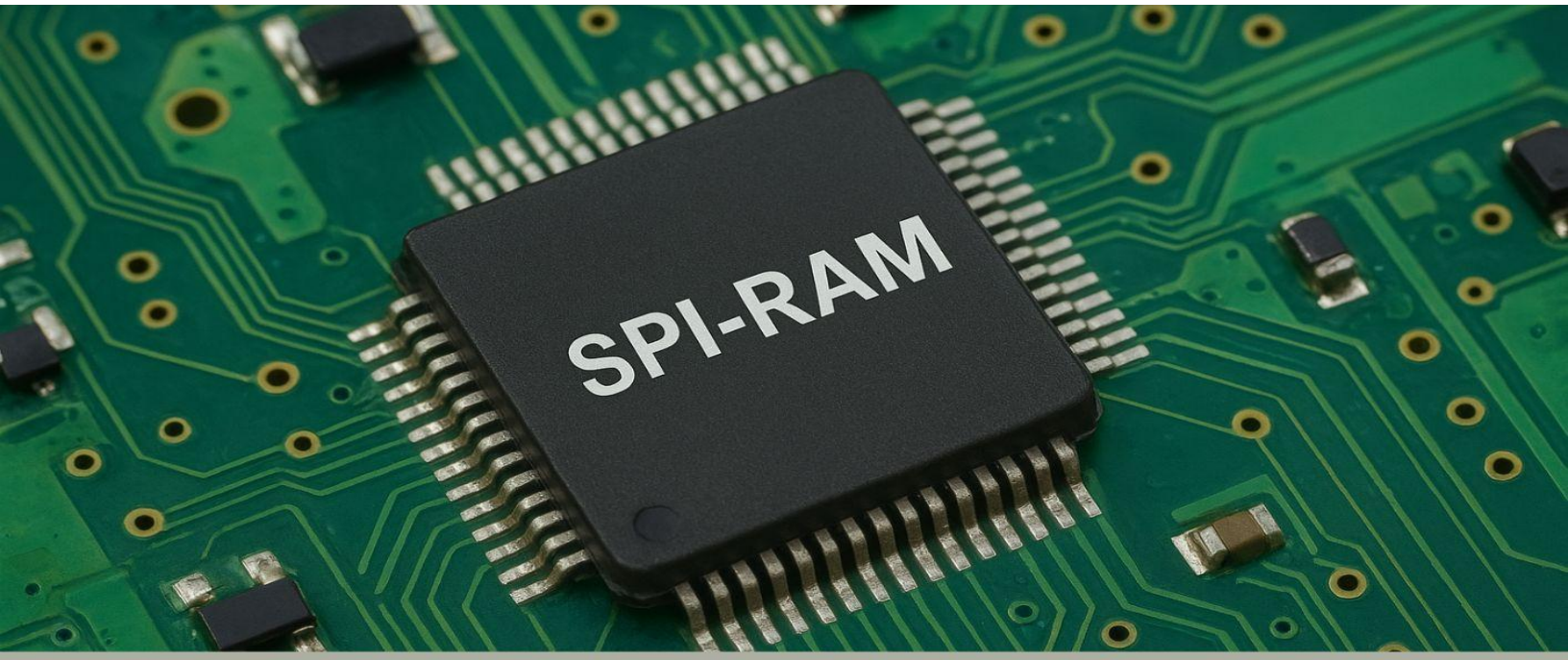
