule
//-------------------------------------------------------------------------------------------------------------------------
Interface code:
interface mem_intf(input logic clk,reset);

 //declaring the signals
 logic [2:0] addr;
 logic wr_en;
 logic rd_en;
 logic [7:0] wdata;
 logic [7:0] rdata;

 //driver clocking block
 clocking driver_cb @(posedge clk);
  default input #1 output #1;
  output addr;
  output wr_en;
  output rd_en;
  output wdata;
  input  rdata;
 endclocking

 //monitor clocking block
 clocking monitor_cb @(posedge clk);
  default input #1 output #1;
```

```
  input addr;
  input wr_en;
  input rd_en;
  input wdata;
  input rdata;
 endclocking

 //driver modport
 modport DRIVER  (clocking driver_cb,input clk,reset);

 //monitor modport
 modport MONITOR (clocking monitor_cb,input clk,reset);

endinterface
//-------------------------------------------------------------------------------------------------------------------------
```

Transaction class code:

```
//access type for transaction
typedef enum{
  RD,
  WR
}access_type_t;

class transaction;
  //declaring the transaction items
  rand bit [2:0] addr;
  rand bit       wr_en;
  rand bit       rd_en;
  rand bit [7:0] wdata;
```

```
    bit [7:0] rdata;
    bit [2:0] cnt;

 //constaint, to generate any one among write and read
 constraint wr_rd_c { wr_en != rd_en; };

 //postrandomize function, displaying randomized values of items
 function void post_randomize();
  $display("--------- [Trans] post_randomize ------");
  $display("\t addr  = %0h",addr);
  if(wr_en) $display("\t wr_en = %0h\t wdata = %0h",wr_en,wdata);
  if(rd_en) $display("\t rd_en = %0h\t rdata = %0h",rd_en,rdata);
  $display("-----------------------------------------");
 endfunction

endclass
//------------------------------------------------------------------------------
class rd_trans extends transaction;
  static bit[2:0] cnt;

  function void pre_randomize();
    wr_en.rand_mode(0);
    rd_en.rand_mode(0);
    addr.rand_mode(0);
     wr_en = 0;
     rd_en = 1;
     cnt  = this.cnt;
    cnt++;
  endfunction

endclass
```

```
//-------------------------------------------------------------------------
class wr_trans extends transaction;
   static bit[2:0] cnt;

   function void pre_randomize();
     wr_en.rand_mode(0);
     rd_en.rand_mode(0);
     addr.rand_mode(0);
       wr_en = 1;
       rd_en = 0;
       cnt  = this.cnt;
     cnt++;
   endfunction

endclass
//-------------------------------------------------------------------------------------------------------------------
Generator class code:
//`include "transaction.sv"
class generator;
  //access type
  access_type_t acc;

  //declaring transaction class
   transaction driv_trans;
   transaction score_trans;

  //declaring wr and rd trans
  wr_trans wtrans;
  rd_trans rtrans;

  //repeat count, to specify number of items to generate
```

```
  int repeat_count;
  int count;

  //mailbox, to generate and send the packet to driver and scoreboard
  mailbox gen2driv;
  mailbox gen2score; // new

  //event
  event ended;

  //constructor
  function new(mailbox gen2driv, mailbox gen2score, access_type_t acc); // changed
    //getting the mailbox handles from env, in order to share the transaction packet between the generator and driver, the same mailbox is shared between both.
    this.gen2driv = gen2driv;
    this.gen2score = gen2score; // new
    this.acc = acc; // new
    driv_trans = new();
    score_trans = new();
    count = 0;
  endfunction

  //main task, generates(create and randomizes) the repeat_count number of transaction packets and puts into mailbox
  task main();
    repeat(repeat_count) begin
      if(count == 8) begin
        $display("Got in here!");
        case (acc)
          RD: acc = WR;
          WR: acc = RD;
        endcase
```

```
    end
    case (acc)
     RD: begin
       $display("RD transaction!!");
       rtrans = new();
       driv_trans = rtrans;
       end
     WR: begin
       $display("WR transaction!!");
       wtrans = new();
       driv_trans = wtrans;
       end
    endcase
    if( !driv_trans.randomize() ) $fatal("Gen:: trans randomization failed");
    gen2driv.put(driv_trans);
    score_trans = driv_trans;
    gen2score.put(score_trans);
    count++;
  end
  -> ended;
 endtask

endclass
```

//---

Driver class code:

//gets the packet from generator and drive the transaction paket items into interface (interface is connected to DUT, so the items driven into interface signal will get driven in to DUT)

