SPI project.

Team name : clean code

team members :-

1- Abdelrahman Adwe Ali

2- Youssef Mohamed Elsayed Taha

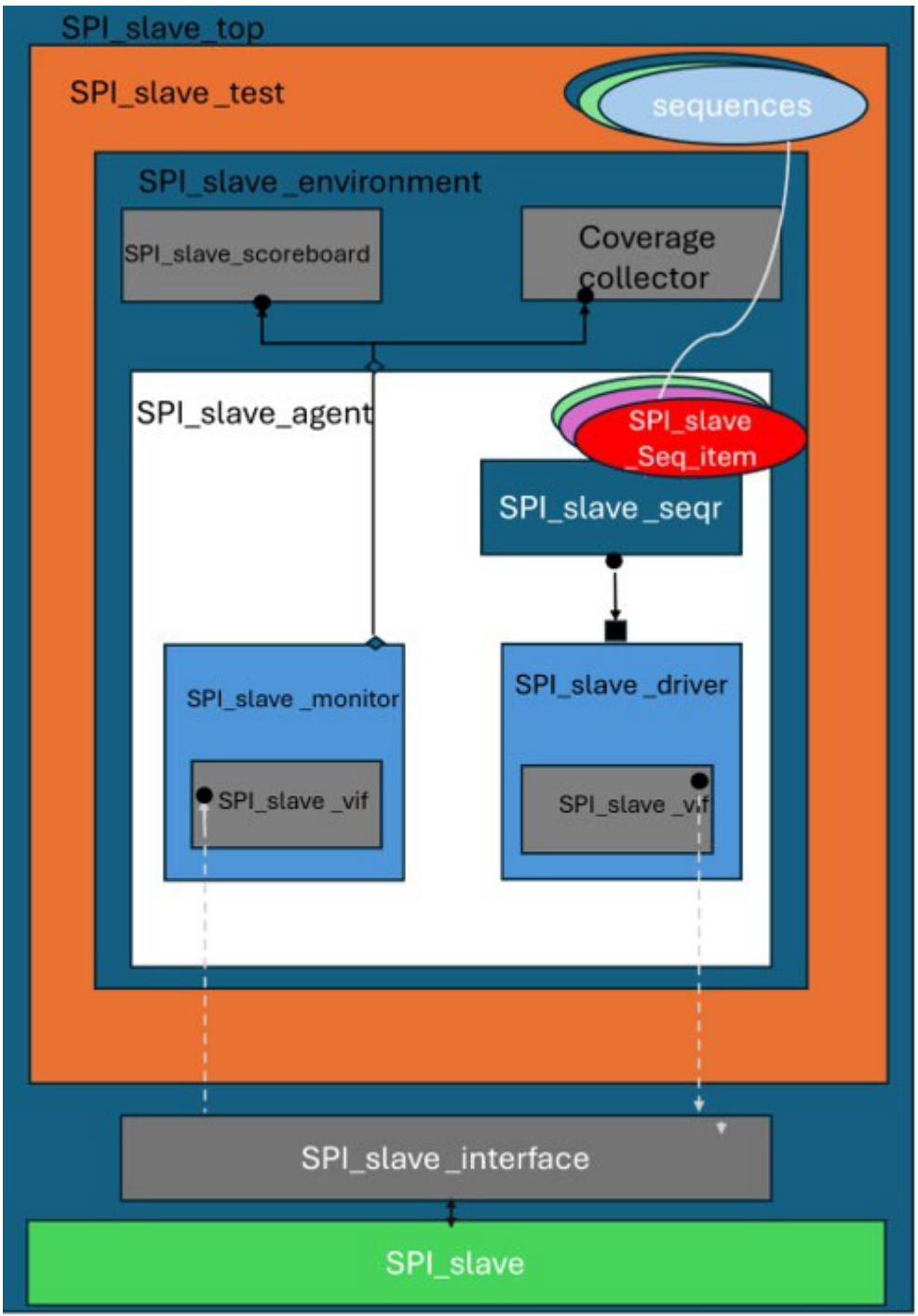
3- Mohamed Elsayed Ebrahim

part1 : spi slave part .

Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI_slave_reset_seq	When the reset is asserted rx_valid, rx_data, MISO should be low	Directed at the start of the simulation	-	A checker in the scoreboard compare the result of the DUT and the Golden Model
SPI_check_1	command decoding only legel command perfixes (000, 001, 110, 111).	Randomization the MOSI with inline constraint(3'b000, 3'b001,3'b110,3'b111)	coverpoint on MOSI [9:8] coross with ss_n transaction	A checker in the scoreboard compare the result of the DUT and the Golden Model
SPI_check_2	rx_valid must asserted exactly after 10 cycle from valid commands	apply valid command sequences with ss_n held 13 or 23 cycle	converpoint on delay between command rx_valid = 10 clk cycle	A checker in the scoreboard compare the result of the DUT and the Golden Model
SPI_check_3	ss_n protocol correct fram length (13 or 23 cycle for read)	randomize ss_n pulse high every 13/23 cycle	coverbins for 13/23 cycle transaction	A checker in the scoreboard compare the result of the DUT and the Golden Model
SPI_check_4	FSM transaction follow legel paths	normal sequence generation via monitor stimulus	cross coverage of current state and next state	A checker in the scoreboard compare the result of the DUT and the Golden Model

uvm structure:



```
import shared_pkg::*;

module SPI_slave (SPI_slave_if.DUT SPI_slaveif);

  reg [3:0] counter;
  reg received_address;
  reg [9:0] out_reg;

  state_t cs, ns;


always @(posedge SPI_slaveif.clk) begin
    if (~SPI_slaveif.rst_n) begin
        cs <= IDLE;
    end
    else begin
        cs <= ns;
    end
end

always @(*) begin
    case (cs)
        IDLE : begin
            if (SPI_slaveif.SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
            if (SPI_slaveif.SS_n)
                ns = IDLE;
            else begin
                if (~SPI_slaveif.MOSI)
                    ns = WRITE;
                else begin
                    if (!received_address)
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
            end
        end
        WRITE : begin
            if (SPI_slaveif.SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
```

```
            if (SPI_slaveif.SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
        READ_DATA : begin
            if (SPI_slaveif.SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
        default : ns = IDLE;
    endcase
end

always @(posedge SPI_slaveif.clk) begin
    if (~SPI_slaveif.rst_n) begin
        SPI_slaveif.rx_data <= 0;
        out_reg <= 0;
        SPI_slaveif.rx_valid <= 0;
        received_address <= 0;
        SPI_slaveif.MISO <= 0;

    end
    else begin
        case (cs)
            IDLE : begin
                SPI_slaveif.rx_valid <= 0;

                out_reg <= 0;
                SPI_slaveif.MISO <= 0;

            end
            CHK_CMD : begin
                counter <= 10;

            end
            WRITE : begin
                if (counter > 0 && !SPI_slaveif.rx_valid) begin
                    out_reg[counter-1] <= SPI_slaveif.MOSI;
                    counter <= counter - 1;

                end
                else begin
                    SPI_slaveif.rx_data <= out_reg;
                    SPI_slaveif.rx_valid <= 1;

                end
```

```
            end
            READ_ADD : begin
                if (counter > 0 && !SPI_slaveif.rx_valid) begin
                    out_reg[counter-1] <= SPI_slaveif.MOSI;
                    counter <= counter - 1;

                end
                else begin
                    SPI_slaveif.rx_data <= out_reg;
                    SPI_slaveif.rx_valid <= 1;
                    received_address <= 1;

                end
            end
            READ_DATA : begin
                if (SPI_slaveif.tx_valid && (SPI_slaveif.rx_valid)) begin

                    if (counter > 0) begin
                        SPI_slaveif.MISO <= SPI_slaveif.tx_data[counter-1];
                        counter <= counter - 1;

                    end
                    else begin
                        received_address <= 0;

                    end
                end
                else begin
                    if (counter > 0 && !SPI_slaveif.rx_valid) begin
                        out_reg[counter-1] <= SPI_slaveif.MOSI;
                        counter <= counter - 1;

                    end
                    else begin
                        SPI_slaveif.rx_data <= out_reg;
                        SPI_slaveif.rx_valid <= 1;
                        counter <= 8;

                    end
                end
            end
            default : ns = IDLE;
        endcase
    end
end
```

```
`ifdef SIM
    property IDLE_to_CHK_CMD;
    @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) ( $fell(SPI_slaveif.SS_n) && cs == IDLE)|-> (ns == CHK_CMD);
    endproperty
    assert property (IDLE_to_CHK_CMD);
    cover  property (IDLE_to_CHK_CMD);

    property CHK_CMD_to_write;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (!SPI_slaveif.SS_n && cs == CHK_CMD && !SPI_slaveif.MOSI) |-> (ns == WRITE);
    endproperty
    assert property (CHK_CMD_to_write);
    cover  property (CHK_CMD_to_write);

    property CHK_CMD_read_address;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (!SPI_slaveif.SS_n && cs == CHK_CMD && SPI_slaveif.MOSI && !received_address) |->
                                                                    (ns == READ_ADD);
    endproperty

    assert property (CHK_CMD_read_address);
    cover  property (CHK_CMD_read_address);

    property CHK_CMD_to_read_data;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (!SPI_slaveif.SS_n && cs == CHK_CMD && SPI_slaveif.MOSI && received_address) |->
                                                                   (ns == READ_DATA);
    endproperty

    assert property (CHK_CMD_to_read_data);
    cover  property (CHK_CMD_to_read_data);

    property WRITE_to_IDLE;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (SPI_slaveif.SS_n && cs == WRITE) |-> (ns == IDLE);
    endproperty
    assert property (WRITE_to_IDLE);
    cover  property (WRITE_to_IDLE);

    property READ_ADD_to_IDLE;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (SPI_slaveif.SS_n && cs == READ_ADD) |-> (ns == IDLE);
    endproperty
    assert property (READ_ADD_to_IDLE);
    cover  property (READ_ADD_to_IDLE);

    property READ_DATA_to_IDLE;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n) (SPI_slaveif.SS_n && cs == READ_DATA) |-> (ns == IDLE);
    endproperty
    assert property (READ_DATA_to_IDLE);
    cover  property (READ_DATA_to_IDLE);
`endif
endmodule
```

spi_slave_if :

```
////////////////////////////////////////////////////////////////////////////////
interface SPI_slave_if (input bit clk);

  logic MOSI, rst_n, SS_n, tx_valid,rx_valid,rx_valid_GM, MISO,MISO_GM;
  logic [7:0] tx_data;
  logic [9:0] rx_data,rx_data_GM;



  modport DUT (
  input MOSI, rst_n, SS_n, tx_valid,clk,tx_data,
  output rx_valid, MISO,rx_data
  );

  // modport GM (
  // input MOSI, rst_n, SS_n, tx_valid,clk,tx_data,
  // output rx_valid_GM, MISO_GM,rx_data_GM
  // );


endinterface : SPI_slave_if
```

spi_slave sva :

```
module SPI_slave_sva (SPI_slave_if.DUT SPI_


    property MISO_reset;
        @(posedge SPI_slaveif.clk) (!SPI_slaveif.rst_n) |=> (~SPI_slaveif.MISO);
    endproperty

    assert property (MISO_reset);
    cover property (MISO_reset);

    property rx_valid_reset;
        @(posedge SPI_slaveif.clk) (!SPI_slaveif.rst_n) |=> (~SPI_slaveif.rx_valid);
    endproperty

    assert property (rx_valid_reset);
    cover property (rx_valid_reset);

    property rx_data_reset;
        @(posedge SPI_slaveif.clk) (!SPI_slaveif.rst_n) |=> (SPI_slaveif.rx_data == 0);
    endproperty
        cover property (MISO_reset);
    assert property (rx_data_reset);
    cover property (rx_data_reset);

    property valid_command_wr_addr;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n)
        ($fell(SPI_slaveif.SS_n) ##1 !SPI_slaveif.MOSI ##1 !SPI_slaveif.MOSI ##1 !SPI_slaveif.MOSI) |-> ##10 (SPI_slaveif.rx_valid) ##[0:$](SPI_slaveif.SS_n);
    endproperty

    assert property (valid_command_wr_addr);
    cover property (valid_command_wr_addr);

    property valid_command_wr_data;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n)
        ($fell(SPI_slaveif.SS_n) ##1 !SPI_slaveif.MOSI ##1 !SPI_slaveif.MOSI ##1 SPI_slaveif.MOSI) |-> ##10 (SPI_slaveif.rx_valid)##[0:$](SPI_slaveif.SS_n);
    endproperty

    assert property (valid_command_wr_data);
    cover property (valid_command_wr_data);

    property valid_command_rd_addr;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n)
        ($fell(SPI_slaveif.SS_n) ##1 SPI_slaveif.MOSI ##1 SPI_slaveif.MOSI ##1 !SPI_slaveif.MOSI) |-> ##10 (SPI_slaveif.rx_valid)##[0:$](SPI_slaveif.SS_n);
    endproperty

    assert property (valid_command_rd_addr);
    cover property (valid_command_rd_addr);
```

```
    property valid_command_rd_data;
        @(posedge SPI_slaveif.clk) disable iff (!SPI_slaveif.rst_n)
        ($fell(SPI_slaveif.SS_n) ##1 SPI_slaveif.MOSI ##1 SPI_slaveif.MOSI ##1 SPI_slaveif.MOSI) |-> ##10 (SPI_slaveif.rx_valid)##[0:$](SPI_slaveif.SS_n);
    endproperty

    assert property (valid_command_rd_data);
    cover property (valid_command_rd_data);




endmodule
```

spi_slave golden model:

```
module SPI_slave_GM (
    input MOSI ,
    input SS_n ,
    input clk , rst_n ,
    input tx_valid ,
    input [7:0] tx_data ,
    output reg MISO ,
    output reg [9:0] rx_data ,
    output reg rx_valid
);
parameter IDLE      = 3'b000 ;
parameter CHK_CMD   = 3'b001 ;
parameter WRITE     = 3'b010 ;
parameter READ_ADD  = 3'b011 ;
parameter READ_DATA = 3'b100 ;

  reg [3:0] counter;
  reg rd_add;
  reg [9:0] out_reg;

  reg [2:0]cs, ns;


always @(posedge clk) begin
    if (~rst_n) begin
        cs <= IDLE;
    end
    else begin
        cs <= ns;
    end
end

always @(*) begin
    case (cs)
        IDLE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = CHK_CMD;
        end
        CHK_CMD : begin
```

```
        CHK_CMD : begin
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else begin
                    if (!rd_add)
                        ns = READ_ADD;
                    else
                        ns = READ_DATA;
                end
            end
        end
        WRITE : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = WRITE;
        end
        READ_ADD : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_ADD;
        end
       READ_DATA : begin
            if (SS_n)
                ns = IDLE;
            else
                ns = READ_DATA;
        end
    endcase
end

always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        rd_add <= 0;
        out_reg <= 0;
        MISO <= 0;

    end
```

```
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;

                out_reg <= 0;
                MISO <= 0;

            end
            CHK_CMD : begin
                counter <= 10;

            end
            WRITE : begin
                if (counter > 0 && !rx_valid) begin
                   out_reg[counter-1] <= MOSI;
                    counter <= counter - 1;

                end
                else begin
                    rx_data <= out_reg;
                    rx_valid <= 1;


                end
            end
            READ_ADD : begin
                if (counter > 0 && !rx_valid) begin
                    out_reg[counter-1] <= MOSI;
                    counter <= counter - 1;

                end
                else begin
                    rx_data <= out_reg;
                    rx_valid <= 1;
                    rd_add <= 1;

                end
            end
           READ_DATA : begin
                if (tx_valid && (rx_valid)) begin

                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;

                    end
```

```
                    else begin
                        rd_add <= 0;

                    end
                end
                else begin
                    if (counter > 0 && !rx_valid) begin
                        out_reg[counter-1] <= MOSI;
                        counter <= counter - 1;

                    end
                    else begin
                        rx_data <= out_reg;
                        rx_valid <= 1;
                        counter <= 8;

                    end
                end
            end
        endcase
    end
end
endmodule
```

spi_slave top:

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: UVM Example
//
////////////////////////////////////////////////////////////////////////////////

import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_slave_test_pkg::*;
module top();
  bit clk;
  initial begin
    forever begin
      #1 clk = ~clk;
    end
  end

  SPI_slave_if SPI_slaveif (clk);
  SPI_slave_GM GM (SPI_slaveif.MOSI,SPI_slaveif.SS_n,SPI_slaveif.clk,SPI_slaveif.rst_n,
                   SPI_slaveif.tx_valid,SPI_slaveif.tx_data,SPI_slaveif.MISO_GM,SPI_slaveif.rx_data_GM,SPI_slaveif.rx_valid_GM);

  SPI_slave DUT (SPI_slaveif);
  bind SPI_slave SPI_slave_sva inst (SPI_slaveif);
  initial begin
    uvm_config_db #(virtual SPI_slave_if)::set(null,"uvm_test_top","SPI",SPI_slaveif);
    run_test("SPI_slave_test");
  end
endmodule
```

spi_slave test:

```
package SPI_slave_test_pkg;
  import SPI_slave_env_pkg::*;
  import SPI_slave_sequencer_pkg::*;
  import SPI_slave_config_pkg::*;
  import SPI_slave_rst_seq_pkg::*;
  import SPI_slave_main_seq_pkg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class SPI_slave_test extends uvm_test;

    `uvm_component_utils(SPI_slave_test);

    SPI_slave_env slave_env;
    SPI_slave_config SPI_slave_cfg;
    SPI_slave_rst_seq reset_sq;
    SPI_slave_write_sequence main_wr_sq;
    SPI_slave_read_sequence main_rd_sq;
    SPI_slave_read_write_sequence main_rd_wr_seq;

    function new(string name = "SPI_slave_test" ,uvm_component parant = null);
      super.new(name,parant);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      slave_env = SPI_slave_env::type_id::create("slave_env",this);
      SPI_slave_cfg = SPI_slave_config::type_id::create("SPI_slave_cfg");
      reset_sq = SPI_slave_rst_seq::type_id::create("reset_sq");
      main_wr_sq = SPI_slave_write_sequence::type_id::create("main_wr_sq");
      main_rd_sq = SPI_slave_read_sequence::type_id::create("main_rd_sq");
      main_rd_wr_seq = SPI_slave_read_write_sequence::type_id::create("main_rd_wr_seq");
      if(!uvm_config_db #(virtual SPI_slave_if)::get(this,"","SPI",SPI_slave_cfg.SPI_slave_vif))
        `uvm_fatal("build_phase", "Test - slave - Unable to get configration object");

      SPI_slave_cfg.is_active = UVM_ACTIVE;
      uvm_config_db #(SPI_slave_config)::set(this,"slave_env.*","CFG",SPI_slave_cfg);
    endfunction

    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      phase.raise_objection(this);

      `uvm_info("run_phase","reset asserted",UVM_LOW)
      reset_sq.start(slave_env.agt.sqr);
      `uvm_info("run_phase","reset deasserted",UVM_LOW)

      `uvm_info("run_phase","stimulus generation write sequence started",UVM_LOW)
      main_wr_sq.start(slave_env.agt.sqr);
```

```
    `uvm_info("run_phase","stimulus generation write sequence ended",UVM_LOW)

    `uvm_info("run_phase","reset asserted",UVM_LOW)
    reset_sq.start(slave_env.agt.sqr);
    `uvm_info("run_phase","reset deasserted",UVM_LOW)

    `uvm_info("run_phase","stimulus generation read squence started",UVM_LOW)
    main_rd_sq.start(slave_env.agt.sqr);
    `uvm_info("run_phase","stimulus generation read sequence ended",UVM_LOW)

    `uvm_info("run_phase","reset asserted",UVM_LOW)
    reset_sq.start(slave_env.agt.sqr);
    `uvm_info("run_phase","reset deasserted",UVM_LOW)

    `uvm_info("run_phase","stimulus generation read write squence started",UVM_LOW)
    main_rd_wr_seq.start(slave_env.agt.sqr);
    `uvm_info("run_phase","stimulus generation read write sequence ended",UVM_LOW)

    phase.drop_objection(this);

  endtask

endclass
endpackage
```

spi_slave env :

```
////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: UVM Example
//
////////////////////////////////////////////////////////////////////////////////
package SPI_slave_env_pkg;
  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import SPI_slave_agent_pkg::*;
  import SPI_slave_scoreboard_pkg::*;
  import SPI_slave_cover_pkg::*;

  class SPI_slave_env extends uvm_env;

    `uvm_component_utils(SPI_slave_env);

    SPI_slave_agent agt;
    SPI_slave_cover cov;
    SPI_slave_scoreboard sb;

    function new (string name = "SPI_slave_env",uvm_component parant = null);
      super.new(name,parant);
    endfunction

    function void build_phase (uvm_phase phase);
      super.build_phase(phase);
      agt = SPI_slave_agent::type_id::create("agt",this);
      cov = SPI_slave_cover::type_id::create("cov",this);
      sb = SPI_slave_scoreboard::type_id::create("sb",this);
    endfunction

    function void connect_phase (uvm_phase phase);
      super.connect_phase(phase);
      agt.agt_ap.connect(sb.sb_export);
      agt.agt_ap.connect(cov.cov_export);
    endfunction

  endclass
endpackage
```

spi_slave agent :

```
package SPI_slave_agent_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_slave_driver_pkg::*;
    import SPI_slave_sequencer_pkg::*;
    import SPI_slave_monitor_pkg::*;
    import SPI_slave_config_pkg::*;
    import SPI_slave_item_pkg::*;

    class SPI_slave_agent extends uvm_agent;
        `uvm_component_utils(SPI_slave_agent);
        SPI_slave_sequencer sqr;
        SPI_slave_driver driver;
        SPI_slave_monitor monitor;
        SPI_slave_config cfg;
        uvm_analysis_port #(SPI_slave_item) agt_ap;

      function new (string name = "SPI_slave_agent",uvm_component parant = null);
        super.new(name,parant);
      endfunction

      function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(SPI_slave_config)::get(this,"","CFG",cfg))
                `uvm_fatal("build_phase", "Unable to get configration object");

            if(cfg.is_active == UVM_ACTIVE)begin
                driver = SPI_slave_driver::type_id::create("driver",this);
                sqr = SPI_slave_sequencer::type_id::create("sqr",this);
            end
            monitor = SPI_slave_monitor::type_id::create("monitor",this);
            agt_ap = new("agt_ap",this);

        endfunction

        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);

            if(cfg.is_active == UVM_ACTIVE)begin
                driver.SPI_slave_vif = cfg.SPI_slave_vif;
                driver.seq_item_port.connect(sqr.seq_item_export);
            end

            monitor.SPI_slave_vif = cfg.SPI_slave_vif;
            monitor.mon_ap.connect(agt_ap);

        endfunction
    endclass
endpackage
```

spi_slave driver :

```
package SPI_slave_driver_pkg;
 `include "uvm_macros.svh"
 import SPI_slave_config_pkg::*;
 import SPI_slave_item_pkg::*;
 import uvm_pkg::*;
class SPI_slave_driver extends uvm_driver #(SPI_slave_item);
    `uvm_component_utils(SPI_slave_driver);

    virtual SPI_slave_if SPI_slave_vif;
    SPI_slave_item seq_item;

    function new (string name = "SPI_slave_driver",uvm_component parant = null );
        super.new(name,parant);
    endfunction

    task run_phase (uvm_phase phase);
        super.run_phase(phase);
        forever begin
            seq_item = SPI_slave_item::type_id::create("seq_item");
            seq_item_port.get_next_item(seq_item);
            SPI_slave_vif.rst_n = seq_item.rst_n;
            SPI_slave_vif.MOSI = seq_item.MOSI;
            SPI_slave_vif.tx_data = seq_item.tx_data;
            SPI_slave_vif.tx_valid = seq_item.tx_valid;
            SPI_slave_vif.SS_n = seq_item.SS_n;
            @(negedge SPI_slave_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase" , seq_item.convert2string_stimulus(),UVM_HIGH)
        end

    endtask
endclass
endpackage
```

```
class SPI_slave_item extends uvm_sequence_item;
```

spi_slave squencer :

```
package SPI_slave_sequencer_pkg;
  import uvm_pkg::*;
  import SPI_slave_item_pkg::*;
  `include "uvm_macros.svh"

  class SPI_slave_sequencer extends uvm_sequencer #(SPI_slave_item);
    `uvm_component_utils(SPI_slave_sequencer);

    function new (string name = "SPI_slave_sequencer",uvm_component parant = null);
      super.new(name,parant);
    endfunction

  endclass
endpackage
```

spi_slave item :

```
package SPI_slave_item_pkg;
  import uvm_pkg::*;
  import shared_pkg::*;

  `include "uvm_macros.svh"
  class SPI_slave_item extends uvm_sequence_item;
    `uvm_object_utils(SPI_slave_item)


    rand logic MOSI, rst_n, SS_n, tx_valid;
    rand logic [7:0] tx_data;

    //output signals
    logic [9:0] rx_data,rx_data_GM;
    logic rx_valid,rx_valid_GM, MISO,MISO_GM;

    //control signals

    rand operation_t operation;
    operation_t pre_operation = read_data;
    rand int cycle_count = 0;
    rand bit[2:0]spi_cmd;

    rand bit MOSI_data[];


    constraint reset_c {
      rst_n dist {0:=1,1:=99};
    }
    constraint MOSI_c {
      MOSI_data.size() == cycle_count-1;
    }
    constraint tx_valid_c {
      if (operation == read_data) {
        tx_valid == 1;
      }
      else{
        tx_valid == 0;
      }

    }
    constraint spi_cmd_cons{
      spi_cmd inside {3'b000, 3'b001, 3'b110, 3'b111};

      if(operation == write_addr){
        spi_cmd == 3'b000;
      }
      else if (operation == write_data){
        spi_cmd == 3'b001;
```

```
    }
    else if(operation == read_data){
      spi_cmd == 3'b111;
    }

  }
  constraint cycle_count_cons{
    if(operation == read_data){
      cycle_count == 23;
    }
    else{
      cycle_count == 13;
    }
  }

  function void post_randomize();
    pre_operation = operation;
    MOSI_data = new[cycle_count-1];
    for(int i = 0; i < 3; i++)begin
      MOSI_data[i] = spi_cmd[2-i];
    end
    for(int i = 3; i < cycle_count-1; i++)begin
      MOSI_data[i] = $urandom_range(0,1);
    end

  endfunction



  function new (string name = "SPI_slave_item");
    super.new(name);
  endfunction

  function string convert2string();
    return $sformatf("%s MOSI = %d,MISO = %d,SS_n = %d,rst_n = %d,rx_data = %d,rx_valid = %d,tx_data = %d,tx_valid = %d",super.convert2string(),
                     MOSI,MISO,SS_n,rst_n,rx_data,rx_valid,tx_data,tx_valid);
  endfunction

  function string convert2string_stimulus();
    return $sformatf("MOSI = %d,MISO = %d,SS_n = %d,rst_n = %d,rx_data = %d,rx_valid = %d,tx_data = %d,tx_valid = %d",
                     MOSI,MISO,SS_n,rst_n,rx_data,rx_valid,tx_data,tx_valid);
  endfunction

endclass
endpackage
```

spi_slave rst seq :

```
package SPI_slave_rst_seq_pkg;
    import uvm_pkg::*;
    import SPI_slave_item_pkg::*;
    `include "uvm_macros.svh"
        class SPI_slave_rst_seq extends uvm_sequence #(SPI_slave_item);
        `uvm_object_utils(SPI_slave_rst_seq)
        SPI_slave_item seq_item;

        function new (string name = "SPI_slave_rst_seq");
            super.new(name);
        endfunction

        task body;
            repeat(5)begin
                seq_item = SPI_slave_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.rst_n = 0;
                seq_item.MOSI = 0;
                seq_item.tx_data = 0;
                seq_item.tx_valid = 0;
                seq_item.SS_n = 1;
                finish_item(seq_item);
            end

            repeat(1)begin
                seq_item = SPI_slave_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.rst_n = 1;
                seq_item.MOSI = 0;
                seq_item.tx_data = 0;
                seq_item.tx_valid = 0;
                seq_item.SS_n = 0;
                finish_item(seq_item);
            end
            repeat(1)begin
                seq_item = SPI_slave_item::type_id::create("seq_item");
                start_item(seq_item);
                seq_item.rst_n = 1;
                seq_item.MOSI = 0;
                seq_item.tx_data = 0;
                seq_item.tx_valid = 0;
                seq_item.SS_n = 1;
                finish_item(seq_item);
            end

        endtask

    endclass
endpackage
```

spi_slave main seq :

```
package SPI_slave_main_seq_pkg;
    import SPI_slave_item_pkg::*;
    import shared_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class SPI_slave_write_sequence extends uvm_sequence #(SPI_slave_item);
        `uvm_object_utils(SPI_slave_write_sequence)

        SPI_slave_item item;

        function new(string name = "SPI_slave_write_sequence");
            super.new(name);
        endfunction

        task body();

            item = SPI_slave_item::type_id::create("item");
            repeat(2000)begin

                assert(item.randomize() with{
                    operation inside{write_addr, write_data};

                });
                send_transaction(item);

            end

        endtask

        task send_transaction(SPI_slave_item trans_item);
            SPI_slave_item item;
            int transaction_cycle = trans_item.cycle_count;
            item = SPI_slave_item::type_id::create("item");
            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n == 1;
                operation == trans_item.operation;
                tx_data   == trans_item.tx_data;

            });
            finish_item(item);
```

```
        for(int j = 0; j < transaction_cycle-1; j++)begin

            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n == 1;
                operation == trans_item.operation;
                tx_data   == trans_item.tx_data;

                MOSI == trans_item.MOSI_data[j];
            });
            finish_item(item);
        end

        start_item(item);
        assert(item.randomize() with{
            SS_n  == 1;
            rst_n == 1;
            operation == trans_item.operation;
            tx_data   == trans_item.tx_data;

        });
        finish_item(item);

    endtask
endclass
```

```
class SPI_slave_read_sequence extends uvm_sequence #(SPI_slave_item);
    `uvm_object_utils(SPI_slave_read_sequence)

    SPI_slave_item item;

    function new(string name = "SPI_slave_read_sequence");
        super.new(name);
    endfunction

    task body();

        item = SPI_slave_item::type_id::create("item");


        repeat(1000)begin


            assert(item.randomize() with{
                operation inside{read_addr, read_data};

                if(pre_operation == read_addr){
                    operation == read_data;
                }
                else if(pre_operation == read_data){
                    operation == read_addr;
                }
                else{
                    operation == read_addr;
                }


            });


            send_transaction(item);

        end

    endtask
```

```
    task send_transaction(SPI_slave_item trans_item);
        SPI_slave_item item;
        int transaction_cycle = trans_item.cycle_count;
        item = SPI_slave_item::type_id::create("item");
        start_item(item);
        assert(item.randomize() with{
            SS_n == 0;
            rst_n ==1;
            operation == trans_item.operation;
            tx_data   == trans_item.tx_data;

        });
        finish_item(item);

        for(int j = 0; j< transaction_cycle-1; j++)begin

            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n == 1;
                operation == trans_item.operation;
                tx_data   == trans_item.tx_data;

                MOSI == trans_item.MOSI_data[j];
            });
            finish_item(item);
        end

        start_item(item);
        assert(item.randomize() with{
            SS_n == 1;
            rst_n ==1;
            operation == trans_item.operation;
            tx_data   == trans_item.tx_data;

        });
        finish_item(item);

    endtask
endclass
```

```
class SPI_slave_read_write_sequence extends uvm_sequence #(SPI_slave_item);
    `uvm_object_utils(SPI_slave_read_write_sequence)

    SPI_slave_item item;

    function new(string name = "SPI_slave_read_write_sequence");
        super.new(name);
    endfunction

    task body();

        item = SPI_slave_item::type_id::create("item");
        repeat(2000)begin


            assert(item.randomize() with{
                if(pre_operation == write_addr){
                    operation inside {write_addr, write_data};
                }
                else if (pre_operation == write_data){
                    operation dist {write_addr :/40, read_addr :/60, read_data :