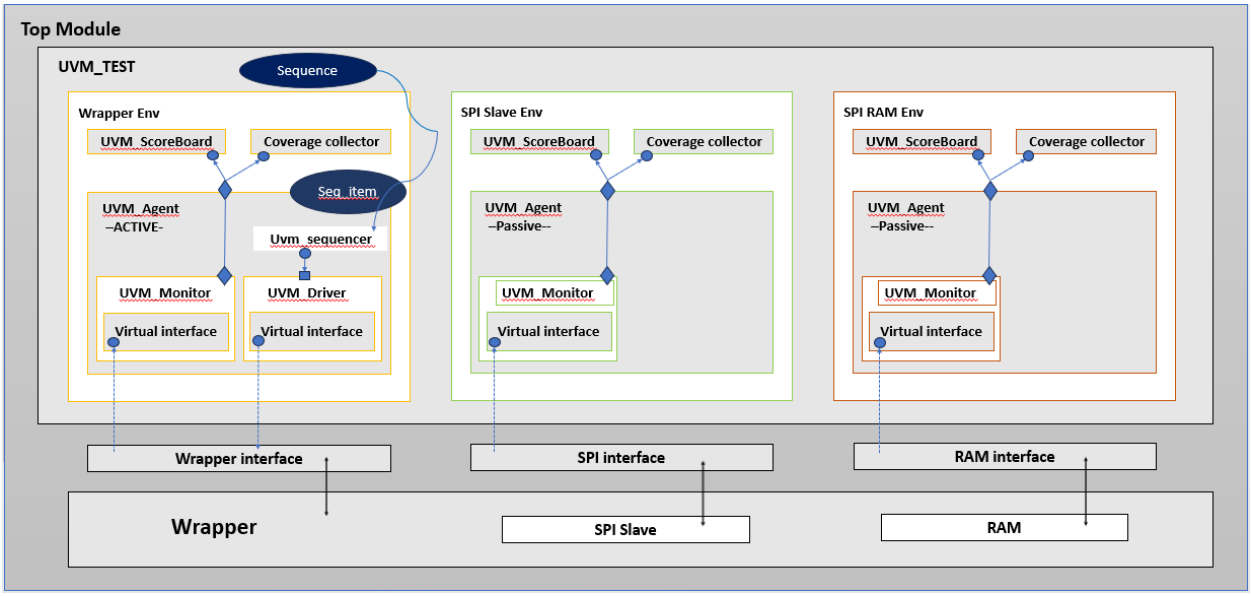


