RAM Design

Design bugs

- RAM bugs:

```
15      integer i;
16      always @(posedge clk or negedge rst_n) begin
17          if (!rst_n) begin
18             dout <= 0;
19             for(i=0; i<MEM_DEPTH; i=i+1)
20              mem[i] <= 0;
21          end
22          else if (rx_valid) begin
23             tx_valid <= 0;
24             case (din[9:8])
25              2'b00: addr_wr <= din[7:0];
26              2'b01: mem[addr_wr] <= din[7:0];
27              2'b10: addr_rd <= din[7:0];
28              2'b11: {dout, tx_valid} <= {mem[addr_rd], 1'b1};
29             endcase
30          end
31      end
32
```

1) rst_n must be Async.
2) rst_n doesn't clear the RAM.
3) When rest asserted the internal signals (addr_rd,addr_wr) and output signal (dout and tx_valid) must equal zeros.

Handling:

```
19      always @(posedge clk ) begin // first bug : rst must be syncrouns no Ashync
20          if (!rst_n) begin
21             dout <= 0;
22             tx_valid <= 0; //second bug : tx must take zero when rst_n is asserted
23             wr_addr <= 0 ;
24             rd_addr <= 0 ;//3th bug : when rst_n asserted , addr and not 2 internal signals it only one
25          end
26          else if (rx_valid) begin
27             tx_valid <= 0;
28              case (din[9:8])
29              2'b00:  wr_addr  <= din[7:0];
30              2'b01: mem[wr_addr] <= din[7:0];
31              2'b10: rd_addr <= din[7:0];
32              default: {dout, tx_valid} <= {mem[rd_addr], 1'b1};
33             endcase
34          end
35          else begin
36              tx_valid  <= 0 ;
37          end
38           //4th bug : when rx_valid not asserted then tx_valid must down to zero
39      end
40
```

Design code

```
module Dp_Sync_RAM #(ADDR_SIZE=8,MEM_DEPTH=256)(interface_RAM.DUT_Design
inst_interface );

    logic clk, rst_n, rx_valid;
    logic [9:0] din;
    logic tx_valid;
    logic [7:0] dout;

   assign clk      = inst_interface.clk;
   assign rst_n    = inst_interface.rst_n;
   assign rx_valid = inst_interface.rx_valid;
   assign din      = inst_interface.din;

   assign inst_interface.dout     = dout;
   assign inst_interface.tx_valid = tx_valid ;
```

```
    reg [7:0] mem [MEM_DEPTH-1:0];
    reg [ADDR_SIZE-1:0] rd_addr,wr_addr ;

    always @(posedge clk ) begin // first bug : rst must be syncrouns no Ashync
        if (!rst_n) begin
           dout <= 0;
           tx_valid <= 0; //second bug : tx must take zero when rst_n is asserted
           wr_addr <= 0 ;
           rd_addr <= 0 ;//3th bug : when rst_n asserted , addr and not 2 internal
signals it only one
        end
        else if (rx_valid) begin
           tx_valid <= 0;
            case (din[9:8])
            2'b00:  wr_addr  <= din[7:0];
            2'b01: mem[wr_addr] <= din[7:0];
            2'b10: rd_addr <= din[7:0];
            default: {dout, tx_valid} <= {mem[rd_addr], 1'b1};
           endcase
        end
        else begin
            tx_valid  <= 0 ;
        end
         //4th bug : when rx_valid not asserted then tx_valid must down to zero
    end

endmodule
```

Golden model code

```
module RAM_Golden #(ADDR_SIZE=8,MEM_DEPTH=256)(interface_RAM.GOLDEN_REF inst_interface);

logic clk, rst_n, rx_valid;
logic [9:0] din;

logic [7:0] dout_expect;
logic tx_valid_expect;

assign clk      = inst_interface.clk;
assign rst_n    = inst_interface.rst_n;
assign rx_valid = inst_interface.rx_valid;
assign din      = inst_interface.din;

assign inst_interface.dout_expect     = dout_expect;
assign inst_interface.tx_valid_expect = tx_valid_expect ;
```

```
reg [7:0] mem [MEM_DEPTH-1:0];

reg [7:0] address_rd,address_wr;

always @(posedge clk) begin  //async rst is not supported with RAM
    if (~rst_n) begin
        // reset
        dout_expect <= 0;
        tx_valid_expect <= 0;
        address_rd <= 0;
        address_wr <= 0;
    end
    else if (rx_valid == 1'b1)begin
        tx_valid_expect <= 0;
        if (din [9:8]== 2'b00)
            address_wr <= din[7:0];
        else if (din [9:8]==2'b01)
                mem[address_wr] <= din[7:0];
        else if (din [9:8]==2'b10 )
            address_rd <= din[7:0];
        else if (din [9:8]==2'b11) begin
                dout_expect <= mem[address_rd];
                tx_valid_expect <= 1;
            end
            else begin
                tx_valid_expect <= 0;
            end
        end
    else
    tx_valid_expect <= 0;

end
endmodule
```

Top code

```
module top();
    bit clk;
    always #10 clk=~clk;
    interface_RAM inst_interface(clk);
    Dp_Sync_RAM DUT_Design (inst_interface);
    RAM_Golden  GOLDEN_REF (inst_interface);
    tb_RAM      TESTBENCH  (inst_interface);
    RAM_Assertion_sva ASSERTION (inst_interface);
     bind Dp_Sync_RAM  RAM_Assertion_sva sva_inst (inst_interface);
endmodule
```