                }
                else if(pre_operation == read_addr){
                    operation == read_data;
                }
                else if(pre_operation == read_data){
                   operation dist {write_addr :/60, read_addr :/40, read_data :/

                }
                else{
                    operation == write_addr;
                }

            });

            send_transaction(item);

        end

    endtask
```

```
        task send_transaction(SPI_slave_item trans_item);
            SPI_slave_item item;
            int transaction_cycle = trans_item.cycle_count;
            item = SPI_slave_item::type_id::create("item");
            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n ==1;
                operation == trans_item.operation;
                tx_data   == trans_item.tx_data;

            });
            finish_item(item);

            for(int j = 0; j< transaction_cycle-1; j++)begin

                start_item(item);
                assert(item.randomize() with{
                    SS_n == 0;
                    rst_n ==1;
                    operation == trans_item.operation;
                    tx_data   == trans_item.tx_data;

                    MOSI == trans_item.MOSI_data[j];
                });
                finish_item(item);
            end

            start_item(item);
            assert(item.randomize() with{
                SS_n == 1;
                rst_n ==1;
                operation == trans_item.operation;
                tx_data   == trans_item.tx_data;

            });
            finish_item(item);

        endtask
    endclass

endpackage
```

spi_slave monitor :

```
package SPI_slave_monitor_pkg;
    `include "uvm_macros.svh"
    import uvm_pkg::*;
    import SPI_slave_item_pkg::*;

    class SPI_slave_monitor extends uvm_monitor;
        `uvm_component_utils(SPI_slave_monitor);

        virtual SPI_slave_if SPI_slave_vif;
        SPI_slave_item rsp_seq_item;
        uvm_analysis_port #(SPI_slave_item) mon_ap;

        function new (string name = "SPI_slave_monitor",uvm_component parant = null
            super.new(name,parant);
        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 = SPI_slave_item::type_id::create("rsp_seq_item");
                @(negedge SPI_slave_vif.clk);

                rsp_seq_item.rst_n = SPI_slave_vif.rst_n;
                rsp_seq_item.MOSI = SPI_slave_vif.MOSI;
                rsp_seq_item.MISO = SPI_slave_vif.MISO;
                rsp_seq_item.SS_n = SPI_slave_vif.SS_n;
                rsp_seq_item.tx_data = SPI_slave_vif.tx_data;
                rsp_seq_item.tx_valid = SPI_slave_vif.tx_valid;
                rsp_seq_item.rx_data = SPI_slave_vif.rx_data;
                rsp_seq_item.rx_valid = SPI_slave_vif.rx_valid;
                rsp_seq_item.rx_data_GM = SPI_slave_vif.rx_data_GM;
                rsp_seq_item.rx_valid_GM = SPI_slave_vif.rx_valid_GM;
                rsp_seq_item.MISO_GM = SPI_slave_vif.MISO_GM;
                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_DEBUG);
            end

        endtask
    endclass
endpackage
```

spi_slave config :

```
package SPI_slave_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class SPI_slave_config extends uvm_object;
        `uvm_object_utils(SPI_slave_config);

        virtual SPI_slave_if SPI_slave_vif;
        uvm_active_passive_enum is_active;

        function new (string name = "SPI_slave_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

spi_slave shared pkg:

```
package shared_pkg;

  typedef enum {IDLE, CHK_CMD, WRITE,  READ_ADD ,READ_DATA}state_t;

  typedef enum {write_addr, write_data, read_addr, read_data}operation_t;
endpackage
```

spi_slave scoreboard:

```
package SPI_slave_scoreboard_pkg;
    import uvm_pkg::*;
    import SPI_slave_item_pkg::*;

    `include "uvm_macros.svh"

    class SPI_slave_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(SPI_slave_scoreboard)

        uvm_analysis_export #(SPI_slave_item) sb_export;
        uvm_tlm_analysis_fifo #(SPI_slave_item) sb_fifo;
        SPI_slave_item sb_item;


        int error = 0,correct = 0;

      function new (string name = "SPI_slave_scoreboard",uvm_component parant = null);
        super.new(name,parant);
      endfunction

      function void build_phase (uvm_phase phase);
            super.build_phase(phase);

            sb_export = new("sb_export",this);
            sb_fifo = new("sb_fifo",this);
            sb_item = SPI_slave_item::type_id::create("sb_item");

        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(sb_item);
                //ref_model(sb_item);
                if (sb_item.MISO != sb_item.MISO_GM || sb_item.rx_data != sb_item.rx_data_GM || sb_item.rx_valid != sb_item.rx_valid_GM ) begin
                    `uvm_error("run_phase" , $sformatf("comprsion failed, transaction recevied by the DUT %s while the ref_rx_data = %0d,
                                ref_rx_valid = %0b, ref_MISO = %d",sb_item.convert2string(),sb_item.rx_data_GM,sb_item.MISO_GM, sb_item.rx_valid_GM ))
                    error++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct out : %s ",sb_item.convert2string()),UVM_HIGH);
                    correct++;
                end
            end
```

```
        endtask

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total successful transactions: %0d",correct), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total failed transactions: %0d",error), UVM_MEDIUM);
        endfunction

    endclass
endpackage
```

spi_slave coverage:

```
package SPI_slave_cover_pkg;
import uvm_pkg::*;
import shared_pkg::*;
import SPI_slave_item_pkg::*;
`include "uvm_macros.svh"

class SPI_slave_cover extends uvm_component;
    `uvm_component_utils(SPI_slave_cover)

    uvm_analysis_export #(SPI_slave_item) cov_export;
    uvm_tlm_analysis_fifo #(SPI_slave_item) cov_fifo;
    SPI_slave_item cov_item;

    covergroup SPI_cvg;

        rx_cov: coverpoint cov_item.rx_data[9:8] iff (cov_item.SS_n){
            bins rx_wr_addr = {2'b00};
            bins rx_wr_data = {2'b01};
            bins rx_rd_addr = {2'b10};
            bins rx_read_data = {2'b11};
            bins w_address_w_data_r_address_r_data = (2'b00 => 2'b01 => 2'b10 => 2'b11);
        }

        SS_cov: coverpoint cov_item.SS_n {
                bins full_tr = (1 => 0[*13] => 1);
                bins extended_tr = (1 => 0[*23] => 1);
        }

        MOSI_cov: coverpoint cov_item.MOSI {
                bins write_addr = (0 => 0 => 0);
                bins write_data = (0 => 0 => 1);
                bins read_addr = (1 => 1 => 0);
                bins read_data = (1 => 1 => 1);
        }

        cross SS_cov,rx_cov {
            option.cross_auto_bin_max   = 0;
            bins write_addr_full_tr     = binsof(SS_cov.full_tr)&& binsof(rx_cov.rx_wr_addr);
            bins write_data_full_tr     = binsof(SS_cov.full_tr)&& binsof(rx_cov.rx_wr_data);
            bins read_addr_full_tr      = binsof(SS_cov.full_tr)&& binsof(rx_cov.rx_rd_addr);
            bins write_data_extended_tr = binsof(SS_cov.extended_tr)&& binsof(rx_cov.rx_read_data);

            ignore_bins extended_with_wr_addr = binsof(SS_cov.extended_tr)&& binsof(rx_cov.rx_wr_addr);
            ignore_bins extended_with_wr_data = binsof(SS_cov.extended_tr)&& binsof(rx_cov.rx_wr_data);
            ignore_bins extended_with_rd_addr = binsof(SS_cov.extended_tr)&& binsof(rx_cov.rx_rd_addr);
            ignore_bins full_with_rd_data = binsof(SS_cov.full_tr)&& binsof(rx_cov.rx_read_data);
        }

    endgroup
```

```
    function new (string name = "SPI_slave_cover",uvm_component parant = null);
      super.new(name,parant);
       SPI_cvg = 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);
        // cov_item = SPI_slave_item::type_id::create("cov_item");

    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(cov_item);
            SPI_cvg.sample();
        end

    endtask

endclass
endpackage
```

spi_slave do file :

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave /top/SPI_slaveif/*
add wave /top/DUT/inst/assert__valid_command_wr_addr /top/DUT/inst/assert__valid_command_wr_data \
/top/DUT/inst/assert__valid_command_rd_addr /top/DUT/inst/assert__valid_command_rd_data
add wave -position insertpoint \
/top/DUT/cs
add wave -position insertpoint \
/top/DUT/received_address
coverage save SPI_slave.ucdb -onexit
run -all
coverage exclude -src SPI_slave.sv -line 61 -code b
coverage exclude -src SPI_slave.sv -line 139 -code b
coverage exclude -src SPI_slave.sv -line 61 -code s
coverage exclude -src SPI_slave.sv -line 139 -code s
# vcover report SPI_slave.ucdb -details -annotate -all -output coverage_rpt.txt
```

code coverage :

```
SPI_slave.sv(149)                    0          1
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        38        38         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      43        43         0   100.00%
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         40        40         0   100.00%
```

```
================================================================================
=== Instance: /top/SPI_slaveif
=== Design Unit: work.SPI_slave_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         74        74         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /top/SPI_slaveif --

                                                  Node      1H->0L      0L->1H                          "Coverage"
                                                  ----------------------------------------------------------------
                                                  MISO           1           1                             100.00
                                               MISO_GM           1           1                             100.00
                                                  MOSI           1           1                             100.00
                                                  SS_n           1           1                             100.00
                                                   clk           1           1                             100.00
                                                 rst_n           1           1                             100.00
                                            rx_data[9-0]           1           1                             100.00
                                           rx_data_GM[9-0]           1           1                             100.00
                                                rx_valid           1           1                             100.00
                                             rx_valid_GM           1           1                             100.00
                                            tx_data[7-0]           1           1                             100.00
                                                tx_valid           1           1                             100.00

Total Node Count     =         37
Toggled Node Count   =         37
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (74 of 74 bins)
```

functional coverage :

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merg
/SPI_slave_cover_pkg/SPI_slave_cover		100.00%					
TYPE SPI_cvg		100.00%	100	100.00...			
CVP SPI_cvg::rx_cov		100.00%	100	100.00...			
CVP SPI_cvg::SS_cov		100.00%	100	100.00...			
CVP SPI_cvg::MOSI_cov		100.00%	100	100.00...			
CROSS SPI_cvg::{#cross__0#}		100.00%	100	100.00...			
INST VSPI_slave_cover_pkg::SPI_slave_cover::SPI_cvg		100.00%	100	100.00...			
CVP rx_cov		100.00%	100	100.00...			
bin rx_wr_addr		3033	1	100.00...			
bin rx_wr_data		2002	1	100.00...			
bin rx_rd_addr		1490	1	100.00...			
bin rx_read_data		1490	1	100.00...			
bin w_address_w_data_r_address_r_data		584	1	100.00...			
CVP SS_cov		100.00%	100	100.00...			
bin full_tr		6510	1	100.00...			
bin extended_tr		1490	1	100.00...			
CVP MOSI_cov		100.00%	100	100.00...			
bin write_addr		20479	1	100.00...			
bin write_data		16145	1	100.00...			
bin read_addr		15327	1	100.00...			
bin read_data		16237	1	100.00...			
CROSS #cross__0#		100.00%	100	100.00...			
bin write_addr_full_tr		3018	1	100.00...			
bin write_data_full_tr		2002	1	100.00...			
bin read_addr_full_tr		1490	1	100.00...			
bin write_data_extended_tr		1490	1	100.00...			
ignore_bin full_with_rd_data		0	-	-			
ignore_bin extended_with_rd_addr		0	-	-			
ignore_bin extended_with_wr_data		0	-	-			
ignore_bin extended_with_wr_addr		0	-	-			

assertion coverage:

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Mer
/top/DUT/cover__READ_DATA_to_I...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__READ_ADD_to_ID...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__WRITE_to_IDLE	SVA	✓	Off	5020	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__CHK_CMD_to_read...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__CHK_CMD_read_a...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__CHK_CMD_to_writ...	SVA	✓	Off	5020	1	Unli...	1	100%		✓	0	0	
/top/DUT/cover__IDLE_to_CHK_CM...	SVA	✓	Off	8000	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__valid_command...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__valid_command...	SVA	✓	Off	1490	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__valid_command...	SVA	✓	Off	2002	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__valid_command...	SVA	✓	Off	3018	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__rx_data_reset	SVA	✓	Off	15	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__rx_valid_reset	SVA	✓	Off	15	1	Unli...	1	100%		✓	0	0	
/top/DUT/inst/cover__MISO_reset	SVA	✓	Off	15	1	Unli...	1	100%		✓	0	0	

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Co
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	
/SPI_slave_main_seq_pkg::SPI_slave_write_sequence::body/#ublk#223898391#18/immed__20	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_write_sequence::send_transaction/immed__35	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_write_sequence::send_transaction/immed__59	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_write_sequence::send_transaction/#anonblk#223898391#44#4...	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_sequence::body/#ublk#223898391#85/immed__88	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_sequence::send_transaction/immed__116	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_sequence::send_transaction/immed__140	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_sequence::send_transaction/#anonblk#223898391#125#4...	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_write_sequence::body/#ublk#223898391#165/immed__16...	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_write_sequence::send_transaction/immed__200	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_write_sequence::send_transaction/immed__224	Immediate	SVA	on	0	1	
/SPI_slave_main_seq_pkg::SPI_slave_read_write_sequence::send_transaction/#anonblk#223898391#...	Immediate	SVA	on	0	1	
/top/DUT/assert__IDLE_to_CHK_CMD	Concurrent	SVA	on	0	1	
/top/DUT/assert__CHK_CMD_to_write	Concurrent	SVA	on	0	1	
/top/DUT/assert__CHK_CMD_read_address	Concurrent	SVA	on	0	1	
/top/DUT/assert__CHK_CMD_to_read_data	Concurrent	SVA	on	0	1	
/top/DUT/assert__WRITE_to_IDLE	Concurrent	SVA	on	0	1	
/top/DUT/assert__READ_ADD_to_IDLE	Concurrent	SVA	on	0	1	
/top/DUT/assert__READ_DATA_to_IDLE	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__MISO_reset	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__rx_valid_reset	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__rx_data_reset	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__valid_command_wr_addr	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__valid_command_wr_data	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__valid_command_rd_addr	Concurrent	SVA	on	0	1	
/top/DUT/inst/assert__valid_command_rd_data	Concurrent	SVA	on	0	1	

assertion:

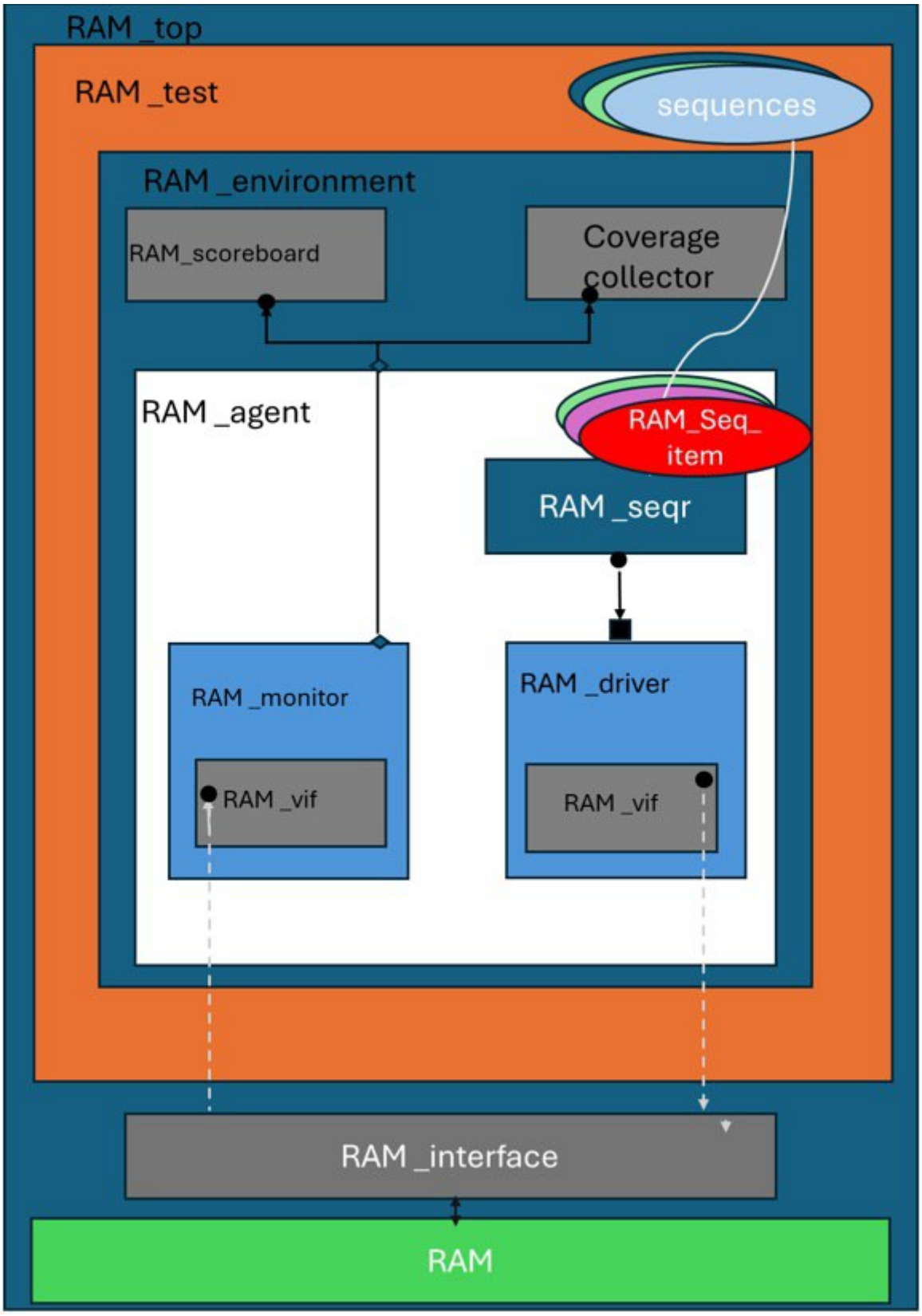
Feature	Assertion
Reset behavior: Whenever reset is asserted, all outputs (MISO, rx_valid, rx_data) must be low.	`@(posedge clk) (!rst_n
rx_valid timing: After any valid command sequence (write or read), rx_valid must assert exactly **10 cycles** later and SS_n must rise after that.	`@(posedge clk) disable iff (!rst_n) (valid_cmd_seq)
FSM transition – IDLE → CHK_CMD: When the slave is idle and SS_n is low, the next state must be CHK_CMD.	`@(posedge clk) disable iff(!rst_n) (cs==IDLE && !SS_n)
FSM transition – CHK_CMD → WRITE: When command check state detects MOSI=0, the FSM must transition to WRITE.	`@(posedge clk) disable iff(!rst_n) (cs==CHK_CMD && !SS_n && !MOSI)
FSM transition – CHK_CMD → READ_ADD: When in CHK_CMD with MOSI=1 and address not yet received, the FSM must go to READ_ADD.	`@(posedge clk) disable iff(!rst_n) (cs==CHK_CMD && !SS_n && MOSI && !received_address)
FSM transition – CHK_CMD → READ_DATA: When in CHK_CMD with MOSI=1 and address already received, the FSM must go to READ_DATA.	`@(posedge clk) disable iff(!rst_n) (cs==CHK_CMD && !SS_n && MOSI && received_address)

part 2 RAM :

verification plan :

RAM_reset_seq	When the reset_seq is start, the output is cleared	Directed at the start of the simulation	-	A checker in the scoreboard compare the result of the DUT and the Golden Model
RAM_read_only_seq	When the rx_valid is high and din [9:8] = 2'b10 the RAM will save the read address , and When the rx_valid is high and din [9:8] = 2'b11 the RAM will out the data of the saved address	Randomization with constrained to be read address and read data only	Coverpoint to check transaction ordering for din[9:8] and Between all bins of din[9:8] and rx_valid signal when it is high and Between din[9:8] when it equals read data and tx_valid when it is high	A checker in the scoreboard compare the result of the DUT and the Golden Model
RAM_write only_seq	When the rx_valid is high and din [9:8] = 2'b00 the RAM will save the write address , and When the rx_valid is high and din [9:8] = 2'b01 the RAM will write the data in the address	Randomization with constrained to be write address and write data only	Coverpoint to check transaction ordering for din[9:8] and Between all bins of din[9:8] and rx_valid signal when it is high and Between din[9:8] when it equals read data and tx_valid when it is high	A checker in the scoreboard compare the result of the DUT and the Golden Model
RAM_write_read_seq	When the rx_valid is high and din [9:8] = 2'b00 the RAM will save the write address , and When the rx_valid is high and din [9:8] = 2'b01 the RAM will write the data in the address , When the rx_valid is high and din [9:8] = 2'b10 the RAM will save the read address , and When the rx_valid is high and din [9:8] = 2'b11 the RAM will out the data of the saved address	Randomization with constrained to be after the address ,the data	Coverpoint to check transaction ordering for din[9:8] and Between all bins of din[9:8] and rx_valid signal when it is high and Between din[9:8] when it equals read data and tx_valid when it is high	A checker in the scoreboard compare the result of the DUT and the Golden Model

uvm structure :



design :

RAM.v > RAM