UVM Project

SPI Slave with Single Port RAM

Prepared by:
Nouran Hesham
Omar Waleed Salah
Sama Gamal Ahmed Salama

UVM Diagram:



SPI SLAVE CODES:

Design after debugging + Assertions:

```
module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);

localparam IDLE      = 3'b000;
localparam WRITE     = 3'b001;
localparam CHK_CMD   = 3'b010;
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;

input MOSI, clk, rst_n, SS_n, tx_valid;
input  [7:0] tx_data;
output reg [9:0] rx_data;
output reg  rx_valid, MISO;

reg [3:0] counter;
reg  received_address;

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
            if (SS_n)
                ns = IDLE;
            else begin
                if (~MOSI)
                    ns = WRITE;
                else begin //mosi =1 , ssn =0
                    if (received_address)
                        ns = READ_DATA; // FIXED changed from read addr to data
                    else
                        ns = READ_ADD;// FIXED changed from read data to addr
                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;
        received_address <= 0;
        MISO <= 0;
        counter <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end
            READ_ADD : begin
                if (counter > 0 ) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    received_address <= 1;
                end
            end
            READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
```

```
                    if (counter > 0 ) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 9;  /// BUG FIXED
                    end
                end

            end
            default : begin
                counter <= 0;
                MISO <= 1'b0;
            end
        endcase
        end
    end

//Assertions

property assert_reset;
    @(posedge clk) (!rst_n) |=> (!MISO && rx_data=='0 &&rx_valid=='0);
endproperty
assert property(assert_reset);
cover property (assert_reset);


property write_add_seq;
  @(posedge clk) disable iff(!rst_n)
     ($fell(SS_n) ##1(!MOSI)[*3] ) |-> ##10 ( rx_valid ##[0:$] SS_n);
endproperty

assert property(write_add_seq);
cover property(write_add_seq);
```

```
property write_data_seq;
  @(posedge clk) disable iff(!rst_n)
     ( $fell(SS_n) ##1(!MOSI)[*2] ##1 (MOSI) ) |-> ##10 ( rx_valid ##[0:$] SS_n);
endproperty

assert property(write_data_seq);
cover property(write_data_seq);

property read_add_seq;
  @(posedge clk) disable iff(!rst_n)
     ( $fell(SS_n) ##1(MOSI)[*2] ##1 (!MOSI) ) |-> ##10 (rx_valid ##[0:$] SS_n);
endproperty

assert property(read_add_seq);
cover property(read_add_seq);

property read_data_seq;
  @(posedge clk) disable iff(!rst_n)
     ($fell(SS_n) ##1(MOSI)[*3]  ) |-> ##10 (rx_valid ##[0:$] SS_n);
endproperty

assert property(read_data_seq);
cover property(read_data_seq);




property idle_to_check_cmd;
  @(posedge clk) disable iff(!rst_n)
    (cs == IDLE && !SS_n) |-> (ns==CHK_CMD);
endproperty
assert property (idle_to_check_cmd);
cover property (idle_to_check_cmd);

// 2) CHK_CMD >> WRITE or READ_ADDR or READ_DATA
property check_cmd_to_write;
  @(posedge clk) disable iff(!rst_n)
    (cs == CHK_CMD && !SS_n && !MOSI) |-> (ns == WRITE );
endproperty
assert property (check_cmd_to_write);
cover property (check_cmd_to_write);

property check_cmd_to_read_addr;
  @(posedge clk) disable iff(!rst_n)
    (cs == CHK_CMD && !SS_n && MOSI && !received_address) |-> (ns == READ_ADD);
```

```
endproperty
assert property (check_cmd_to_read_addr);
cover property (check_cmd_to_read_addr);

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

// 3) WRITE >> IDLE
property write_to_idle;
  @(posedge clk) disable iff(!rst_n)
    (cs == WRITE && SS_n)|-> (ns == IDLE);
endproperty
assert property (write_to_idle);
cover property (write_to_idle);

// 4) READ_ADD >> IDLE
property read_add_to_idle;
  @(posedge clk) disable iff(!rst_n)
    (cs == READ_ADD && SS_n) |-> (ns == IDLE);
endproperty
assert property (read_add_to_idle);
cover property (read_add_to_idle);

// 5) READ_DATA >> IDLE
property read_data_to_idle;
  @(posedge clk) disable iff(!rst_n)
    (cs == READ_DATA && SS_n) |->  (ns == IDLE);
endproperty
assert property (read_data_to_idle);
cover property (read_data_to_idle);


endmodule
```

Bugs Report:

- The condition should check for not ‘received_address’ to remain in the READ_ADDR , else should Transition to the READ_DATA.

- The design should should reload the counter to 9 as there will be a waiting cycle before the tx_valid comes and the Data gets out starting from address 7 in the tx_data.

Before Debugging:

```
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 8;
                    end
                end
```

```
            else begin
                if (received_address)
                    ns = READ_ADD;
                else
                    ns = READ_DATA;
            end
        end
```

After Debugging:

```
        else begin //mosi =1 , ssn =0
            if (received_address)
                ns = READ_DATA; // FIXED changed from read addr to data
            else
                ns = READ_ADD;// FIXED changed from read data to addr
        end
```

```
                else begin
                    rx_valid <= 1;
                    counter <= 9;  /// BUG FIXED
                end
```