Interface code

```
interface interface_RAM (clk);

    input bit  clk;
    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8  ;
    logic rst_n     ;
    logic [9:0] din ;
    logic rx_valid  ;
    logic [7:0] dout, dout_expect;
    logic tx_valid ,tx_valid_expect ;

    modport DUT_Design (input clk, rst_n, din, rx_valid,
                        output dout, tx_valid);

    modport ASSERTION  (input clk, rst_n, din, rx_valid,
                          dout, tx_valid,dout_expect, tx_valid_expect);

    modport TESTBENCH  (input clk, dout_expect, tx_valid_expect, dout, tx_valid,
                        output rst_n, din, rx_valid );

    modport GOLDEN_REF(input clk, rst_n, din, rx_valid,
                        output dout_expect, tx_valid_expect );
endinterface
```

Packages code

```
package package_RAM;
    class  RAM ;
    bit clk;
    rand logic rst_n;
    bit [1:0] opcode ;
    bit [7:0] address  , data_wr , addr_wr,addr_rd;
    rand logic [9:0] din;
    rand logic rx_valid;
    logic [7:0] dout;
    logic tx_valid;

    constraint ports {
    rst_n    dist {1:=1000 , 0:=10};
    rx_valid dist {1:=1000 , 0:=10};
    din[9:8] == opcode ;
    din[7:0] == address ;

    }
function void Data_out();
  if (opcode==2'b00)
```

```
            addr_wr=din [7:0];
    else if (opcode==2'b01)
            data_wr=din [7:0];
    else if (opcode==2'b10)
            addr_rd=din [7:0];
endfunction

function void opcode_0(logic[1:0] state);
 opcode =state;
if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b00)
        opcode=2'b01;
    else if (opcode==2'b01)
        opcode=2'b10;
    else if (opcode==2'b10)
        opcode=2'b11;
    else if (opcode==2'b11)
        opcode=2'b00;

endfunction
function void set_rx_valid(logic select);
    rx_valid = select ;
endfunction

function void writing(logic[1:0] state);
 opcode =state;
 if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b00)
        opcode=2'b01;
    else if (opcode==2'b01)
        opcode=2'b00;
    else
        opcode=2'b00;

endfunction
function void Reading(logic[1:0] state);
 opcode =state;
if (rst_n==0)
        opcode=2'b00;
    else if (opcode==2'b11)
        opcode=2'b10;
    else if (opcode==2'b10)
        opcode=2'b11;
```

```
    else
        opcode=2'b10;

endfunction

    covergroup cvg@(posedge clk);
        reset: coverpoint rst_n{
        bins reset_asserted ={0};
        bins reset_disable  ={1};
        }

        data_valid:coverpoint din[9:8]{
        bins writing_complete = (2'b00 => 2'b01);
        bins writing = {2'b00};
        bins repeat_writing =(2'b01 => 2'b00 => 2'b01);
        bins reading_complete = (2'b10 => 2'b11);
        bins reading = {2'b10};
        bins change_wr_rd =(2'b01 => 2'b10);
        bins change_rd_wr =(2'b11 => 2'b00);
        bins repeat_reading =(2'b11 => 2'b10 => 2'b11);
        bins default_values []={2'b00,2'b01,2'b10,2'b11};
        }
        data_written   :coverpoint data_wr;
        data_read      :coverpoint dout   ;
        address_written:coverpoint addr_wr;
        address_read   :coverpoint addr_rd;

        receive_valid: coverpoint rx_valid{
        bins receive_asserted ={1};
        bins receive_disable  ={0};
        }
        send_valid:coverpoint tx_valid{
        bins send_asserted ={1};
        bins send_disable  ={0};
        }

        Writing_address: cross address_written,receive_valid{
         // ignore_bins ignore_bin1 = binsof (receive_valid. receive_disable);
        }
        Writing_data: cross data_written,receive_valid{
         // ignore_bins ignore_bin1 = binsof (receive_valid. receive_disable);
        }
        reading_address: cross address_read,receive_valid{
         // ignore_bins ignore_bin1 = binsof (receive_valid. receive_disable);
        }
```

```
        reading_data: cross data_read,receive_valid{
         ignore_bins ignore_bin1 = binsof (receive_valid. receive_disable);
        }

    endgroup
    cvg=new();
    endclass
endpackage
```