```
//`include "transaction.sv"
`define DRIV_IF mem_vif.DRIVER.driver_cb
class driver;
```

```
//used to count the number of transactions
int no_transactions;

//creating virtual interface handle
virtual mem_intf mem_vif;

//creating mailbox handle
mailbox gen2driv;

//constructor
function new(virtual mem_intf mem_vif, mailbox gen2driv);
  //getting the interface
  this.mem_vif = mem_vif;
  //getting the mailbox handles from  environment
  this.gen2driv = gen2driv;
endfunction

//Reset task, Reset the Interface signals to default/initial values
task reset;
  wait(mem_vif.reset);
  $display("--------- [DRIVER] Reset Started ---------");
  `DRIV_IF.wr_en <= 0;
  `DRIV_IF.rd_en <= 0;
  `DRIV_IF.addr  <= 0;
  `DRIV_IF.wdata <= 0;
  wait(!mem_vif.reset);
  $display("--------- [DRIVER] Reset Ended ---------");
endtask

// Simple write task to test rd test
```

```
task wr_sample_data;
  $display("--------- [DRIVER] Sample_write Started ---------");
  for(int i = 0; i < 4; ++i) begin
    `DRIV_IF.wr_en <= 0;
    @(posedge mem_vif.DRIVER.clk);
    `DRIV_IF.addr <= i;
    `DRIV_IF.wr_en <= 1;
    `DRIV_IF.wdata <= i**2;
    $display("\tADDR = %0h \tWDATA = %0h", i, i**2);
    @(posedge mem_vif.DRIVER.clk);
  end
  $display("--------- [DRIVER] Sample_write Ended ---------");
endtask

//drivers the transaction items to interface signals
task main;
  forever begin
    transaction trans;
    `DRIV_IF.wr_en <= 0;
    `DRIV_IF.rd_en <= 0;
    gen2driv.get(trans);
    $display("--------- [DRIVER-TRANSFER: %0d] ---------",no_transactions);
    @(posedge mem_vif.DRIVER.clk);
      `DRIV_IF.addr <= trans.addr;
    if(trans.wr_en) begin
      `DRIV_IF.wr_en <= trans.wr_en;
      `DRIV_IF.wdata <= trans.wdata;
      $display("\tADDR = %0h \tWDATA = %0h",trans.addr,trans.wdata);
      @(posedge mem_vif.DRIVER.clk);
    end
    if(trans.rd_en) begin
```

```
      `DRIV_IF.rd_en <= trans.rd_en;
      @(posedge mem_vif.DRIVER.clk);
      `DRIV_IF.rd_en <= 0;
      @(posedge mem_vif.DRIVER.clk);
      trans.rdata = `DRIV_IF.rdata;
      $display("\tADDR = %0h \tRDATA = %0h",trans.addr,`DRIV_IF.rdata);
    end
    $display("----------------------------------------");
    no_transactions++;
  end
 endtask

endclass
//-------------------------------------------------------------------------------------------------------------------
```

Monitor class code:

```
// The test is modified to write to 8 mem addresses then read from those
// The functional coverage covers rdata, wdata, and addr port
// The command to compile the program: vcs -sverilog -ntb_opts dtm tb_top.sv dut.sv
// The command to generate the coverage report: urg -dir simv.vdb
// The reports are generated in the urgReport folder in html format

//`include "transaction.sv"
`define MON_IF mem_vif.MONITOR.monitor_cb
class monitor;
  //Mailbox handle
  mailbox mon2score;

  //Transaction handle
  transaction trans;
```

```
//creating virtual interface handle
virtual mem_intf mem_vif;

//Define covergroup
//Updated the number of bins to cover 4 bit width mem
covergroup CovMem;
   coverpoint trans.addr {
      bins zero = {0};
      bins one = {1};
      bins two = {2};
      bins three = {3};
      bins four = {4};
      bins five = {5};
      bins six = {6};
      bins seven = {7};
   }
   coverpoint trans.rd_en {
      bins on = {1};
      bins off = {0};
   }
   coverpoint trans.wr_en {
      bins on = {1};
      bins off = {0};
   }
endgroup

//constructor
function new(virtual mem_intf mem_vif, mailbox mon2score);
   //getting the interface
   this.mem_vif = mem_vif;
```