SPI_Slave Golden:

```
module spi_golden (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
localparam IDLE      = 3'b000;
localparam WRITE     = 3'b001;
localparam CHK_CMD   = 3'b010;
localparam READ_ADD  = 3'b011;
localparam READ_DATA = 3'b100;

input MOSI,SS_n,clk,rst_n,tx_valid ;
input [7:0] tx_data;
output reg [9:0] rx_data ;
output reg MISO,rx_valid;

reg read_addr_received ; // law b 1 y3ni yroh ygeb l data law b zero yero7 ll
address
reg [3:0] counter; //tracks how many bits we’ve received

reg [2:0] current_state,next_state ;
always @(posedge clk) begin
    if (~rst_n)
        current_state <= IDLE ;
    else
        current_state <= next_state ;
end

//Next state logic always block
always @(*) begin
    case (current_state)
        IDLE: begin
            if (SS_n == 0 ) next_state = CHK_CMD;
            else next_state = IDLE;
        end
        CHK_CMD: begin
            if (SS_n == 0 && MOSI == 1) begin
                if (read_addr_received)
                    next_state = READ_DATA;
                else
                    next_state = READ_ADD;
            end else if (SS_n == 0 && MOSI == 0) begin
                next_state = WRITE;
            end
            else if (SS_n) begin
```

```
                next_state = IDLE ;
            end
        end
        WRITE: begin
            if (SS_n == 1)
                next_state = IDLE;
            else
                next_state = WRITE;
        end

        READ_ADD: begin
            if (SS_n == 1) begin
                next_state = IDLE;
            end
            else
                next_state = READ_ADD;
        end
        READ_DATA: begin
            if (SS_n == 1)
                next_state = IDLE;
            else
                next_state = READ_DATA;
        end

        default: next_state = IDLE;
    endcase
end

//Output logic always block
always @(posedge clk) begin
    if (!rst_n) begin
        counter <= 0;
        rx_data <= 0;
        read_addr_received <= 0;
        MISO <= 1'b0;
        rx_valid <= 0;
    end
    else begin
        case (current_state)

IDLE:  begin
            rx_valid <= 0;
        end

CHK_CMD: counter <= 10;
```

```
WRITE: begin
    if (counter >0) begin
            rx_data[counter-1] <= MOSI;
            counter <= counter - 1;
                end
            else rx_valid <= 1;
            end

READ_ADD: begin
        if (counter >0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
        else begin
                    rx_valid <= 1;
                    read_addr_received <= 1;
                end
            end

READ_DATA: begin
    if (tx_valid) begin
        rx_valid<=0;
        if (counter >0) begin
            MISO <= tx_data[counter-1];
            counter <= counter - 1;
        end
        else read_addr_received<=0;
    end
    else begin
        if (counter>0) begin
            rx_data[counter-1] <= MOSI;
            counter <= counter - 1;
        end
        else begin
            rx_valid <= 1;
            counter<=9;
        end
    end
end

default: begin
            counter <= 0;
            MISO <= 1'b0;
            end
```

```
        endcase
    end
end
endmodule
```

SPI_Slave Interface:

```
interface SPI_interface ( clk);
input clk;
  logic rst_n;
  logic MOSI;
  logic MISO;
  logic SS_n;
  logic tx_valid;
  logic [7:0] tx_data;
  logic [9:0] rx_data;
  logic rx_valid;
  logic  [9:0] rx_data_golden;
  logic rx_valid_golden;
  logic MISO_golden;
endinterface
```

SPI _Slave shared pkg:

```
package SPI_shared_pkg;
    bit[5:0] count;
    logic [10:0] arr_of_data;
    bit is_read;
    bit have_address_to_read;
    int limit;
endpackage
```

SPI_Slave Config:

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

class SPI_config extends uvm_object;
`uvm_object_utils(SPI_config)
```

```
virtual SPI_interface SPI_vif;
uvm_active_passive_enum is_active;

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

endclass
endpackage
```

SPI_Slave environment:

```
package SPI_env_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import SPI_agent_pkg::*;
import SPI_coverage_pkg::*;
import SPI_scoreboard_pkg::*;

class SPI_env extends uvm_env;
`uvm_component_utils(SPI_env);
SPI_agent agt;
SPI_scoreboard sb;
SPI_coverage cov;

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

function void build_phase(uvm_phase phase);
super.build_phase(phase);
agt = SPI_agent::type_id::create("agt",this);
sb = SPI_scoreboard::type_id::create("sb",this);
cov = SPI_coverage::type_id::create("cov",this);
endfunction

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

endclass
```

```
endpackage
```

SPI_Slave Agent:

```
package SPI_agent_pkg;
import uvm_pkg::*;
import SPI_sequencer_pkg::*;
import SPI_driver_pkg::*;
import SPI_monitor_pkg::*;
import SPI_config_pkg::*;
import SPI_seq_item_pkg::*;
`include "uvm_macros.svh"

class SPI_agent extends uvm_agent;
`uvm_component_utils(SPI_agent)
SPI_sequencer sqr;
SPI_driver driv;
SPI_monitor mon;
SPI_config SPI_cfg;
uvm_analysis_port #(SPI_seq_item) agent_analport;