Assertion Code

```
module RAM_Assertion_sva (interface_RAM.ASSERTION inst_interface );
    bit  clk;
    logic rst_n     ;
    logic [9:0] din ;
    logic rx_valid  ;
    logic [7:0] dout, dout_expect;
    logic tx_valid ,tx_valid_expect ;
    logic [1:0] opcode;

    assign clk      = inst_interface.clk;
    assign rst_n    = inst_interface.rst_n;
    assign rx_valid = inst_interface.rx_valid;
    assign din      = inst_interface.din;
    assign dout_expect     = inst_interface.dout_expect;
    assign tx_valid_expect = inst_interface.tx_valid_expect ;
    assign dout     = inst_interface.dout;
    assign tx_valid = inst_interface.tx_valid ;
    assign opcode   = din[9:8];


    property DATA_OUT;
    @(posedge clk) disable iff (~rst_n )
     (opcode == 2'b00)  |=> (opcode == 2'b01) |=> (opcode == 2'b10) |=> (opcode ==
2'b11) |=> (dout == dout_expect) |-> (tx_valid == tx_valid_expect);
    endproperty

     property RESET;
    @(posedge clk)
     (rst_n==0) |=> (dout==0) |->(tx_valid==0);
    endproperty

     property ENABLE ;
    @(posedge clk) disable iff (~rst_n)
     (rx_valid==0) |=> ($past(dout)==dout) ;
    endproperty
```

```
        DATA_OUT_Assertion: assert property (DATA_OUT) else $display("DATA_OUT fail");
        RESET_Assertion   : assert property (RESET)    else $display("RESET fail   ");
        ENABLE_Assertion  : assert property (ENABLE)   else $display("ENABLE fail  ");

        DATA_OUT_Cover: cover property (DATA_OUT) $display("DATA_OUT pass");
        RESET_Cover   : cover property (RESET)    $display("RESET pass   ");
        ENABLE_Cover  : cover property (ENABLE)   $display("ENABLE pass  ");
    endmodule
```

Testbench code

```
import package_RAM::*;
typedef enum logic [1:0] {Write_address=2'b00 ,Write_data=2'b01,
Read_address=2'b10 ,Read_data=2'b11} state_e;
module tb_RAM (interface_RAM.TESTBENCH inst_interface);

    RAM class_RAM =new();
    bit clk ;
    logic rst_n     ;
    logic [9:0] din ;
    logic rx_valid  ;
    logic [7:0] dout,dout_expect;
    logic tx_valid,tx_valid_expect ;
    logic [1:0] opcode;
    integer Correct_count =0;
    integer Error_count   =0;
    state_e state;
    logic [7:0] Random_address[0 : 255];
    logic [7:0] Random_data [0 : 255];
    logic [7:0] Queue [$];
    integer counter_rd = 0 ;
    integer counter_wr = 0 ;

assign clk             = inst_interface.clk;
assign dout            = inst_interface.dout;
assign dout_expect     = inst_interface.dout_expect ;
assign tx_valid        = inst_interface.tx_valid;
assign tx_valid_expect = inst_interface.tx_valid_expect;
assign inst_interface.rst_n    = rst_n;
assign inst_interface.din      = din;
assign inst_interface.rx_valid = rx_valid;



   always @(din)begin
    case (din[9:8])
        2'b00: state=Write_address;
```

```
        2'b01: state=Write_data;
        2'b10: state=Read_address;
        2'b11: state=Read_data;

    endcase
   end

    always@(clk)begin
        class_RAM.clk=clk;
    end

    task inst();

        rst_n    = class_RAM.rst_n;
        din      = class_RAM.din;
        rx_valid = class_RAM.rx_valid;
        opcode   = class_RAM.opcode;

        @(negedge clk)begin
            if (dout===dout_expect)
                Correct_count++;
            else begin
                Error_count++;
                $display("There is an Error in %0t ,dout != dout_expect ,dout =%0h and
dout_expect=%0h ",$time(),dout,dout_expect);
            end
            if (tx_valid==tx_valid_expect)
                Correct_count++;
            else begin
                Error_count++;
                $display("There is an Error in %0t ,tx_valid !=
tx_valid_expect ,tx_valid =%0h and
tx_valid_expect=%0h ,",$time(),tx_valid,tx_valid_expect);
            end
        end
        class_RAM.dout=dout;
        class_RAM.tx_valid=tx_valid;
        class_RAM.Data_out();
    endtask

task Get_information();
    for (int i = 0; i < 256; i++) begin
        Queue.push_front(i);
    end
    Queue.shuffle();
```

```
    for (int i = 0; i < 256; i++) begin
        Random_address[i]=Queue.pop_front();
    end
    for (int i = 255; i >= 0; i--) begin
        Queue.push_front(i);
    end
    Queue.shuffle();
    for (int i = 0; i < 256; i++) begin
        Random_data[i]=Queue.pop_front();
    end
endtask

initial begin
    Get_information();//address and data are ready
    opcode = 0 ;
    for (int i = 0; i < 6; i++) begin
       if (i==0)begin//testing Reset
            assert (class_RAM.randomize());
            class_RAM.rst_n=0;
            inst();
       end

       else if (i==1)begin//write addr -> write data -> read addr -> read data
           for (int j = 0; j < 1500; j++) begin
               class_RAM.opcode_0(opcode);
               case (class_RAM.opcode)
                2'b00 : class_RAM.address = Random_address [counter_wr];
                2'b01 :begin
                 class_RAM.address = Random_data  [counter_wr];
                 counter_wr = ( counter_wr + 1 ) % 256 ;
                       end
                2'b10 : class_RAM.address = Random_address [counter_rd];
                2'b11 :begin
                 class_RAM.address = Random_data [counter_rd];
                 counter_rd = ( counter_rd + 1 ) % 256 ;
                       end
               endcase
               assert (class_RAM.randomize());
               class_RAM.rx_valid=1;
               inst();
           end
       end
       else if (i==2)begin//write addr -> write data -> read addr -> read data
           for (int j = 0; j < 500; j++) begin
               class_RAM.opcode_0(opcode);
```

```
            case (class_RAM.opcode)
             2'b00 : class_RAM.address = Random_address [counter_wr];
             2'b01 :begin
              class_RAM.address = Random_data  [counter_wr];
              counter_wr = ( counter_wr + 1 ) % 255 ;
                    end
             2'b10 : class_RAM.address = Random_address [counter_rd];
             2'b11 :begin
              class_RAM.address = Random_data [counter_rd];
              counter_rd = ( counter_rd + 1 ) % 255 ;
                    end
            endcase
            assert (class_RAM.randomize());
            inst();
        end
    end
     else if (i==3)begin//write addr -> write data -> read addr -> read data
        for (int k = 0; k < 600; k++) begin
            class_RAM.writing(opcode);
            case (class_RAM.opcode)
             2'b00 : class_RAM.address = Random_address [counter_wr];
             2'b01 :begin
              class_RAM.address = Random_data  [counter_wr];
              counter_wr = ( counter_wr + 1 ) % 255 ;
                 end
            endcase
            assert (class_RAM.randomize());
            class_RAM.rx_valid = 1 ;
            inst();
        end
    end
     else if (i==4)begin//write addr -> write data -> read addr -> read data
        for (int L = 0; L < 800; L++) begin
            class_RAM.Reading(opcode);
            case (class_RAM.opcode)
             2'b10 : class_RAM.address = Random_address [counter_rd];
             2'b11 :begin
              class_RAM.address = Random_data [counter_rd];
              counter_rd = ( counter_rd + 1 ) % 255 ;
                    end
            endcase
            assert (class_RAM.randomize());
            class_RAM.rx_valid  = 1 ;
            inst();
        end
```

```
        end
         else if (i==5)begin//write addr -> write data -> read addr -> read data
            for (int M = 0; M < 1000; M++) begin
                class_RAM.opcode_0(opcode);
                case (class_RAM.opcode)
                 2'b00 : class_RAM.address = Random_address [counter_wr];
                 2'b01 :begin
                  class_RAM.address = Random_data  [counter_wr];
                  counter_wr = ( counter_wr + 1 ) % 255 ;
                        end
                 2'b10 : class_RAM.address = Random_address [counter_rd];
                 2'b11 :begin
                  class_RAM.address = Random_data [counter_rd];
                  counter_rd = ( counter_rd + 1 ) % 255 ;
                        end
                endcase
                assert (class_RAM.randomize());
                class_RAM.rx_valid = 0 ;
                inst();
            end
        end


    end

        $display("The total correct count =%0d ,The total Error count =%0d
",Correct_count/2,Error_count );
        $stop;
    end
endmodule
```