```
module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);

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

output reg [7:0] dout;
output reg       tx_valid;

reg [7:0] MEM [255:0];
reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else begin
        if (rx_valid) begin
            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];
                2'b11 : begin
                    //tx_valid <=1;
                    dout <= MEM[Rd_Addr];//the index should be the read address not the write adress
                end
                default : dout <= 0;
            endcase
        end
        tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
    end
end

endmodule
```

ram_if :

RAM_if.sv > ...

```
interface RAM_if(clk);
logic      [9:0] din;
logic            rst_n, rx_valid;
logic      [7:0] dout, ex_dout;
logic      tx_valid, ex_tx_valid;
input       clk;
endinterface
```

ram golden model :

RAM_golden_model.v > ...

```v
module RAM_golden_model (din, rx_valid, clk, rst_n, dout, tx_valid);

parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;

input [ADDR_SIZE+1 : 0] din;
input rx_valid, clk, rst_n;

output reg [ADDR_SIZE-1:0] dout;
output reg tx_valid;

reg [ADDR_SIZE-1:0] mem [MEM_DEPTH-1:0];
reg [ADDR_SIZE-1:0] ADDR_RD, ADDR_WR ;                    //internal

//Read/Write Operation
always @(posedge clk)begin
    if(~rst_n)begin
        dout <= 0;
        tx_valid <= 0;
        ADDR_RD <= 0;
        ADDR_WR <= 0;
    end
    else
    if(rx_valid) begin
       case (din[ADDR_SIZE+1:ADDR_SIZE])
       //Write
       2'b00 : begin
            ADDR_WR<=din[ADDR_SIZE-1:0];
       end
       2'b01 : begin
            mem[ADDR_WR]<=din[ADDR_SIZE-1:0];
       end
       //Read
       2'b10 : begin
            ADDR_RD<=din[ADDR_SIZE-1:0];
       end
       2'b11 : begin
            dout<=mem[ADDR_RD];
            tx_valid<=1;
       end
    endcase
    end
    else begin
         tx_valid=0;
    end
 end
endmodule
```

ram sva :

```
module RAM_assertions(din,clk,rst_n,rx_valid,dout,tx_valid);
input      [9:0] din;
input            clk, rst_n, rx_valid;
input reg [7:0] dout;
input reg       tx_valid;

reset: assert property  (@(posedge clk)   !rst_n
                  |=>     (dout==0) && (tx_valid==0));
//-------------------------------------------------------------------------------------------
tx_deasserted: assert property (@(posedge clk)   (!(din[9:8]==2'b11) && rx_valid)
                  |=>     tx_valid==0 );
//-------------------------------------------------------------------------------------------
tx_asserted: assert property (@(posedge clk) disable iff(!rst_n) ((din[9:8]==2'b11) && rx_valid)
                  |=>    (($rose(tx_valid)) ##1 ($fell(tx_valid)) [ -> 1] ) );
//-------------------------------------------------------------------------------------------
write_assert:assert property (@(posedge clk)   (din[9:8]==2'b00) && rx_valid
                  |=>    ((din[9:8]==2'b01) && rx_valid) [ -> 1]  );
//-------------------------------------------------------------------------------------------
read_assert: assert property (@(posedge clk)    (din[9:8]==2'b10) && rx_valid
                  |=>     ((din[9:8]==2'b11) && rx_valid) [ -> 1] );

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

reset_cover: cover property (@(posedge clk)   !rst_n   |=>     (dout==0) && (tx_valid==0));

tx_deasserted_cover: cover property (@(posedge clk)   (!(din[9:8]==2'b11) && rx_valid)     |=>     tx_valid==0 );

tx_asserted_cover: cover property (@(posedge clk)    ((din[9:8]==2'b11) && rx_valid)     |=>
                                    (($rose(tx_valid)|| (din[9:8]==2'b11)||!rst_n) ##1 ($fell(tx_valid) || (din[9:8]==2'b11)||!$past(rst_n))) );

write_cover:cover property (@(posedge clk)   (din[9:8]==2'b00) && rx_valid   |=>   ##1 ((din[9:8]==2'b01) && rx_valid) [ -> 1]  );

read_cover: cover property (@(posedge clk)    (din[9:8]==2'b10) && rx_valid   |=>    ##1 ((din[9:8]==2'b11) && rx_valid) [ -> 1] );

endmodule
```

ram top :

```
import uvm_pkg::*;
import RAM_test_pkg::*;
`include "uvm_macros.svh"
module top();
    bit clk;

    initial begin
        clk=0;
        forever begin
            #1 clk=~clk;
        end
    end

    RAM_if IF(clk);

    RAM dut (IF.din, IF.clk, IF.rst_n, IF.rx_valid, IF.dout, IF.tx_valid);

    bind RAM assertions sva(IF.din, IF.clk, IF.rst_n, IF.rx_valid, IF.dout, IF.tx_valid);

    RAM_golden_model u1 (IF.din, IF.rx_valid, IF.clk, IF.rst_n, IF.ex_dout, IF.ex_tx_valid);

    initial begin
        uvm_config_db #(virtual RAM_if):: set (null,"uvm_test_top", "CFG",IF);
        run_test("RAM_test");
     end
endmodule
```

ram test :

```
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("run_phase","reset asserted",UVM_LOW);
            r_seq.start(env.agt.sqr);
            `uvm_info("run_phase","reset deasserted",UVM_LOW);

            `uvm_info("run_phase","write_only_seq stimulus generation started",UVM_LOW);
            wr_only_seq.start(env.agt.sqr);
            `uvm_info("run_phase","stimulus generation ended",UVM_LOW);

            `uvm_info("run_phase","reset asserted",UVM_LOW);
            r_seq.start(env.agt.sqr);
            `uvm_info("run_phase","reset deasserted",UVM_LOW);

            `uvm_info("run_phase","write_and_read_seq stimulus generation started",UVM_LOW);
            wr_and_rd_seq.start(env.agt.sqr);
            `uvm_info("run_phase","stimulus generation ended",UVM_LOW);

            `uvm_info("run_phase","read_only_seq stimulus generation started",UVM_LOW);
            rd_only_seq.start(env.agt.sqr);
            `uvm_info("run_phase","stimulus generation ended",UVM_LOW);

            `uvm_info("run_phase","reset asserted",UVM_LOW);
            r_seq.start(env.agt.sqr);
            `uvm_info("run_phase","reset deasserted",UVM_LOW);

            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

```
            if (!uvm_config_db#(virtual RAM_if) ::get(this,"","CFG",conf.RAM_vif))
                `uvm_fatal("build_phase", "unable to get the interface to the config");
            conf.is_active = UVM_ACTIVE;
            uvm_config_db#(RAM_configer)::set(this,"*","CFG",conf);
        endfunction
```

ram env :

RAM_environment.sv > ...

```
1   package RAM_environment_pkg;
2   import uvm_pkg::*;
3   import RAM_agent_pkg::*;
4   import RAM_scoreboard_pkg::*;
5   import RAM_coverage_collector_pkg::*;
6   `include "uvm_macros.svh"
7   class RAM_environment extends uvm_env;
8       `uvm_component_utils(RAM_environment)
9       RAM_agent agt;
10      RAM_scoreboard sc;
11      RAM_coverage_collector cc;
12      function new(string name = "RAM_envirnment", uvm_component parent = null);
13          super.new(name, parent);
14      endfunction
15
16      function void build_phase(uvm_phase phase);
17          super.build_phase(phase);
18          agt = RAM_agent :: type_id :: create("agt", this);
19          sc = RAM_scoreboard::type_id::create("sc", this);
20          cc = RAM_coverage_collector::type_id::create("cc",this);
21      endfunction
22
23      function void connect_phase(uvm_phase phase);
24          super.connect_phase (phase);
25          agt.agent_ap.connect(sc.sb_export);
26          agt.agent_ap.connect(cc.covr);
27      endfunction
28  endclass
29  endpackage
```

ram agent :

RAM_agent.sv > ...

```
package RAM_agent_pkg;
    `include "uvm_macros.svh"
    import uvm_pkg::*;
    import RAM_driver_pkg::*;
    import RAM_monitor_pkg::*;
    import RAM_seqr_pkg::*;
    import RAM_configer_pkg::*;
    import RAM_seq_item_pkg::*;
    class RAM_agent extends uvm_agent;
        `uvm_component_utils(RAM_agent)
        RAM_seqr sqr;
        RAM_driver driv;
        RAM_monitor mon;
        RAM_configer cfg;
        uvm_analysis_export #(RAM_seq_item) agent_ap;

        function new(string name = "agent", uvm_component parent);
            super.new(name, parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if(!uvm_config_db#(RAM_configer)::get(this,"","CFG",cfg))begin
                `uvm_fatal("build_phase","unable to configuration object");
            end
            if (cfg.is_active==UVM_ACTIVE) begin
                sqr=RAM_seqr::type_id::create("sqr",this);
                driv=RAM_driver::type_id::create("driv",this);
            end

            mon=RAM_monitor::type_id::create("mon",this);
            agent_ap=new("agent_ap",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if (cfg.is_active==UVM_ACTIVE) begin
                driv.seq_item_port.connect(sqr.seq_item_export);
                driv.RAM_vif=cfg.RAM_vif;
            end
            mon.RAM_vif=cfg.RAM_vif;
            mon.mon_ap.connect(agent_ap);
        endfunction
    endclass
endpackage
```

ram 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_vif;
        RAM_seq_item s_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
                s_item = RAM_seq_item::type_id::create("s_item");

                seq_item_port.get_next_item(s_item);
                RAM_vif.din=s_item.din;
                RAM_vif.rx_valid=s_item.rx_valid;
                RAM_vif.rst_n=s_item.rst_n;