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

function void build_phase(uvm_phase phase);
super.build_phase(phase);
if(!uvm_config_db#(SPI_config):: get(this,"","Config_key",SPI_cfg))begin
`uvm_fatal("build_phase","unable to get configuration object");
end
mon = SPI_monitor ::type_id::create("mon",this);
agent_analport = new("agent_analport",this);

if(SPI_cfg.is_active == UVM_ACTIVE) begin
    sqr = SPI_sequencer::type_id::create("sqr",this);
    driv = SPI_driver::type_id::create("driv",this);
end
endfunction

function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
mon.SPI_vif = SPI_cfg.SPI_vif;
mon.mon_ap.connect(agent_analport);
```

```
if(SPI_cfg.is_active == UVM_ACTIVE) begin
        driv.SPI_vif = SPI_cfg.SPI_vif;
        driv.seq_item_port.connect(sqr.seq_item_export);
    end

endfunction
endclass

endpackage
```

SPI_Slave Coverage:

```
package SPI_coverage_pkg;
 import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_seq_item_pkg::*;
import SPI_shared_pkg::*;
class SPI_coverage extends uvm_component;
`uvm_component_utils(SPI_coverage)
uvm_analysis_export #(SPI_seq_item) cov_export;
uvm_tlm_analysis_fifo #(SPI_seq_item) cov_fifo;
SPI_seq_item seq_item_cov;

covergroup cov_cg ;

rx_data_cp: coverpoint seq_item_cov.rx_data[9:8]  iff(seq_item_cov.rst_n){
  bins b00 = {2'b00};
  bins b01 = {2'b01};
  bins b10 = {2'b10};
  bins b11 = {2'b11};
  bins trans_00_01 = (2'b00 => 2'b01);//write addreess to write data
  bins trans_10_11 = (2'b10 => 2'b11); //read address to write data
  bins trans_00_00 = (2'b00 => 2'b00); // hold
  bins trans_01_01 = (2'b01 => 2'b01); // hold
  bins trans_10_10 = (2'b10 => 2'b10); // hold
  bins trans_11_11 = (2'b11 => 2'b11); // hold
 }

 SS_n_cp: coverpoint seq_item_cov.SS_n  iff(seq_item_cov.rst_n){
    bins normal_transaction = (1 => 0[*13] => 1);
    bins extended_transaction = (1 => 0[*23] => 1);
 }

MOSI_cp: coverpoint seq_item_cov.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);
}

SSn_MOSI_cross: cross SS_n_cp, MOSI_cp iff(seq_item_cov.rst_n){
     ignore_bins illegal_cross = binsof(SS_n_cp.normal_transaction) &&
binsof(MOSI_cp.read_data)
                    || binsof(SS_n_cp.extended_transaction) &&
binsof(MOSI_cp.write_addr)
                    || binsof(SS_n_cp.normal_transaction) &&
binsof(MOSI_cp.write_addr)
                    || binsof(SS_n_cp.extended_transaction) &&
binsof(MOSI_cp.read_data)
                    || binsof(SS_n_cp.extended_transaction) &&
binsof(MOSI_cp.write_data)
                    || binsof(SS_n_cp.extended_transaction) &&
binsof(MOSI_cp.read_addr);
}
endgroup

function new(string name = "SPI_coverage" , uvm_component parent = null);
    super.new(name, parent);
            seq_item_cov = SPI_seq_item::type_id::create("seq_item_cov");
            cov_cg = new();
    endfunction

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

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

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever begin
    cov_fifo.get(seq_item_cov);
    cov_cg.sample();
end
```

```
endtask

endclass
endpackage
```

SPI_Slave driver:

```
package SPI_driver_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_seq_item_pkg::*;
import SPI_shared_pkg::*;

class SPI_driver extends uvm_driver #(SPI_seq_item);
`uvm_component_utils(SPI_driver)
virtual SPI_interface SPI_vif;
SPI_seq_item stim_seq_item;