Do file

```
vlib work
vlog testbench.sv Dp_Sync_RAM.sv top.sv package.sv Golden_model_RAM.sv interface.sv +cover
vsim -voptargs=+acc work.top -cover
run -all
coverage save top.ucdb -du Dp_Sync_RAM -onexit
coverage report -detail -cvg -comments -output fcover_report.txt {}
quit -sim
vlog testbench.sv Dp_Sync_RAM.sv top.sv package.sv Golden_model_RAM.sv interface.sv +cover
vsim -voptargs=+acc work.top -cover
run -all
coverage save top.ucdb -du Dp_Sync_RAM -onexit
quit -sim
vcover report top.ucdb -details -annotate -all -output Code_coverage_report.txt
```

Code Coverage

```
Coverage Report by instance with details

================================================================================
=== Instance: /\top#DUT_Design /sva_inst
=== Design Unit: work.RAM_Assertion_sva
================================================================================
```

Assertion Coverage:

Assertions	3	3	0	100.00%

```
Name                File(Line)              Failure     Pass
                                            Count       Count
--------------------------------------------------------------------------------
/\top#DUT_Design /sva_inst/DATA_OUT_Assertion
                    Assertion.sv(36)        0           1
/\top#DUT_Design /sva_inst/RESET_Assertion
                    Assertion.sv(37)        0           1
/\top#DUT_Design /sva_inst/ENABLE_Assertion
                    Assertion.sv(38)        0           1

Directive Coverage:
    Directives                  3       3       0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                            Design Design   Lang File(Line)     Hits Status
                                Unit   UnitType
--------------------------------------------------------------------------------------------
/\top#DUT_Design /sva_inst/DATA_OUT_Cover
                                RAM_Assertion_sva Verilog  SVA  Assertion.sv(40) 719 Covered
/\top#DUT_Design /sva_inst/RESET_Cover   RAM_Assertion_sva Verilog  SVA  Assertion.sv(41)  49 Covered
/\top#DUT_Design /sva_inst/ENABLE_Cover  RAM_Assertion_sva Verilog  SVA  Assertion.sv(42) 968 Covered

================================================================================
=== Instance: /\top#DUT_Design
=== Design Unit: work.Dp_Sync_RAM
================================================================================
```

Branch Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Branches	7	7	0	100.00%

```
=================================Branch Details================================

Branch Coverage for instance /\top#DUT_Design

    Line       Item                 Count     Source
    ----       ----                 -----     ------
  File Dp_Sync_RAM.sv
------------------------------------IF Branch------------------------------------
    20                              4401    Count coming in to IF
    20         1                      49                                  if (!rst_n) begin
    26         1                    3367                                  else if (rx_valid) begin
    35         1                     985                                  else begin
Branch totals: 3 hits of 3 branches = 100.00%

----------------------------------CASE Branch-----------------------------------
    28                              3367    Count coming in to CASE
    29         1                     814                                  2'b00:  wr_addr  <= din[7:0];
    30         1                     795                                  2'b01: mem[wr_addr] <= din[7:0];
    31         1                     885                                  2'b10: rd_addr <= din[7:0];
    32         1                     873                                  default: {dout, tx_valid} <= {mem[rd_addr], 1'b1};
Branch totals: 4 hits of 4 branches = 100.00%
```