```
    //getting the mailbox handles from  environment
    this.mon2score = mon2score;
    CovMem = new();
  endfunction

  function get_signals(transaction trans);
       trans.addr  = `MON_IF.addr;
       trans.wdata = `MON_IF.wdata;
       trans.rdata = `MON_IF.rdata;
      // trans.rd_en = `MON_IF.rd_en;
      // trans.wr_en = `MON_IF.wr_en;
  endfunction

  task main;
    forever begin
      @(posedge mem_vif.MONITOR.clk);
      if(`MON_IF.rd_en || `MON_IF.wr_en) begin
        trans = new();
        if(`MON_IF.rd_en)   trans.rd_en = 1;
        else if (`MON_IF.wr_en) trans.wr_en = 1;
        @(posedge mem_vif.MONITOR.clk);
        get_signals(trans);
        mon2score.put(trans);
        $display("MONITOR --> SCORE");
        CovMem.sample();
      end
    end
  endtask

endclass
//-------------------------------------------------------------------------------------------------------------------
```

Scoreboard class code:

```
//`include "transaction.sv"

// get the transaction from generator and compare with the output from the DUT to
class scoreboard;
        transaction gen_trans;
        transaction mon_trans;
 //score
 bit[4:0] score;

 //used to count the number of transactions
 int no_transactions;

 //creating mailbox handles
 mailbox gen2score;
 mailbox mon2score;

 //constructor
 function new(mailbox mon2score, mailbox gen2score); //changed
  //getting the mailbox handles from  environment
  this.gen2score = gen2score;
  this.mon2score = mon2score;
 endfunction

// scoreboard get the transaction and compare with the transaction from monitor
 task main;
  forever begin
     score = 4'b0;
        mon2score.get(mon_trans);
```

```
      gen2score.get(gen_trans);
      no_transactions++;
      if(gen_trans.wr_en) begin
        score[0] = ((gen_trans.wr_en == mon_trans.wr_en) ? 1 : 0);
        score[1] = ((gen_trans.addr == mon_trans.addr) ? 1 : 0);
        score[2] = ((gen_trans.wdata == mon_trans.wdata) ? 1 : 0);
        score[3] = ((gen_trans.cnt == mon_trans.cnt) ? 1 : 0);
        if(score == 15) $display("Transaction %0d checked!!", no_transactions);
        else $display("Error transaction %0d! -- %0b", no_transactions, score);
      end

           else begin
       $display("You are here");
      end
   end
  endtask

endclass
//-------------------------------------------------------------------------------------------------------------------
Base test (prog blk) code


`include "env_mem.sv"
program test(mem_intf intf);
  //declaring environment instance
  environment env;
  access_type_t acc;

  initial begin
    //creating environment
    env = new(intf);
```

```
  //setting the repeat count of generator as 4, means to generate 4 packets
  env.gen.repeat_count = 16;

  env.gen.acc = WR;

  //calling run of env, it interns calls generator and driver main tasks.
  env.run();

  //run 2nd time for writing test
  //env.gen.repeat_count = 4;

  //env.gen.acc= WR;

  //calling run of env, it interns calls generator and driver main tasks.
  //env.run();
 end
endprogram
//-----------------------------------------------------------------------------------------------------------------------
```

Top module code

```
//-----------------------------------------------------------------------
//tbench_top or testbench top, this is the top most file, in which DUT(Design Under Test) and Verification environment are connected.
//-----------------------------------------------------------------------

//including interfcae and testcase files

`include "intf_mem.sv"
```

```
//-------------------------[NOTE]---------------------------------
//Particular testcase can be run by uncommenting, and commenting the rest
//`include "random_test.sv"
//`include "wr_rd_test.sv"
//------------------------------------------------------------------

`include "base_test_mem.sv"

module top_module;

 //clock and reset signal declaration
 bit clk;
 bit reset;

 //clock generation
 always #5 clk = ~clk;

 //reset Generation
 initial begin
  reset = 1;
  #5 reset =0;
 end



 //creatinng instance of interface, in order to connect DUT and testcase
 mem_intf intf(clk,reset);

 //Testcase instance, interface handle is passed to test as an argument
 test t1(intf);
```

```
 //DUT instance, interface signals are connected to the DUT ports
 memory DUT (
  .clk(intf.clk),
  .reset(intf.reset),
  .addr(intf.addr),
  .wr_en(intf.wr_en),
  .rd_en(intf.rd_en),
  .wdata(intf.wdata),
  .rdata(intf.rdata)
 );

 //enabling the wave dump
 initial begin
  $dumpfile("dump.vcd"); $dumpvars;
 end
endmodule
```