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

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

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

endtask
endclass
endpackage
```

SPI_Slave monitor:

```
package SPI_monitor_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import SPI_seq_item_pkg::*;
import SPI_shared_pkg::*;

class SPI_monitor extends uvm_monitor;
    `uvm_component_utils(SPI_monitor)

    virtual SPI_interface SPI_vif;
    SPI_seq_item rsp_seq_item;
    uvm_analysis_port #(SPI_seq_item) mon_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        mon_ap = new("mon_ap", this);
    endfunction
task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
        rsp_seq_item = SPI_seq_item::type_id::create("rsp_seq_item");
        @(negedge SPI_vif.clk);

        // DUT inputs
        rsp_seq_item.rst_n    = SPI_vif.rst_n;
        rsp_seq_item.SS_n     = SPI_vif.SS_n;
        rsp_seq_item.MOSI     = SPI_vif.MOSI;
        rsp_seq_item.tx_data  = SPI_vif.tx_data;
        rsp_seq_item.tx_valid = SPI_vif.tx_valid;

        // DUT outputs
        rsp_seq_item.MISO     = SPI_vif.MISO;
        rsp_seq_item.rx_data  = SPI_vif.rx_data;
        rsp_seq_item.rx_valid = SPI_vif.rx_valid;
```

```
        // Golden model outputs
        rsp_seq_item.MISO_golden     = SPI_vif.MISO_golden;
        rsp_seq_item.rx_data_golden  = SPI_vif.rx_data_golden;
        rsp_seq_item.rx_valid_golden = SPI_vif.rx_valid_golden;

        // send to scoreboard and coverage
        mon_ap.write(rsp_seq_item);
`uvm_info("MONITOR", rsp_seq_item.convert2string(), UVM_HIGH)
    end
endtask

endclass

endpackage
```

SPI_Slave seq item:

```
      package SPI_seq_item_pkg;

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

class SPI_seq_item extends uvm_sequence_item;
`uvm_object_utils(SPI_seq_item)

bit clk ;
rand logic MOSI , SS_n, rst_n, tx_valid;
rand logic [7:0] tx_data;
rand logic [10:0] MOSI_data;  // 11-bit array

// DUT outputs
logic [9:0] rx_data;
logic MISO, rx_valid;

// Golden model outputs
logic [9:0] rx_data_golden;
logic MISO_golden, rx_valid_golden;

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

function string convert2string();
```

```
    return $sformatf("rst_n=%0b, SS_n=%0b, MOSI_data=0x%0h, tx_data=0x%0h,
tx_valid=%0b | DUT: MISO=%0b rx_data=0x%0h rx_valid=%0b | GOLD: MISO=%0b
rx_data=0x%0h rx_valid=%0b",
                      rst_n, SS_n, MOSI_data, tx_data, tx_valid,
                      MISO, rx_data, rx_valid,
                      MISO_golden, rx_data_golden, rx_valid_golden);
endfunction

function string convert2string_stimulus();
    return $sformatf("rst_n=%0b, SS_n=%0b, MOSI_data=0x%0h, tx_data=0x%0h,
tx_valid=%0b",
                      rst_n, SS_n, MOSI_data,  tx_data, tx_valid);
endfunction

constraint reset_constraint {
    rst_n dist { 0 := 5, 1 := 95 };
}

constraint valid_MOSI_command {
            MOSI_data[10:8] inside {3'b000, 3'b001, 3'b110, 3'b111};
        }

constraint ready_to_read {
            if(count>=15) tx_valid ==1;
            else tx_valid == 0;
        }

function void post_randomize();
       if(count == 0) arr_of_data = MOSI_data;
     is_read = (arr_of_data[10:8] == 3'b111)? 1:0;
            limit = (is_read)? 23:13;

            SS_n = (count == limit)? 1:0;

            if(arr_of_data[10:8] == 3'b110) have_address_to_read = 1'b1;
            if (is_read || (!rst_n)) have_address_to_read = 1'b0;

            //
            if((count > 0) && (count < 12)) begin
                MOSI = arr_of_data [11-count];
            end