Statement Coverage:

Enabled Coverage	Bins	Hits	Misses	Coverage
Statements	15	15	0	100.00%

```
=============================Statement Details=================================

Statement Coverage for instance /\top#DUT_Design  --

    Line       Item                 Count     Source
    ----       ----                 -----     ------
  File Dp_Sync_RAM.sv
    1                                       module Dp_Sync_RAM #(ADDR_SIZE=8,MEM_DEPTH=256)(interface_RAM.DUT_Design inst_interface );
    2
    3                                           logic clk, rst_n, rx_valid;
    4                                           logic [9:0] din;
    5                                           logic tx_valid;
    6                                           logic [7:0] dout;
    7
    8          1                 8803        assign clk     = inst_interface.clk;
    9          1                   99        assign rst_n   = inst_interface.rst_n;
    10         1                    5        assign rx_valid = inst_interface.rx_valid;
    11         1                 4395        assign din     = inst_interface.din;
    12
```

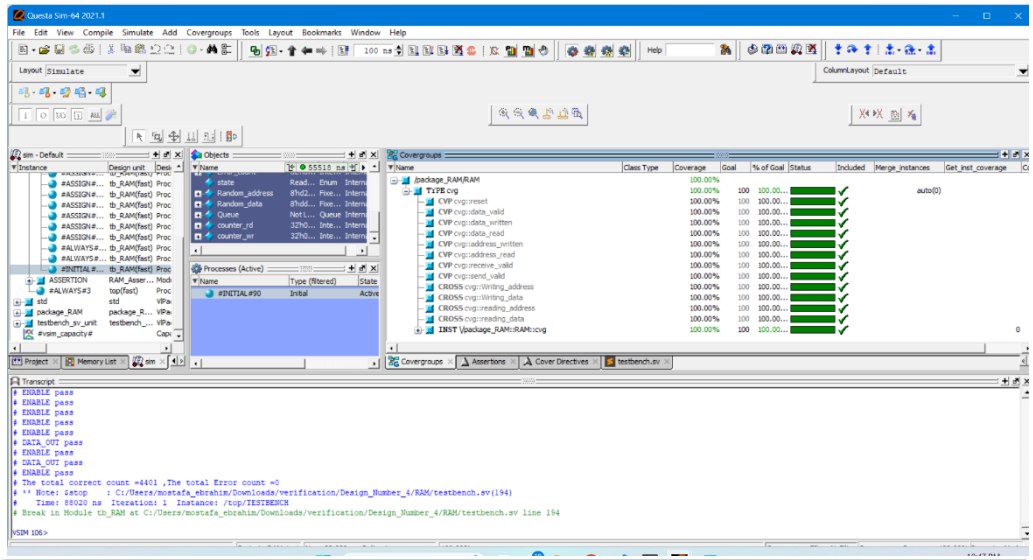
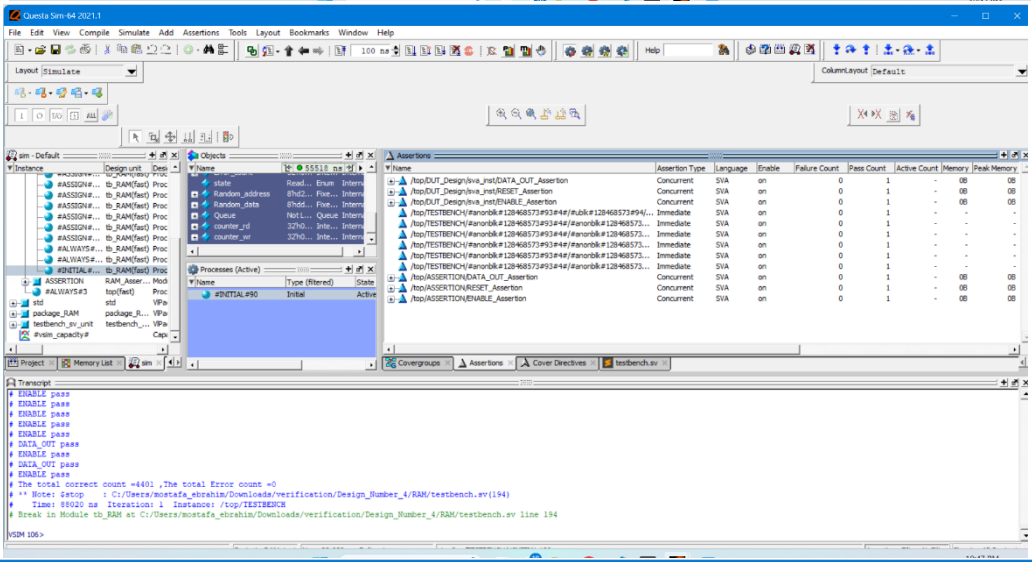
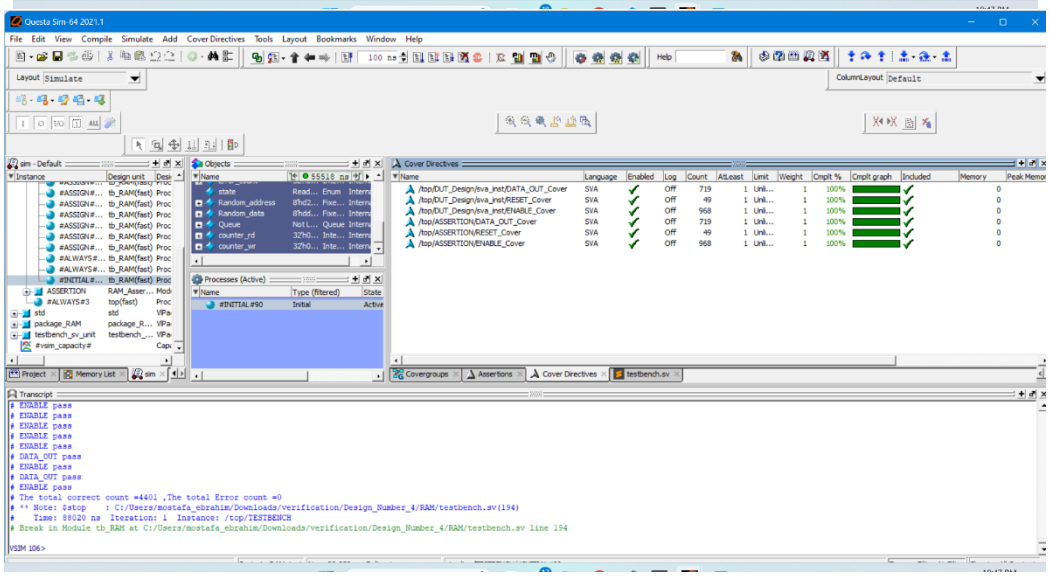
```
  13                              assign inst_interface.dout     = dout;
  14                              assign inst_interface.tx_valid = tx_valid ;
  15
  16                                      reg [7:0] mem [MEM_DEPTH-1:0];
  17                                      reg [ADDR_SIZE-1:0] rd_addr,wr_addr ;
  18
  19        1             4401            always @(posedge clk ) begin // first bug : rst must be syncrouns no Ashync
  20                                              if (!rst_n) begin
  21        1              49                         dout <= 0;
  22        1              49                         tx_valid <= 0; //second bug : tx must take zero when rst_n is asserted
  23        1              49                         wr_addr <= 0 ;
  24        1              49                         rd_addr <= 0 ;//3th bug : when rst_n asserted , addr and not 2 internal signals it only one
  25                                              end
  26                                              else if (rx_valid) begin
  27        1            3367                         tx_valid <= 0;
  28                                                   case (din[9:8])
  29        1             814                                 2'b00:  wr_addr  <= din[7:0];
  30        1             795                                 2'b01: mem[wr_addr] <= din[7:0];
  31        1             885                                 2'b10: rd_addr <= din[7:0];
  32        1             873                                 default: {dout, tx_valid} <= {mem[rd_addr], 1'b1};
  33                                                   endcase
  34                                              end
  35                                              else begin
  36        1             985                                 tx_valid  <= 0 ;
```