                @(negedge RAM_vif.clk);

                seq_item_port.item_done();
            end
        endtask
    endclass
endpackage
```

ram 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 s_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
                s_item = RAM_seq_item::type_id::create("s_item");
                    @(negedge RAM_vif.clk);
                    s_item.din=RAM_vif.din;
                    s_item.rx_valid=RAM_vif.rx_valid;
                    s_item.rst_n=RAM_vif.rst_n;
                    s_item.tx_valid=RAM_vif.tx_valid;
                    s_item.dout=RAM_vif.dout;
                    s_item.ex_tx_valid=RAM_vif.ex_tx_valid;
                    s_item.ex_dout=RAM_vif.ex_dout;
                    mon_ap.write(s_item);
                    `uvm_info("run_phase", s_item.convert2string(), UVM_HIGH);
            end
        endtask
    endclass
endpackage
```

ram sequencer:

```
RAM_seqr.sv > ...
1   package RAM_seqr_pkg;
2       import uvm_pkg::*;
3       import RAM_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5
6       class RAM_seqr extends uvm_sequencer #(RAM_seq_item);
7           `uvm_component_utils(RAM_seqr);
8
9           function new (string name = "RAM_seqr", uvm_component parent = null);
10              super.new(name,null);
11          endfunction
12      endclass
13  endpackage
```

ram item :

```
RAM_seq_item.sv > {} RAM_seq_item_pkg > RAM_seq_item
1   package RAM_seq_item_pkg;
3       `include "uvm_macros.svh"
4
5       class RAM_seq_item extends uvm_sequence_item;
6           `uvm_object_utils(RAM_seq_item)
7           rand logic      [9:0] din;
8           rand logic            rst_n,   rx_valid;
9           logic                 tx_valid,ex_tx_valid;
0           logic           [7:0] dout,    ex_dout;
1           logic           [1:0] old=0;
2
3           function new(string name = "RAM_seq_item");
4               super.new(name);
5           endfunction
6
7           function string convert2string ();
8               return $sformatf ("%0s din = %0d,rst_n = %0d, rx_valid = %0d, dout = %0d, tx_valid = %0b, tx_data = %0d", super.convert2string(),
9                                 din,rst_n,rx_valid, dout, tx_valid, tx_data);
0           endfunction
1
2           function string convert2string_stimulus();
3               return $sformatf("din = %0d,rst_n = %0d, rx_valid = %0d",din,rst_n,rx_valid);
4           endfunction
5
6           constraint reset{ rst_n dist{0:= 1, 1:= 99};}
7           constraint rxvalid{rx_valid dist{0:= 1, 1:= 99};}
8
9           function void post_randomize;
0             old = din[9:8];
1           endfunction
2       endclass
3   endpackage
```

ram rst seq:

```systemverilog
package RAM_reset_seq_pkg;
    import RAM_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class RAM_reset_seq extends uvm_sequence #(RAM_seq_item);
        `uvm_object_utils (RAM_reset_seq)
        RAM_seq_item s_item;

        function  new (string name = "RAM_reset_seq");
            super.new(name);
        endfunction

        task body();
            s_item = RAM_seq_item::type_id::create("s_item");
            start_item(s_item);
            s_item.rst_n=1;
            s_item.rx_valid=0;
            s_item.din=0;
            s_item.rst_n=0;
            finish_item(s_item);
        endtask
    endclass
endpackage
```

ram write seq :

```systemverilog
package RAM_write_only_seq_pkg;
    import uvm_pkg::*;
    import RAM_seq_item_pkg::*;
    `include "uvm_macros.svh"
    class RAM_write_only_seq extends uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_write_only_seq)
        RAM_seq_item s_item;

        function new(string name = "RAM_write_only_seq");
            super.new(name);
        endfunction

        task body ();
            s_item = RAM_seq_item::type_id::create("s_item");
            repeat(1000) begin
            start_item(s_item);
            assert(s_item.randomize() with{(din[9:8]==2'b10) || (din[9:8]==2'b00);});
            finish_item(s_item);
            end
        endtask
    endclass
endpackage
```

ram read seq :

RAM_read_only_seq.sv > ...

```
package RAM_read_only_seq_pkg;
    import uvm_pkg::*;
    import RAM_seq_item_pkg::*;
    `include "uvm_macros.svh"
    class RAM_read_only_seq extends uvm_sequence #(RAM_seq_item);
        `uvm_object_utils(RAM_read_only_seq)
        RAM_seq_item s_item;

        function new(string name = "RAM_read_only_seq");
            super.new(name);
        endfunction

        task body ();
            s_item = RAM_seq_item::type_id::create("s_item");
            repeat(1000) begin
            start_item(s_item);
            assert(s_item.randomize() with{ if (din[9:8]==2'b10)  (din[9:8]==2'b11) ;
                                            if (din[9:8]==2'b11)  (din[9:8]==2'b10); });
            finish_item(s_item);
            end
        endtask
    endclass
endpackage
```

ram read write seq :

```
RAM_write_and_read_seq.sv > ...
1   package RAM_write_and_read_seq_pkg;
2       import uvm_pkg::*;
3       import RAM_seq_item_pkg::*;
4       `include "uvm_macros.svh"
5       class RAM_write_and_read_seq extends uvm_sequence #(RAM_seq_item);
6           `uvm_object_utils(RAM_write_and_read_seq)
7           RAM_seq_item s_item;
8
9           function new(string name = "RAM_write_and_read_seq");
10              super.new(name);
11          endfunction
12
13          task body ();
14              s_item = RAM_seq_item::type_id::create("s_item");
15              repeat(1000) begin
16              start_item(s_item);
17              assert(s_item.randomize() with{ if(old==2'b00) {
18                                                  din[9:8] inside {2'b00, 2'b01};
19                                              }
20                                              else if(old==2'b01){
21                                                  din[9:8] dist {2'b00 :=40, 2'b10 :=60};
22                                              }
23                                              else if(old==2'b10){
24                                                  din[9:8] inside {2'b10, 2'b11};
25                                              }
26                                              else if(old==2'b11){
27                                              din[9:8] dist {2'b00 :=60, 2'b10 :=40};
28                                              }
29                                              }
30                                              );
31              finish_item(s_item);
32              end
33          endtask
34      endclass
35  endpackage
```

ram config :

```
RAM_configer.sv > ...
1   package RAM_configer_pkg;
2   `include "uvm_macros.svh"
3   import uvm_pkg::*;
4   class RAM_configer extends uvm_object;
5       `uvm_object_utils(RAM_configer)
6
7           virtual RAM_if RAM_vif;
8           uvm_active_passive_enum is_active;
9       function  new(string name = "RAM_configer");
10          super.new(name);
11      endfunction
12  endclass
13  endpackage
```

ram coverage :

```
package RAM_coverage_collector_pkg;
    `include "uvm_macros.svh"
    import uvm_pkg::*;
    import RAM_seq_item_pkg::*;
    class RAM_coverage_collector extends uvm_component;
        `uvm_component_utils(RAM_coverage_collector)
        uvm_analysis_export #(RAM_seq_item) covr;
        uvm_tlm_analysis_fifo #(RAM_seq_item) fifo;
        RAM_seq_item s_item;

        covergroup cov;
          control_bits: coverpoint s_item.din[9:8] {
            bins four_possible = {[3:0]};
            bins w_data_after_w_address = (2'b00 => 2'b01);
            bins r_data_after_r_adderss = (2'b10 => 2'b11);
            bins w_address_w_data_r_address_r_data = (2'b00 => 2'b01=>2'b10 => 2'b11);
          }
          rx_valid_cp: coverpoint s_item.rx_valid;
          tx_valid_cp: coverpoint s_item.tx_valid;
          cross_rx: cross control_bits, rx_valid_cp{
                bins rx_high = binsof (control_bits) && binsof (rx_valid_cp) intersect {0};
          }
          cross_tx: cross control_bits, tx_valid_cp{
                bins tx_high = binsof (control_bits) intersect {2'b01} && binsof (tx_valid_cp) intersect {1};
          }

        endgroup

        function new(string name = "coverage_collector", uvm_component parent = null);
            super.new(name, parent);
            cov = new ();
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            covr = new("covr",this);
            fifo  = new("fifo",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            covr.connect(fifo.analysis_export);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                fifo.get(s_item);
                cov.sample();
                `uvm_info("COV", $sformatf("  stimulus%0d", s_item.convert2string_stimulus), UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

ram scoreboard :

```
package RAM_scoreboard_pkg;

    import uvm_pkg::*;
    import RAM_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class RAM_scoreboard extends uvm_scoreboard;
        //define the export and the fifo
        `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 s_item;
        //the expcted value
        logic [3:0] expected_value;
        //the error counter and the correct counter
        int errors=0, corrects=0;
        //the essential
        function new(string name = "RAM_scoreboard", uvm_component parent = null);
            super.new(name, parent);
        endfunction
        //build phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export = new("sb_export", this);
            sb_fifo   = new("sb_fifo", this);
        endfunction
        //connect phase
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction
        //run phase
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(s_item);
                if((s_item.dout != s_item.ex_dout)&&(s_item.tx_valid!=s_item.ex_tx_valid)) begin
                    `uvm_error("run_phase", $sformatf("comparsion faild"));
                    errors++;
                    `uvm_info("COV", $sformatf("  stimulus %0d ", s_item.convert2string), UVM_HIGH)
                end
                else begin
                    corrects++;
                end
            end
        endtask

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("the number of errors=0d%0d",errors),UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("the number of corrects=0d%0d",corrects),UVM_MEDIUM)
        endfunction
    endclass
endpackage
```

ram do file :

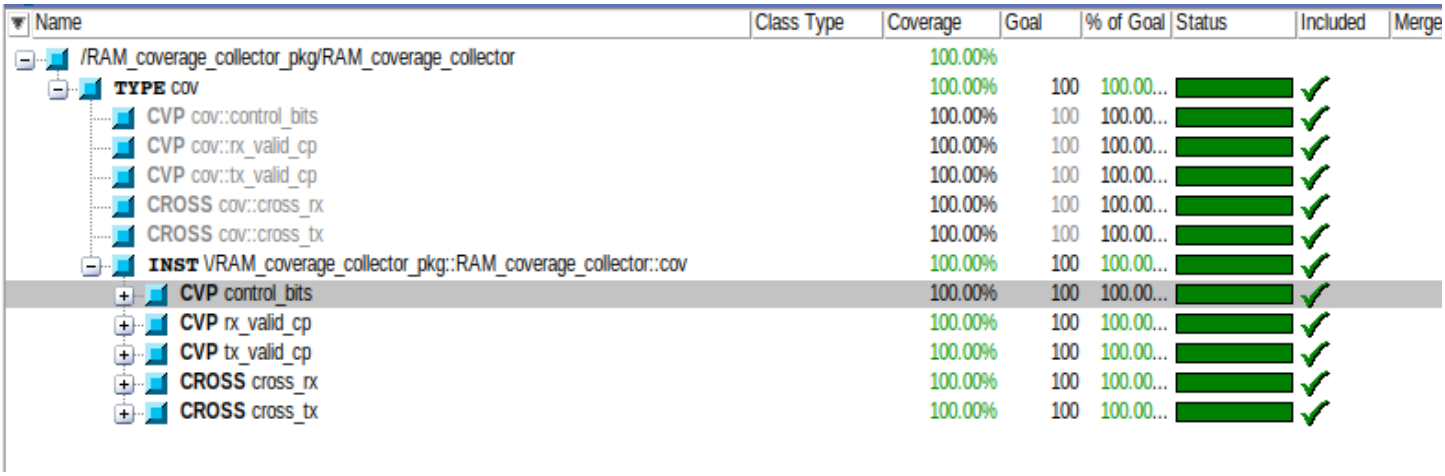
```
vlib work
vmap work work
vlog -cover bcesft +acc -f src_files.list
vsim -coverage -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/IF/*
coverage save RAM.ucdb -onexit
run -all
coverage exclude -src RAM.v -line 29 -code b
coverage exclude -src RAM.v -line 29 -code s
#quit -sim
#vcover report RAM.ucdb -details -code bcesft -output coverage_report.txt
```

coverage report :

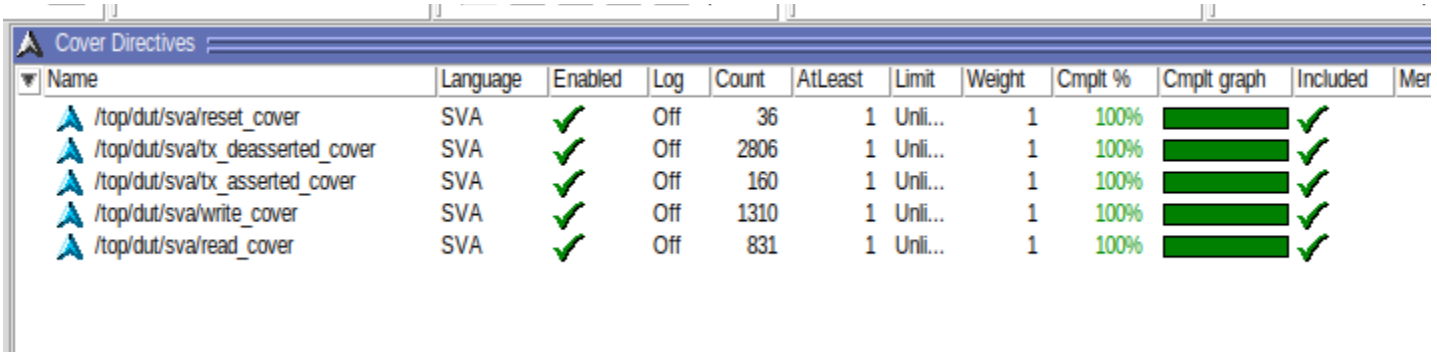
```
Coverage Report by instance with details

=================================================================================
=== Instance: /top/IF
=== Design Unit: work.RAM_if
=================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         62        62         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /top/IF --

                                                    Node      1H->0L      0L->1H  "Coverage"
                                                    ----------------------------------------

Total Node Count     =         31
Toggled Node Count   =         31
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (62 of 62 bins)

=================================================================================
=== Instance: /top/dut
=== Design Unit: work.RAM
---------------------------------------------------------------------------------


Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      10        10         0   100.00%
```