Toggle Coverage:

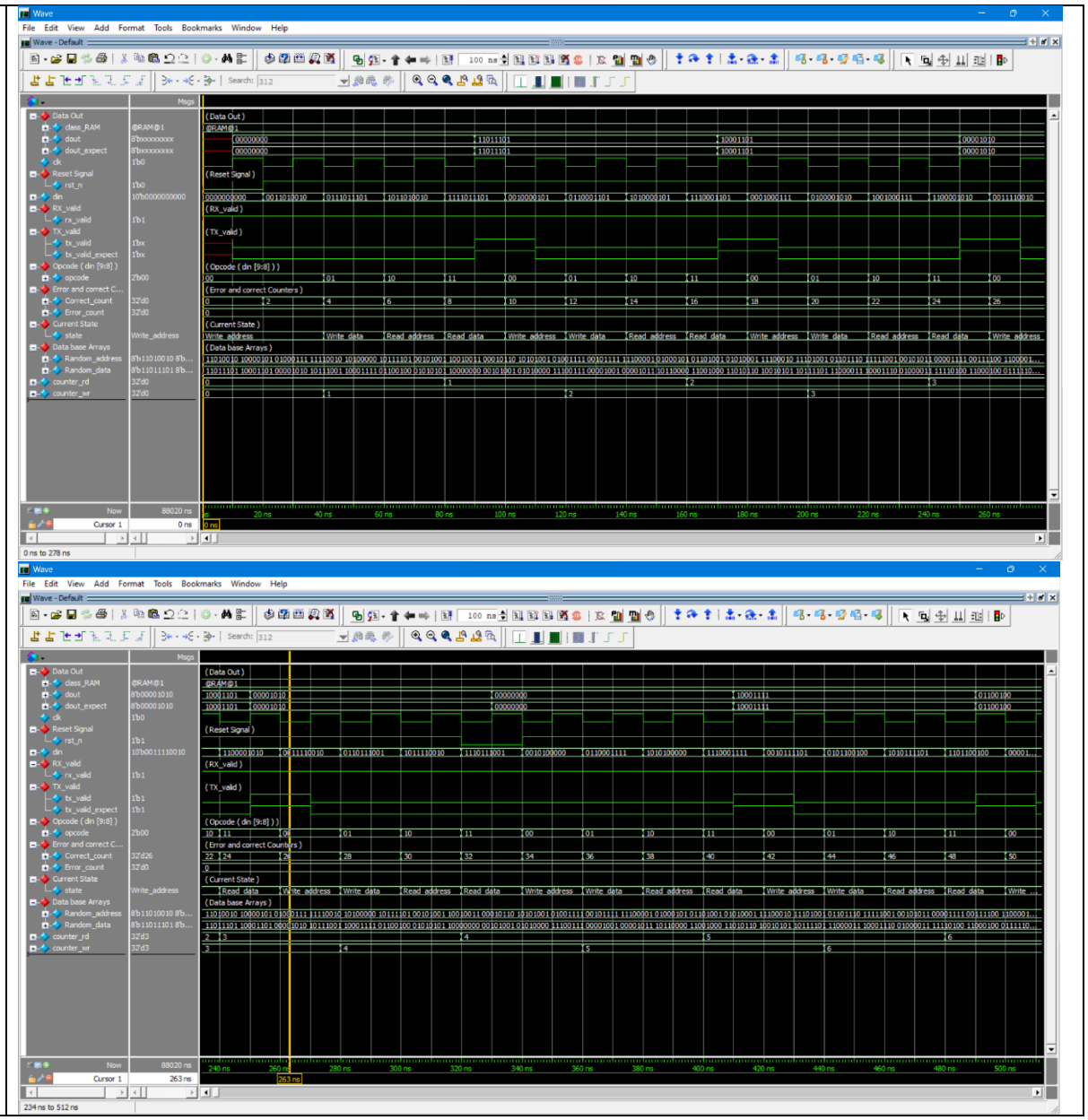
Enabled Coverage	Bins	Hits	Misses	Coverage
Toggles	76	76	0	100.00%

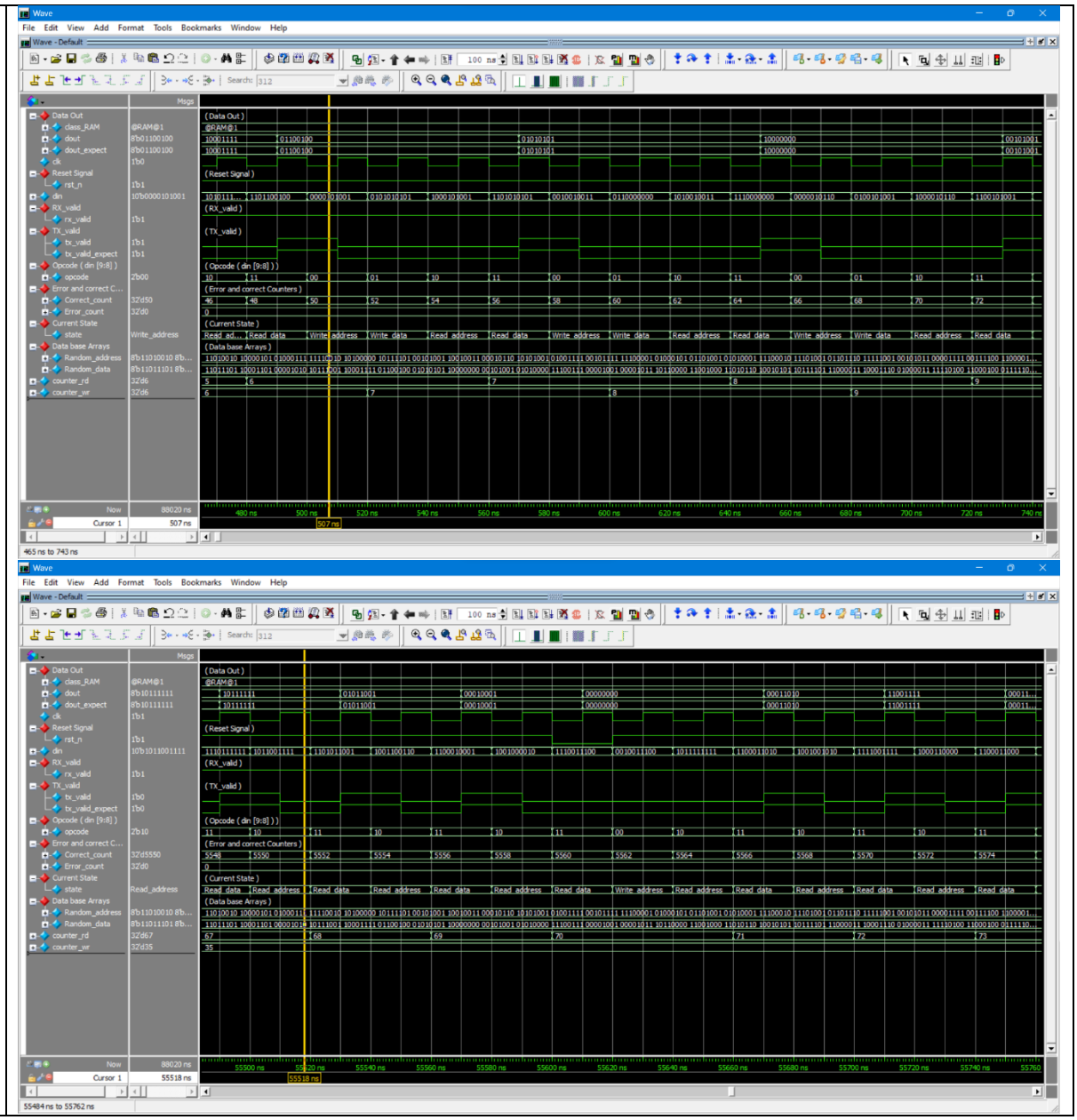
```
================================Toggle Details================================

Toggle Coverage for instance /\top#DUT_Design  --

                                Node     1H->0L     0L->1H                 "Coverage"
                                --------------------------------------------------------------
                                 clk         1          1                    100.00
                               din[9-0]      1          1                      100.00
                               dout[7-0]     1          1                      100.00
                             rd_addr[7-0]    1          1                      100.00
                                 rst_n       1          1                    100.00
                                rx_valid     1          1                      100.00
                                tx_valid     1          1                      100.00
                             wr_addr[7-0]    1          1                      100.00

Total Node Count    =       38
Toggled Node Count  =       38
Untoggled Node Count =        0

Toggle Coverage     =    100.00% (76 of 76 bins)


DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------
Name                          Design Design   Lang File(Line)     Hits Status
                              Unit   UnitType
--------------------------------------------------------------------------------------
/\top#DUT_Design /sva_inst/DATA_OUT_Cover
                         RAM_Assertion_sva Verilog  SVA  Assertion.sv(40) 719 Covered
/\top#DUT_Design /sva_inst/RESET_Cover   RAM_Assertion_sva Verilog  SVA  Assertion.sv(41)  49 Covered
/\top#DUT_Design /sva_inst/ENABLE_Cover  RAM_Assertion_sva Verilog  SVA  Assertion.sv(42) 968 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 3

ASSERTION RESULTS:
---------------------------------------------------------------------
Name             File(Line)              Failure     Pass
                                         Count       Count
---------------------------------------------------------------------
/\top#DUT_Design /sva_inst/DATA_OUT_Assertion
                 Assertion.sv(36)              0        1
/\top#DUT_Design /sva_inst/RESET_Assertion
                 Assertion.sv(37)              0        1
/\top#DUT_Design /sva_inst/ENABLE_Assertion
                 Assertion.sv(38)              0        1
```