```
Statement Coverage for instance /top/dut --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File RAM.v
    12              1                       3003
    14              1                         37
    15              1                         37
    16              1                         37
    17              1                         37
    22              1                       1298
    23              1                        657
    24              1                        819
    27              1                        158
    32              1                       2966
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         76        76         0   100.00%
```

functional coverage :

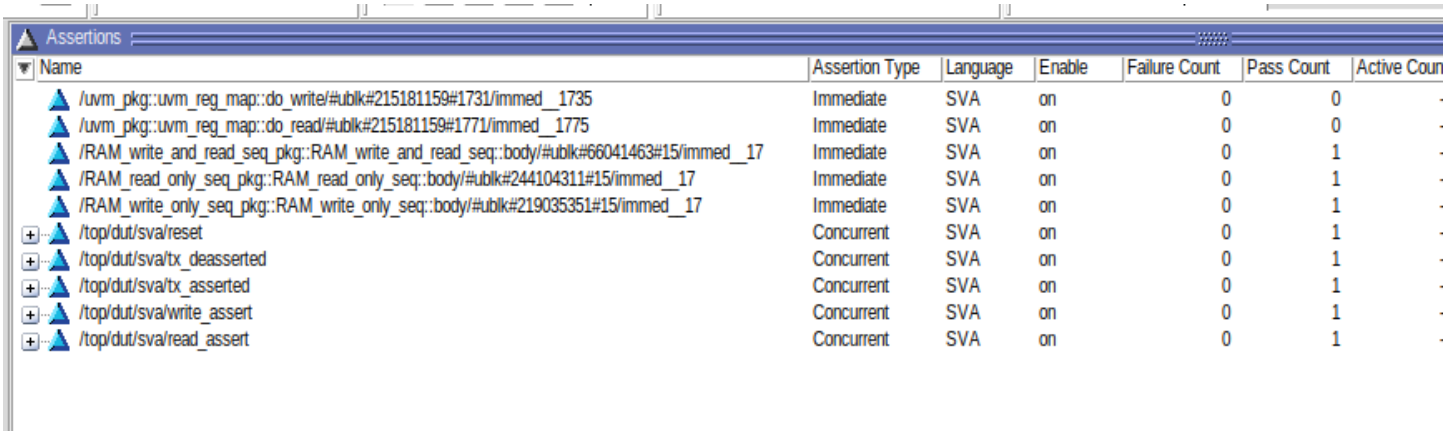


assertion coverage :



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Mer
/top/dut/sva/reset_cover	SVA	✓	Off	36	1	Unli...	1	100%		✓	
/top/dut/sva/tx_deasserted_cover	SVA	✓	Off	2806	1	Unli...	1	100%		✓	
/top/dut/sva/tx_asserted_cover	SVA	✓	Off	160	1	Unli...	1	100%		✓	
/top/dut/sva/write_cover	SVA	✓	Off	1310	1	Unli...	1	100%		✓	
/top/dut/sva/read_cover	SVA	✓	Off	831	1	Unli...	1	100%		✓	



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Coun
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	
/RAM_write_and_read_seq_pkg::RAM_write_and_read_seq::body/#ublk#66041463#15/immed__17	Immediate	SVA	on	0	1	
/RAM_read_only_seq_pkg::RAM_read_only_seq::body/#ublk#244104311#15/immed__17	Immediate	SVA	on	0	1	
/RAM_write_only_seq_pkg::RAM_write_only_seq::body/#ublk#219035351#15/immed__17	Immediate	SVA	on	0	1	
/top/dut/sva/reset	Concurrent	SVA	on	0	1	
/top/dut/sva/tx_deasserted	Concurrent	SVA	on	0	1	
/top/dut/sva/tx_asserted	Concurrent	SVA	on	0	1	
/top/dut/sva/write_assert	Concurrent	SVA	on	0	1	
/top/dut/sva/read_assert	Concurrent	SVA	on	0	1	

assertions :

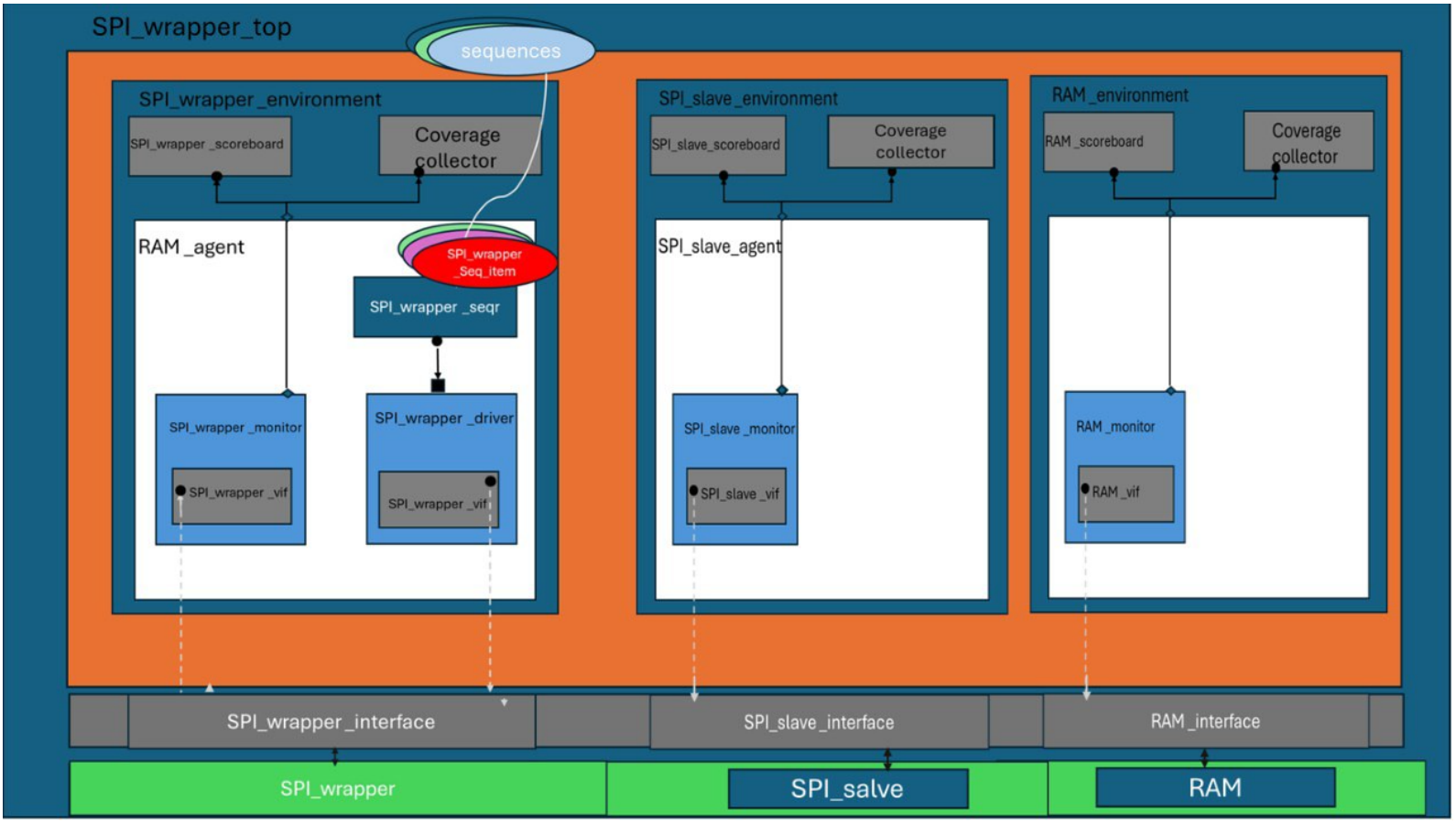
Feature	Assertion
Reset behavior: Whenever reset is asserted, both tx_valid and dout must be low.	`@(posedge clk) (!rst_n
Access control: During any address or data input phase (write_add, write_data, or read_add), tx_valid must remain low.	`@(posedge clk) disable iff(!rst_n) (all_seq)
tx_valid pulse behavior: When a read data command (din[9:8]==2'b11) occurs, tx_valid must rise and eventually fall.	`@(posedge clk) disable iff(!rst_n) (din[9:8]==2'b11 && rx_valid && !tx_valid)
Write Address → Write Data sequence: Every write address operation must eventually be followed by a write data operation.	`@(posedge clk) disable iff(!rst_n) write_add
Read Address → Read Data sequence: Every read address operation must eventually be followed by a read data operation.	`@(posedge clk) disable iff(!rst_n) read_add

part 3 wrapper :

verification plan :

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SPI_wrapper_reset	when the reset is asserted the MISO should be low	directed in the start of the simulation	-	a checker in the score board that compare the result from the DUT and the golden model
SPI_wrapper_1	ss_n transation length (13/23 cycle)	randomize ss_n pulse control_in MOSI sequence	coverbins 13 cycle and 23 cycle frame	a checker in the score board that compare the result from the DUT and the golden model
SPI_wrapper_2	write throw path from SPI to the RAM	SPI MOSI commands 000/001	cross MOSI command, write_RAM_address	a checker in the score board that compare the result from the DUT and the golden model
SPI_wrapper_3	read throw path from RAM to the MISO	SPI MOSI commands 110/111	cross command type and the MISO response	a checker in the score board that compare the result from the DUT and the golden model
SPI_wrapper_4	tx_valid and rx_valid handshake between modules	normal traffic with passive RAM and slave agent	cross rx_valid, tx_valid	a checker in the score board that compare the result from the DUT and the golden model
SPI_wrapper_5	MISO stable when not in read_data	mix read / write operations	cross MISO stable periods	a checker in the score board that compare the result from the DUT and the golden model

uvm structure :



design :

```
module WRAPPER (SPI_wrapper_if.DUT SPI_wrapperif);

    SPI_slave_if   SPI_slaveif(SPI_wrapperif.clk);

    wire [9:0] rx_data_din;
    wire       rx_valid;
    wire       tx_valid;
    wire [7:0] tx_data_dout;

    assign SPI_slaveif.MOSI  = SPI_wrapperif.MOSI;
    assign SPI_slaveif.SS_n  = SPI_wrapperif.SS_n;
    assign SPI_slaveif.rst_n = SPI_wrapperif.rst_n;

    assign SPI_slaveif.tx_data  = tx_data_dout;
    assign SPI_slaveif.tx_valid = tx_valid;

    assign rx_data_din = SPI_slaveif.rx_data;
    assign rx_valid = SPI_slaveif.rx_valid;


    assign SPI_wrapperif.MISO = SPI_slaveif.MISO;



RAM   RAM_instance   (rx_data_din,SPI_wrapperif.clk,SPI_wrapperif.rst_n,rx_valid,tx_data_dout,tx_valid);
SLAVE SLAVE_instance (SPI_slaveif);

endmodule
```

wrapper_if :

```
interface SPI_wrapper_if(clk);
    input bit clk;
    logic  MOSI, SS_n, rst_n;
    logic  MISO;

    logic MISO_GM;

    modport DUT (
    input clk,SS_n,MOSI,rst_n,
    output MISO
    );

    // modport GOLDEN_MODEL (
    // input clk,SS_n,MOSI,rst_n,
    // output MISO_GM
    // );

endinterface
```

wrapper golden model :

```
module WRAPPER_GM (MOSI,MISO,SS_n,clk,rst_n);

input  MOSI, SS_n, clk, rst_n;
output MISO;

wire [9:0] rx_data_din;
wire       rx_valid;
wire       tx_valid;
wire [7:0] tx_data_dout;

RAM_golden_model   RAM_instance   (rx_data_din, rx_valid, clk, rst_n, tx_data_dout, tx_valid);
SPI_slave_GM SLAVE_instance (MOSI, SS_n, clk, rst_n, tx_valid, tx_data_dout, MISO,rx_data_din,rx_valid);

endmodule
```

wrapper sva :

```
import shared_pkg::*;


module SPI_wrapper_sva(SPI_wrapper_if.DUT SPI_wrapperif);


    property MISO_reset;
        @(posedge SPI_wrapperif.clk) (!SPI_wrapperif.rst_n) |=> (~SPI_wrapperif.MISO);
    endproperty

    assert property (MISO_reset);
    cover  property (MISO_reset);

    property rx_valid_reset;
        @(posedge SPI_wrapperif.clk) (!SPI_wrapperif.rst_n) |=> (~WRAPPER.rx_valid);
    endproperty

    assert property (rx_valid_reset);
    cover  property (rx_valid_reset);

    property rx_data_reset;
        @(posedge SPI_wrapperif.clk) (!SPI_wrapperif.rst_n) |=> (WRAPPER.rx_data_din == 0);
    endproperty

    assert property (rx_data_reset);
    cover  property (rx_data_reset);

    property MISO_satble;
        @(posedge SPI_wrapperif.clk) disable iff (!SPI_wrapperif.rst_n) (WRAPPER.SLAVE_instance.cs != READ_DATA) |-> $stable(SPI_wrapperif.MISO);
    endproperty

    assert property (MISO_satble);
    cover  property (MISO_satble);


endmodule
```

wrapper top :

```
import uvm_pkg::*;
`include "uvm_macros.svh"

import SPI_wrapper_test_pkg::*;

module SPI_wrapper_top();
    bit clk;

    initial begin
        forever begin
            #1 clk = ~clk;
        end
    end
    SPI_wrapper_if SPI_wrapperif(clk);
    SPI_slave_if   SPI_slaveif(clk);
    RAM_if         RAMif(clk);

    WRAPPER SPI_wrapper_instance (SPI_wrapperif);

    //spi_slave wiring
    assign SPI_slaveif.MOSI = SPI_wrapperif.MOSI;
    assign SPI_slaveif.MISO = SPI_wrapperif.MISO;
    assign SPI_slaveif.SS_n = SPI_wrapperif.SS_n;
    assign SPI_slaveif.rst_n= SPI_wrapperif.rst_n;
    assign SPI_slaveif.rx_data = SPI_wrapper_instance.rx_data_din;
    assign SPI_slaveif.rx_valid = SPI_wrapper_instance.rx_valid;
    assign SPI_slaveif.tx_data = SPI_wrapper_instance.tx_data_dout;
    assign SPI_slaveif.tx_valid = SPI_wrapper_instance.tx_valid;

    //ram wiring
    assign RAMif.din = SPI_wrapper_instance.rx_data_din;
    assign RAMif.rst_n = SPI_wrapperif.rst_n;
    assign RAMif.rx_valid = SPI_wrapper_instance.rx_valid;
    assign RAMif.dout = SPI_wrapper_instance.tx_data_dout;
    assign RAMif.tx_valid = SPI_wrapper_instance.tx_valid;

    WRAPPER_GM gm(SPI_wrapperif.MOSI, SPI_wrapperif.MISO_GM, SPI_wrapperif.SS_n, SPI_wrapperif.clk, SPI_wrapperif.rst_n);

    bind SLAVE SPI_slave_sva slave_sva (SPI_slaveif);
    bind WRAPPER SPI_wrapper_sva wrapper_sva (SPI_wrapperif);
    bind RAM RAM_assertions ram_sva(RAMif.din,RAMif.clk,RAMif.rst_n,RAMif.rx_valid,RAMif.dout,RAMif.tx_valid);
    initial begin
        uvm_config_db #(virtual SPI_wrapper_if) ::set(null, "uvm_test_top", "SPI_WRAPPER_VIF",SPI_wrapperif);
        uvm_config_db #(virtual SPI_slave_if)   ::set(null, "uvm_test_top", "SPI_SLAVE_VIF",SPI_slaveif);
        uvm_config_db #(virtual RAM_if)         ::set(null, "uvm_test_top", "RAM_VIF",RAMif);
        run_test("SPI_wrapper_test");
    end

endmodule
```

wrapper test :

```
package SPI_wrapper_test_pkg;

    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_env_pkg::*;
    import SPI_slave_env_pkg::*;
    import RAM_environment_pkg::*;
    import SPI_wrapper_config_obj_pkg::*;
    import SPI_slave_config_pkg::*;
    import RAM_configer_pkg::*;
    import SPI_wrapper_sequence_pkg::*;

    class SPI_wrapper_test extends uvm_test;
        `uvm_component_utils(SPI_wrapper_test)

        SPI_wrapper_env wrapper_env;
        SPI_slave_env slave_env;
        RAM_environment ram_env;
        SPI_wrapper_config_obj wrapper_config_obj_test;
        SPI_slave_config slave_config_obj_test;
        RAM_configer ram_config_obj_test;

        SPI_wrapper_reset_sequence reset_seq;
        SPI_wrapper_write_sequence write_seq;
        SPI_wrapper_read_sequence read_seq;
        SPI_wrapper_read_write_sequence read_write_seq;

        function new (string name, uvm_component parent);
            super.new(name, parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            wrapper_env = SPI_wrapper_env::type_id::create("wrapper_env", this);
            slave_env   = SPI_slave_env::type_id::create("slave_env", this);
            ram_env     = RAM_environment::type_id::create("ram_env", this);

            reset_seq   = SPI_wrapper_reset_sequence::type_id::create("reset_seq");
            write_seq   = SPI_wrapper_write_sequence::type_id::create("write_seq");
            read_seq    = SPI_wrapper_read_sequence::type_id::create("read_seq");
            read_write_seq = SPI_wrapper_read_write_sequence::type_id::create("read_write_seq");


            wrapper_config_obj_test = SPI_wrapper_config_obj::type_id::create("wrapper_config_obj_test");
            slave_config_obj_test = SPI_slave_config::type_id::create("slave_config_obj_test");
            ram_config_obj_test = RAM_configer::type_id::create("ram_config_obj_test");
```

```
    if(!( uvm_config_db #(virtual SPI_wrapper_if)::get(this,"","SPI_WRAPPER_VIF",wrapper_config_obj_test.SPI_wrapper_vif)))begin
        `uvm_fatal("build_phase","Test - wrapper - Unable to get the virtual interface");
    end
    if(!( uvm_config_db #(virtual SPI_slave_if)::get(this,"","SPI_SLAVE_VIF",slave_config_obj_test.SPI_slave_vif)))begin
        `uvm_fatal("build_phase","Test - slave - Unable to get the virtual interface");
    end
    if(!( uvm_config_db #(virtual RAM_if)::get(this,"","RAM_VIF",ram_config_obj_test.RAM_vif)))begin
        `uvm_fatal("build_phase","Test - RAM - Unable to get the virtual interface");
    end
    uvm_config_db #(SPI_wrapper_config_obj)::set(this,"wrapper_env.*","SPI_WRAPPER_CFG",wrapper_config_obj_test);
    wrapper_config_obj_test.is_active = UVM_ACTIVE;

    uvm_config_db #(SPI_slave_config)::set(this,"slave_env.*","SPI_SLAVE_CFG",slave_config_obj_test);
    slave_config_obj_test.is_active = UVM_PASSIVE;

    uvm_config_db #(RAM_configer)::set(this,"ram_env.*","RAM_CFG",ram_config_obj_test);
    ram_config_obj_test.is_active = UVM_PASSIVE;


endfunction

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);

    //reset sequence
    `uvm_info("run_phase", "Reset asserted", UVM_LOW);
    reset_seq.start(wrapper_env.agent.sequencer);
    `uvm_info("run_phase", "Reset deasserted", UVM_LOW);

    //write sequence
    `uvm_info("run_phase", "write sequence started", UVM_LOW);
    write_seq.start(wrapper_env.agent.sequencer);
    `uvm_info("run_phase", "write sequence ended", UVM_LOW);

    //read sequence
    `uvm_info("run_phase", "read sequence started", UVM_LOW);
    read_seq.start(wrapper_env.agent.sequencer);
    `uvm_info("run_phase", "read sequence ended", UVM_LOW);