Total Coverage By Instance (filtered view): 100.00%

Function coverage	  

Questa Snippets




Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Data Out
class_RAM
dout
dout_expect
clk
Reset Signal
rst_n
din
RX_valid
rx_valid
TX_valid
tx_valid
tx_valid_expect
Opcode (din [9:8])
opcode
Error and correct C...
Correct_count
Error_count
Current State
state
Data base Arrays
Random_address
Random_data
counter_rd
counter_wr
Now
Cursor 1
507 ns
465 ns to 743 ns
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Cursor 1
55518 ns
55484 ns to 55762 ns

Verification req document

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM Reset	Incase rst_n is asserted , it will change the data out and turn the output validity	under constrain with size data in = 10 bits and rx_valid 1bit.	This case included in cover point to hit and count number of apperance of this valid bins : reset_asserted = {0};	Outputs signals were checked by two ways : 1) by using assertion for reset . 2) by comparing in the testbench with expected value from Golden model RAM
Writing state (activate)	Incase of this state must din[9:8] firtly have 2'b00 to load din[7:0] in internal signal named addr_wr and wait the next cycle or next of it to reciver din[9:8]=2'b01 which mean store din[7:0] (data) in the Given address before.	under constrain by creating Queue and have unique 256 element (address) to get in all addresses in the RAM and pop values in array to use it to pass the values.	included in coverpoints and cross coverage , to hit and count number of apperance of this valid bins : bins writing_complete = (2'b00 => 2'b01); bins writing = {2'b00}; bins repeat_writing = (2'b01=>2'b0=>2'b01); bins change_wr_rd = (2'b01 => 2'b10); bins default_values[ ] = { 2'b00 , 2'b01 , 2'b10 , 2'b11}; coverpoint data_wr ; coverpoint addr_wr; coverpoint addr_rd; bins receive_asserted = {1}; bins receive_disable = {0}; cross address_written,receive_valid ; cross data_written,receive_valid;	it can't be checked until reading operation done
Reading State(activate)	Incase of this state , din [9:8] must have 2'b10 and din[7:0] that is considered as the address of location that will load in internal signal also called addr_rd , and wait to get in posedge clk din [9:8] = 2'b11, then loaded dout with the value which is in the perivious cycle (din[9:8]=2'b10)	under some constrians address and data for written operation , and read all addresses in RAM	included in coverpoints and cross coverage to hit and count the appearance of all the values bins bins reading_complete = (2'b10 => 2'b11); bins reading = {2'b10}; bins change_wr_rd = (2'b01 => 2'b10); bins repeat_reading = (2'b11 => 2'b10 => 2'b11); bins default_values []={2'b00,2'b01,2'b10,2'b11}; coverpoint dout ; bins send_asserted = {1}; bins send_disable = {0}; cross address_read,receive_valid. cross data_read,receive_valid.	Output will be ched by comparing outputs to Golden_model outputs + help to check by using Assertions
Writing and Reading opeation but with Random Activiate for both (rx_valid = $random)	in this case rst_n will be randomize and also rx_valid	under consrain of randomized inputs and reciecing addr and data	included in coverpoint to hit and count the appearance of all the values bins : all bins must be incremented	Output will be ched by comparing outputs to Golden_model outputs and some Assertions