    //read write sequence
    `uvm_info("run_phase", "read_write sequence started", UVM_LOW);
    read_write_seq.start(wrapper_env.agent.sequencer);
    `uvm_info("run_phase", "read_write sequence ended", UVM_LOW);
```

```
        phase.drop_objection(this);
    endtask

endclass

endpackage
```

wrapper env:

```
package SPI_wrapper_env_pkg;

    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_agent_pkg::*;
    import SPI_wrapper_scoreboard_pkg::*;
    import SPI_wrapper_coverage_pkg::*;

    class SPI_wrapper_env extends uvm_env;
        `uvm_component_utils(SPI_wrapper_env)

        SPI_wrapper_agent agent;
        SPI_wrapper_scoreboard scoreboard;
        SPI_wrapper_coverage coverage;

        function new(string name, uvm_component parent);
            super.new(name, parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agent = SPI_wrapper_agent::type_id::create("agent", this);
            scoreboard = SPI_wrapper_scoreboard::type_id::create("scoreboard", this);
            coverage = SPI_wrapper_coverage::type_id::create("coverage", this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agent.SPI_wrapper_agent_aport.connect(scoreboard.SPI_wrapper_scoreboard_aexport);
            agent.SPI_wrapper_agent_aport.connect(coverage.SPI_wrapper_coverage_aexport);
        endfunction

    endclass

endpackage
```

wrapper agent :

```
package SPI_wrapper_agent_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_driver_pkg::*;
    import SPI_wrapper_monitor_pkg::*;
    import SPI_wrapper_sequencer_pkg::*;
    import SPI_wrapper_config_obj_pkg::*;
    import SPI_wrapper_seq_item_pkg::*;

    class SPI_wrapper_agent extends uvm_agent;
        `uvm_component_utils(SPI_wrapper_agent)

        SPI_wrapper_driver driver;
        SPI_wrapper_monitor monitor;
        SPI_wrapper_sequencer sequencer;
        SPI_wrapper_config_obj SPI_wrapper_agent_config_obj;
        uvm_analysis_port #(SPI_wrapper_seq_item) SPI_wrapper_agent_aport;

        function new(string name, uvm_component parent);
            super.new(name, parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if(!(uvm_config_db #(SPI_wrapper_config_obj)::get(this,"","SPI_WRAPPER_CFG",SPI_wrapper_agent_config_obj)))begin
                `uvm_fatal("build_phase","agent - wrapper - Unable to get the configuration object")
            end
            if(SPI_wrapper_agent_config_obj.is_active == UVM_ACTIVE)begin
                driver = SPI_wrapper_driver::type_id::create("driver", this);
                sequencer = SPI_wrapper_sequencer::type_id::create("sequencer", this);
            end

            monitor = SPI_wrapper_monitor::type_id::create("monitor", this);
            SPI_wrapper_agent_aport = new("SPI_wrapper_agent_aport", this);

        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(SPI_wrapper_agent_config_obj.is_active == UVM_ACTIVE)begin
                driver.SPI_wrapper_driver_vif = SPI_wrapper_agent_config_obj.SPI_wrapper_vif;
                driver.seq_item_port.connect(sequencer.seq_item_export);
            end

            monitor.SPI_wrapper_monitor_vif = SPI_wrapper_agent_config_obj.SPI_wrapper_vif;
            monitor.SPI_wrapper_monitor_aport.connect(SPI_wrapper_agent_aport);
        endfunction
    endclass
endpackage
```

wrapper driver :

```
package SPI_wrapper_driver_pkg;

    import uvm_pkg::*;
    `include"uvm_macros.svh"
    import SPI_wrapper_seq_item_pkg::*;

    class SPI_wrapper_driver extends uvm_driver #(SPI_wrapper_seq_item);
        `uvm_component_utils(SPI_wrapper_driver)

        virtual SPI_wrapper_if SPI_wrapper_driver_vif;
        SPI_wrapper_seq_item SPI_wrapper_driver_seq_item;

        function new (string name, uvm_component parent);
            super.new(name, parent);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                SPI_wrapper_driver_seq_item = SPI_wrapper_seq_item::type_id::create("SPI_wrapper_driver_seq_item");
                seq_item_port.get_next_item(SPI_wrapper_driver_seq_item);
                drive_transaction(SPI_wrapper_driver_seq_item);
                seq_item_port.item_done();
            end

        endtask

        task drive_transaction(SPI_wrapper_seq_item item);
          SPI_wrapper_driver_vif.rst_n = item.rst_n;
          SPI_wrapper_driver_vif.SS_n = item.SS_n;
          SPI_wrapper_driver_vif.MOSI = item.MOSI;
          @(negedge SPI_wrapper_driver_vif.clk);
          `uvm_info(get_type_name(), $sformatf("Driving: %s", item.convert2string()), UVM_DEBUG)
        endtask

    endclass


endpackage
```

wrapper monitor :

```
package SPI_wrapper_monitor_pkg;


    import shared_pkg::*;
    import uvm_pkg::*;
    import SPI_wrapper_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class SPI_wrapper_monitor extends uvm_monitor;
        `uvm_component_utils(SPI_wrapper_monitor)

        SPI_wrapper_seq_item SPI_wrapper_monitor_seq_item;
        virtual SPI_wrapper_if SPI_wrapper_monitor_vif;
        uvm_analysis_port #(SPI_wrapper_seq_item) SPI_wrapper_monitor_aport;

        function new(string name, uvm_component parent);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            SPI_wrapper_monitor_aport = new("SPI_wrapper_monitor_aport", this);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                SPI_wrapper_monitor_seq_item = SPI_wrapper_seq_item::type_id::create("SPI_wrapper_monitor_seq_item");
                @(negedge SPI_wrapper_monitor_vif.clk);

                SPI_wrapper_monitor_seq_item.MOSI     = SPI_wrapper_monitor_vif.MOSI ;
                SPI_wrapper_monitor_seq_item.rst_n    = SPI_wrapper_monitor_vif.rst_n;
                SPI_wrapper_monitor_seq_item.SS_n     = SPI_wrapper_monitor_vif.SS_n ;
                SPI_wrapper_monitor_seq_item.MISO     = SPI_wrapper_monitor_vif.MISO ;
                SPI_wrapper_monitor_seq_item.MISO_GM  = SPI_wrapper_monitor_vif.MISO_GM ;

                SPI_wrapper_monitor_aport.write(SPI_wrapper_monitor_seq_item);
            end
        endtask

    endclass

endpackage
```

wrapper sequencer :

```
package SPI_wrapper_sequencer_pkg;

    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_seq_item_pkg::*;

    class SPI_wrapper_sequencer extends uvm_sequencer #(SPI_wrapper_seq_item);
        `uvm_component_utils(SPI_wrapper_sequencer)

        function new(string name, uvm_component parent);
            super.new(name, parent);
        endfunction
    endclass

endpackage
```

wrapper item :

SPI_wrapper_seq_item.sv > {} SPI_wrapper_seq_item_pkg

```
package SPI_wrapper_seq_item_pkg;

  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import shared_pkg::*;
  class SPI_wrapper_seq_item extends uvm_sequence_item;
      `uvm_object_utils(SPI_wrapper_seq_item)


    rand logic MOSI, rst_n, SS_n;


    //output signals

    logic MISO,MISO_GM;

    //control signals

    operation_t prev_operation = read_data;
    rand operation_t operation;
    rand int cycle_count = 0;
    rand bit[2:0]spi_cmd;

    rand bit MOSI_data[];



    constraint reset_c {
      rst_n dist {0:=1,1:=99};
    }
    constraint MOSI_c {
      MOSI_data.size() == cycle_count-1;
    }

    constraint spi_cmd_cons{
      spi_cmd inside {3'b000, 3'b001, 3'b110, 3'b111};

      if(operation == write_addr){
        spi_cmd == 3'b000;
      }
      else if (operation == write_data){
        spi_cmd == 3'b001;
      }
      else if(operation == read_addr){
        spi_cmd == 3'b110;
      }
      else if(operation == read_data){
        spi_cmd == 3'b111;
      }
```

```
  }
  constraint cycle_count_cons{
    if(operation == read_data){
      cycle_count == 23;
    }
    else{
      cycle_count == 13;
    }
  }

  function void post_randomize();
    prev_operation = operation;
    MOSI_data = new[cycle_count-1];
    for(int i = 0; i < 3; i++)begin
      MOSI_data[i] = spi_cmd[2-i];
    end
    for(int i = 3; i < cycle_count-1; i++)begin
      MOSI_data[i] = $urandom_range(0,1);
    end

  endfunction

  function string convert2string();
      return $sformatf("%s MOSI =0b%0b,\n SS_n =0b%0b,\n rst_n =0b%0b,\n MISO =0b%0b",super.convert2string(),
                       MOSI, SS_n, rst_n,MISO);
  endfunction
  function string convert2string_stimulus();
      return $sformatf("MOSI =0b%0b,\n SS_n =0b%0b,\n rst_n =0b%0b,\n",MOSI, SS_n, rst_n);
  endfunction
endclass

endpackage
```

wrapper seq :

```
package SPI_wrapper_sequence_pkg;

    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_seq_item_pkg::*;
    import shared_pkg::*;

    class SPI_wrapper_reset_sequence extends uvm_sequence #(SPI_wrapper_seq_item);
        `uvm_object_utils(SPI_wrapper_reset_sequence)

        SPI_wrapper_seq_item item;
        function new(string name = "SPI_wrapper_reset_sequence");
            super.new(name);
        endfunction

        task body();
            repeat(5) begin
                item = SPI_wrapper_seq_item::type_id::create("item");
                start_item(item);
                assert(item.randomize() with {rst_n == 0; SS_n == 1;});
                finish_item(item);
            end
        endtask

    endclass

    class SPI_wrapper_write_sequence extends uvm_sequence #(SPI_wrapper_seq_item);
        `uvm_object_utils(SPI_wrapper_write_sequence)

      function new(string name = "SPI_wrapper_write_sequence");
            super.new(name);
        endfunction

        task body();
            SPI_wrapper_seq_item item;
            item = SPI_wrapper_seq_item::type_id::create("item");
            repeat(2000)begin

                assert(item.randomize() with{
                    operation inside{write_addr, write_data};

                });
                send_transaction(item);

            end

        endtask
```

```
    task send_transaction(SPI_wrapper_seq_item trans_item);
        SPI_wrapper_seq_item item;
        int transaction_cycle = trans_item.cycle_count;
        item = SPI_wrapper_seq_item::type_id::create("item");
        start_item(item);
        assert(item.randomize() with{
            SS_n == 0;
            rst_n == 1;
            operation == trans_item.operation;

        });
        finish_item(item);

        for(int j = 0; j < transaction_cycle-1; j++)begin

            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n == 1;
                operation == trans_item.operation;

                MOSI == trans_item.MOSI_data[j];
            });
            finish_item(item);
        end

        start_item(item);
        assert(item.randomize() with{
            SS_n  == 1;
            rst_n == 1;
            operation == trans_item.operation;

        });
        finish_item(item);

    endtask
endclass
```

```
class SPI_wrapper_read_sequence extends uvm_sequence #(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_read_sequence)

    SPI_wrapper_seq_item item;

    function new(string name = "SPI_wrapper_read_sequence");
        super.new(name);
    endfunction

    task body();

        item = SPI_wrapper_seq_item::type_id::create("item");


        repeat(1000)begin


            assert(item.randomize() with{
                operation inside{read_addr, read_data};

                if(prev_operation == read_addr){
                    operation == read_data;
                }
                else if(prev_operation == read_data){
                    operation == read_addr;
                }
                else{
                    operation == read_addr;
                }

            });


            send_transaction(item);

        end

    endtask
```

```
    task send_transaction(SPI_wrapper_seq_item trans_item);
        SPI_wrapper_seq_item item;
        int transaction_cycle = trans_item.cycle_count;
        item = SPI_wrapper_seq_item::type_id::create("item");
        start_item(item);
        assert(item.randomize() with{
            SS_n == 0;
            rst_n ==1;
            operation == trans_item.operation;

        });
        finish_item(item);

        for(int j = 0; j< transaction_cycle-1; j++)begin

            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n == 1;
                operation == trans_item.operation;

                MOSI == trans_item.MOSI_data[j];
            });
            finish_item(item);
        end

        start_item(item);
        assert(item.randomize() with{
            SS_n == 1;
            rst_n ==1;
            operation == trans_item.operation;

        });
        finish_item(item);

    endtask
endclass
```

```
class SPI_wrapper_read_write_sequence extends uvm_sequence #(SPI_wrapper_seq_item);
    `uvm_object_utils(SPI_wrapper_read_write_sequence)

    SPI_wrapper_seq_item item;

    function new(string name = "SPI_slave_read_write_sequence");
        super.new(name);
    endfunction

    task body();

        item = SPI_wrapper_seq_item::type_id::create("item");
        repeat(2000)begin


            assert(item.randomize() with{
                if(prev_operation == write_addr){
                    operation inside {write_addr, write_data};
                }
                else if (prev_operation == write_data){
                    operation dist {write_addr :/40, read_addr :/60, read_data :/0, write_data :/0};

                }
                else if(prev_operation == read_addr){
                    operation == read_data;
                }
                else if(prev_operation == read_data){
                   operation dist {write_addr :/60, read_addr :/40, read_data :/0, write_data :/0};

                }
                else{
                    operation == write_addr;
                }

            });

            send_transaction(item);

        end

    endtask
```

```
        task send_transaction(SPI_wrapper_seq_item trans_item);
            SPI_wrapper_seq_item item;
            int transaction_cycle = trans_item.cycle_count;
            item = SPI_wrapper_seq_item::type_id::create("item");
            start_item(item);
            assert(item.randomize() with{
                SS_n == 0;
                rst_n ==1;
                operation == trans_item.operation;

            });
            finish_item(item);

            for(int j = 0; j< transaction_cycle-1; j++)begin

                start_item(item);
                assert(item.randomize() with{
                    SS_n == 0;
                    rst_n ==1;
                    operation == trans_item.operation;

                    MOSI == trans_item.MOSI_data[j];
                });
                finish_item(item);
            end

            start_item(item);
            assert(item.randomize() with{
                SS_n == 1;
                rst_n ==1;
                operation == trans_item.operation;

            });
            finish_item(item);

        endtask
    endclass


endpackage
```

wrapper config :

```
1   package SPI_wrapper_config_obj_pkg;
2
3       import uvm_pkg::*;
4       `include "uvm_macros.svh"
5
6       class SPI_wrapper_config_obj extends uvm_object;
7           `uvm_object_utils(SPI_wrapper_config_obj)
8           virtual SPI_wrapper_if SPI_wrapper_vif;
9           uvm_active_passive_enum is_active;
0
1           function new(string name = "shift_reg_cfg");
2               super.new(name);
3           endfunction
4
5       endclass
6
7
8   endpackage
```

wrapper coverage :

```
package SPI_wrapper_coverage_pkg;

    import uvm_pkg::*;
    import SPI_wrapper_seq_item_pkg::*;
    `include "uvm_macros.svh"

    class SPI_wrapper_coverage extends uvm_component;
        `uvm_component_utils(SPI_wrapper_coverage)

        uvm_analysis_export #(SPI_wrapper_seq_item) SPI_wrapper_coverage_aexport;
        uvm_tlm_analysis_fifo #(SPI_wrapper_seq_item) cov_fifo;
        SPI_wrapper_seq_item cov_item;

      function new (string name = "SPI_wrapper_coverage",uvm_component parant = null);
        super.new(name,parant);
      endfunction

      function void build_phase (uvm_phase phase);
            super.build_phase(phase);

            SPI_wrapper_coverage_aexport = new("SPI_wrapper_coverage_aexport",this);
            cov_fifo = new("cov_fifo",this);

        endfunction

        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            SPI_wrapper_coverage_aexport.connect(cov_fifo.analysis_export);
        endfunction

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_item);
            end

        endtask

    endclass

endpackage
```

wrapper scoreboard :

```
package SPI_wrapper_scoreboard_pkg;

    import uvm_pkg::*;
    import SPI_wrapper_seq_item_pkg::*;

    `include "uvm_macros.svh"

    class SPI_wrapper_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(SPI_wrapper_scoreboard)

        uvm_analysis_export #(SPI_wrapper_seq_item) SPI_wrapper_scoreboard_aexport;
        uvm_tlm_analysis_fifo #(SPI_wrapper_seq_item) sb_fifo;
        SPI_wrapper_seq_item sb_item;

        int error_count = 0,correct_count = 0;

        function new (string name = "SPI_wrapper_scoreboard",uvm_component parant = null);
            super.new(name,parant);
        endfunction

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);

            SPI_wrapper_scoreboard_aexport = new("SPI_wrapper_scoreboard_aexport",this);
            sb_fifo = new("sb_fifo",this);

        endfunction

        function void connect_phase (uvm_phase phase);
            super.connect_phase(phase);
            SPI_wrapper_scoreboard_aexport.connect(sb_fifo.analysis_export);
        endfunction

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_item);

                if (sb_item.MISO != sb_item.MISO_GM) begin
                    `uvm_error("run_phase" , $sformatf("comprsion failed, transaction recevied by the DUT %s while the ref_MISO = %d",sb_item.convert2string(),
                                                        sb_item.MISO_GM))
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct_count out : %s ",sb_item.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            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_LOW);
            `uvm_info("report_phase", $sformatf("Total failed transactions: %0d",error_count), UVM_LOW);
        endfunction

    endclass

endpackage
```

wrapper do file :

```
vlib work
vlog -f src_files.list +cover -covercells +define+SIM
vsim -voptargs=+acc work.SPI_wrapper_top -classdebug -uvmcontrol=all -cover
add wave /SPI_wrapper_top/SPI_wrapperif/*
add wave -position insertpoint \
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/received_address \
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cs

coverage save SPI_wrapper.ucdb -onexit
run -all
coverage exclude -src RAM.v -line 29 -code s
coverage exclude -src RAM.v -line 29 -code b
coverage exclude -src SPI_slave.sv -line 60 -code b
coverage exclude -src SPI_slave.sv -line 136 -code b
coverage exclude -src SPI_slave.sv -line 136 -code s
coverage exclude -src SPI_slave.sv -line 60 -code s
# vcover report SPI_wrapper.ucdb -details -annotate -all -output coverage_rpt.txt
```

wrapper scr_file :

```
shared_pkg.sv
SPI_slave_if.sv
SPI_slave.sv
SPI_slave_GM.v
SPI_slave_sva.sv
SPI_slave_config.sv
SPI_slave_seq_item.sv
SPI_slave_driver.sv
SPI_slave_sequencer.sv
SPI_slave_monitor.sv
SPI_slave_agent.sv
SPI_slave_cover.sv
SPI_slave_scoreboard.sv
SPI_slave_main_seq.sv
SPI_slave_rst_seq.sv
SPI_slave_env.sv
SPI_slave_test.sv
SPI_slave_top.sv
```

```
19
20   RAM.v
21   RAM_if.sv
22   RAM_golden_model.v
23   RAM_SVA.sv
24   RAM_configer.sv
25   RAM_seq_item.sv
26   RAM_reset_seq.sv
27   RAM_read_only_seq.sv
28   RAM_write_only_seq.sv
29   RAM_write_and_read_seq.sv
30   RAM_seqr.sv
31   RAM_driver.sv
32   RAM_monitor.sv
33   RAM_agent.sv
34   RAM_scoreboard.sv
35   RAM_coverage_collector.sv
36   RAM_environment.sv
37   RAM_test.sv
38   RAM_top.sv
39
```

```
39
40   SPI_wrapper_if.sv
41   SPI_wrapper.sv
42   SPI_wrapper_GM.v
43   SPI_wrapper_sva.sv
44   SPI_wrapper_config_obj.sv
45   SPI_wrapper_seq_item.sv
46   SPI_wrapper_driver.sv
47   SPI_wrapper_sequencer.sv
48   SPI_wrapper_monitor.sv
49   SPI_wrapper_agent.sv
50   SPI_wrapper_coverage.sv
51   SPI_wrapper_scoreboard.sv
52   SPI_wrapper_sequence.sv
53   SPI_wrapper_env.sv
54   SPI_wrapper_test.sv
55   SPI_wrapper_top.sv
```

code coverage :

```
================================================================================
=== Instance: /SPI_wrapper_top/SPI_wrapperif
=== Design Unit: work.SPI_wrapper_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         12        12         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /SPI_wrapper_top/SPI_wrapperif --

                                    Node      1H->0L      0L->1H                          "Coverage"
                                    ----------------------------------------------------------------
                                    MISO           1           1                             100.00
                                  MISO_GM          1           1                             100.00
                                    MOSI           1           1                             100.00
                                    SS_n           1           1                             100.00
                                     clk           1           1                             100.00
                                   rst_n           1           1                             100.00

Total Node Count     =          6
Toggled Node Count   =          6
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (12 of 12 bins)
```

```
================================================================================
=== Instance: /SPI_wrapper_top/SPI_wrapper_instance/RAM_instance
=== Design Unit: work.RAM
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         8         8         0   100.00%
```

```
Branch totals: 4 hits of 4 branches = 100.00%


Expression Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Expressions                      3         3         0   100.00%

================================Expression Details================================
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      10        10         0   100.00%

--------------------------------Statement Details--------------------------------
```

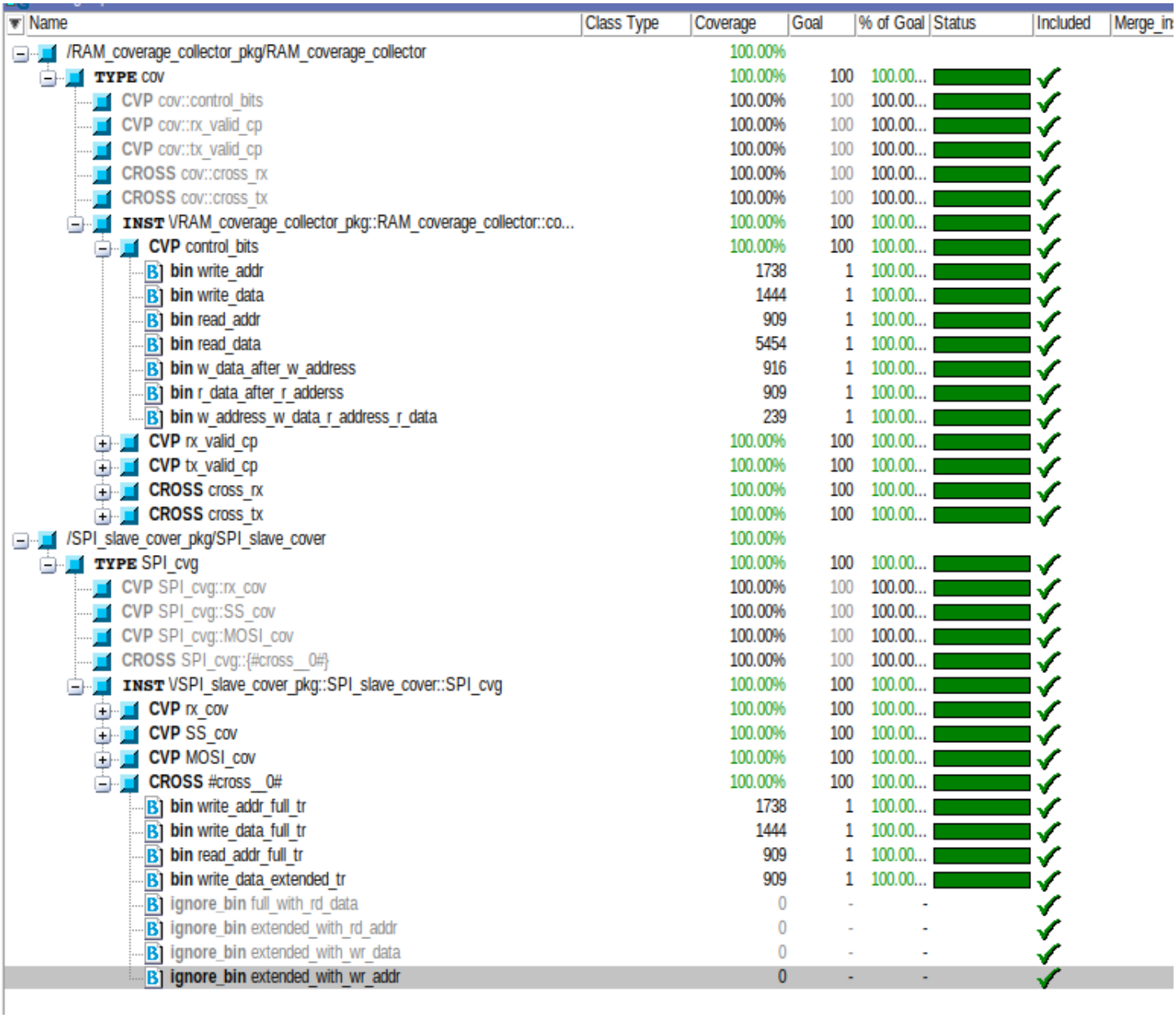
```
==============================Toggle Details================================

Toggle Coverage for instance /SPI_wrapper_top/SPI_wrapper_instance/RAM_instance --

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

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

Toggle Coverage      =     100.00% (76 of 76 bins)
```

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

Toggle Coverage for instance /SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance --

                                    Node      1H->0L      0L->1H                          "Coverage"
                                    ----------------------------------------------------------------
                               counter[3-0]          1           1                             100.00
                                         cs              ENUM type       Value       Count
                                                             IDLE        2001      100.00
                                                          CHK_CMD        2002      100.00
                                                            WRITE        2000      100.00
                                                         READ_ADD           1      100.00
                                                        READ_DATA           1      100.00
                                         ns              ENUM type       Value       Count
                                                             IDLE        2001      100.00
                                                          CHK_CMD        2002      100.00
                                                            WRITE        2001      100.00
                                                         READ_ADD        1003      100.00
                                                        READ_DATA           1      100.00
                                received_address          1           1                             100.00

Total Node Count     =         15
Toggled Node Count   =         15
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (20 of 20 bins)
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      37        37         0   100.00%

==============================Statement Details================================

Statement Coverage for instance /SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance --
```

```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        37        37         0   100.00%

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

functional coverage :



Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_in
/RAM_coverage_collector_pkg/RAM_coverage_collector		100.00%					
TYPE cov		100.00%	100	100.00...			
CVP cov::control_bits		100.00%	100	100.00...			
CVP cov::rx_valid_cp		100.00%	100	100.00...			
CVP cov::tx_valid_cp		100.00%	100	100.00...			
CROSS cov::cross_rx		100.00%	100	100.00...			
CROSS cov::cross_tx		100.00%	100	100.00...			
INST \RAM_coverage_collector_pkg::RAM_coverage_collector::co...		100.00%	100	100.00...			
CVP control_bits		100.00%	100	100.00...			
bin write_addr		1738	1	100.00...			
bin write_data		1444	1	100.00...			
bin read_addr		909	1	100.00...			
bin read_data		5454	1	100.00...			
bin w_data_after_w_address		916	1	100.00...			
bin r_data_after_r_adderss		909	1	100.00...			
bin w_address_w_data_r_address_r_data		239	1	100.00...			
CVP rx_valid_cp		100.00%	100	100.00...			
CVP tx_valid_cp		100.00%	100	100.00...			
CROSS cross_rx		100.00%	100	100.00...			
CROSS cross_tx		100.00%	100	100.00...			
/SPI_slave_cover_pkg/SPI_slave_cover		100.00%					
TYPE SPI_cvg		100.00%	100	100.00...			
CVP SPI_cvg::rx_cov		100.00%	100	100.00...			
CVP SPI_cvg::SS_cov		100.00%	100	100.00...			
CVP SPI_cvg::MOSI_cov		100.00%	100	100.00...			
CROSS SPI_cvg::{#cross__0#}		100.00%	100	100.00...			
INST \SPI_slave_cover_pkg::SPI_slave_cover::SPI_cvg		100.00%	100	100.00...			
CVP rx_cov		100.00%	100	100.00...			
CVP SS_cov		100.00%	100	100.00...			
CVP MOSI_cov		100.00%	100	100.00...			
CROSS #cross__0#		100.00%	100	100.00...			
bin write_addr_full_tr		1738	1	100.00...			
bin write_data_full_tr		1444	1	100.00...			
bin read_addr_full_tr		909	1	100.00...			
bin write_data_extended_tr		909	1	100.00...			
ignore_bin full_with_rd_data		0	-	-			
ignore_bin extended_with_rd_addr		0	-	-			
ignore_bin extended_with_wr_data		0	-	-			
ignore_bin extended_with_wr_addr		0	-	-			

assertion coverage :

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/reset_cover	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/tx_deasserted_cover	SVA	✓	Off	8180	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/tx_asserted_cover	SVA	✓	Off	9090	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/write_cover	SVA	✓	Off	3476	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/read_cover	SVA	✓	Off	1818	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__READ_DATA_to_IDLE	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__READ_ADD_to_IDLE	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__WRITE_to_IDLE	SVA	✓	Off	3182	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__CHK_CMD_to_read_data	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__CHK_CMD_read_address	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__CHK_CMD_to_write	SVA	✓	Off	3182	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/cover__IDLE_to_CHK_CMD	SVA	✓	Off	5000	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__valid_command_rd_data	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__valid_command_rd_addr	SVA	✓	Off	909	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__valid_command_wr_data	SVA	✓	Off	1444	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__valid_command_wr_addr	SVA	✓	Off	1738	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__rx_data_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__rx_valid_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/cover__MISO_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/cover__MISO_satble	SVA	✓	Off	58608	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/cover__rx_data_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/cover__rx_valid_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/cover__MISO_reset	SVA	✓	Off	5	1	Unli...	1	100%		✓	0	0	0 ns	0

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active C
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	
/SPI_wrapper_sequence_pkg::SPI_wrapper_reset_sequence::body/#ublk#252082231#17/immed__20	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_write_sequence::body/#ublk#252082231#37/immed__39	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_write_sequence::send_transaction/immed__54	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_write_sequence::send_transaction/immed__76	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_write_sequence::send_transaction/#anonblk#252082231#62#4#/#ublk#252082231#62/immed__65	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_sequence::body/#ublk#252082231#101/immed__104	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_sequence::send_transaction/immed__131	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_sequence::send_transaction/immed__153	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_sequence::send_transaction/#anonblk#252082231#139#4#/#ublk#252082231#139/immed__142...	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_write_sequence::body/#ublk#252082231#177/immed__180	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_write_sequence::send_transaction/immed__212	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_write_sequence::send_transaction/immed__234	Immediate	SVA	on	0	1	
/SPI_wrapper_sequence_pkg::SPI_wrapper_read_write_sequence::send_transaction/#anonblk#252082231#220#4#/#ublk#252082231#220/immed...	Immediate	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/tx_deasserted	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/tx_asserted	Concurrent	SVA	on	0	0	
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/write_assert	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/RAM_instance/ram_sva/read_assert	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__IDLE_to_CHK_CMD	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__CHK_CMD_to_write	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__CHK_CMD_read_address	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__CHK_CMD_to_read_data	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__WRITE_to_IDLE	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__READ_ADD_to_IDLE	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/assert__READ_DATA_to_IDLE	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__MISO_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__rx_valid_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__rx_data_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__valid_command_wr_addr	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__valid_command_wr_data	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__valid_command_rd_addr	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/SLAVE_instance/slave_sva/assert__valid_command_rd_data	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/assert__MISO_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/assert__rx_valid_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/assert__rx_data_reset	Concurrent	SVA	on	0	1	
/SPI_wrapper_top/SPI_wrapper_instance/wrapper_sva/assert__MISO_satble	Concurrent	SVA	on	0	1	

assertions :

Feature	Assertion
Reset behavior: Whenever reset is asserted, MISO, rx_valid, and rx_data must all be low.	`@(posedge clk) (!rst_n
MISO stability: When a valid transaction occurs that is **not** a read-data operation (rx_data[9:8] != 2'b11), the MISO output must remain stable.	`@(posedge clk) (rx_data[9:8] != 2'b11 && rx_valid)