            //count
            if (!rst_n) begin
                count = 0;
```

```
            end
            else begin
                if (count == limit) count = 0;
                else count++;
            end

endfunction
endclass
endpackage
```

SPI_Slave sequence:

```
package SPI_sequence_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import SPI_seq_item_pkg::*;
import SPI_sequencer_pkg::*;

class SPI_reset_sequence extends uvm_sequence #(SPI_seq_item);
`uvm_object_utils(SPI_reset_sequence);
SPI_seq_item seq_item ;

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

task body();
seq_item =SPI_seq_item::type_id::create("seq_item");
start_item(seq_item);
seq_item.rst_n=0;
//inputs
seq_item.MOSI =0;
seq_item.SS_n =0;
seq_item.tx_valid=0;
seq_item.tx_data=0;
//outputs from DUT
seq_item.rx_valid=0;
seq_item.rx_data=0;
seq_item.MISO=0;
//outputs from Golden model
seq_item.rx_data_golden=0;
seq_item.rx_valid_golden=0;
seq_item.MISO_golden=0;
finish_item(seq_item);
```

```
endtask
endclass

class SPI_main_sequence extends uvm_sequence #(SPI_seq_item);
`uvm_object_utils(SPI_main_sequence);
SPI_seq_item seq_item ;

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

task body();
repeat(5000) begin
    seq_item =SPI_seq_item::type_id::create("seq_item");
    start_item(seq_item);
    assert(seq_item.randomize());
    finish_item(seq_item);
end
endtask

endclass

endpackage
```

SPI_Slave Sequencer:

```
package SPI_sequencer_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import SPI_seq_item_pkg::*;

class SPI_sequencer extends uvm_sequencer #(SPI_seq_item);
`uvm_component_utils(SPI_sequencer)

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

endclass
endpackage
```

SPI_Slave ScoreBoard:

```
package SPI_scoreboard_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_seq_item_pkg::*;

class SPI_scoreboard extends uvm_scoreboard;
`uvm_component_utils(SPI_scoreboard)
uvm_analysis_export #(SPI_seq_item) sb_export;
uvm_tlm_analysis_fifo #(SPI_seq_item) sb_fifo;
SPI_seq_item seq_item_sb;

int error_count = 0;
int correct_count = 0;

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

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

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

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
        sb_fifo.get(seq_item_sb);
        check_data(seq_item_sb);
    end
endtask

task check_data(SPI_seq_item item);
if (item.rx_data !== item.rx_data_golden) begin
error_count++;
    `uvm_error("SCOREBOARD", $sformatf("rx_data Mismatch - Expected: 0x%0h, Got:
0x%0h | %s",
                    item.rx_data_golden, item.rx_data, item.convert2string()))
```

```
    end
else correct_count++;

if (item.rx_valid !== item.rx_valid_golden) begin
error_count++;
    `uvm_error("SCOREBOARD", $sformatf("rx_valid Mismatch - Expected: %0b, Got:
%0b | %s",
                    item.rx_valid_golden, item.rx_valid, item.convert2string()))
    end
else correct_count++;

if (item.MISO !== item.MISO_golden) begin
error_count++;
    `uvm_error("SCOREBOARD", $sformatf("MISO Mismatch - Expected: %0b, Got: %0b |
%s",
                    item.MISO_golden, item.MISO, item.convert2string()))
    end
else correct_count++;
endtask

function void report_phase(uvm_phase phase);
super.report_phase(phase);
    `uvm_info("SCOREBOARD", "================== SCOREBOARD REPORT
==================", UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Total Checked: %0d", correct_count +
error_count), UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Passed:        %0d", correct_count),
UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Failed:        %0d", error_count),
UVM_NONE)
    `uvm_info("SCOREBOARD",
"========================================================", UVM_NONE)
endfunction

endclass
endpackage
```

SPI_Slave test:

```
package SPI_test_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import SPI_env_pkg::*;
import SPI_config_pkg::*;
```

```
import  SPI_sequence_pkg::*;

class SPI_test extends uvm_test;
`uvm_component_utils(SPI_test)
SPI_env env;
SPI_config SPI_config_obj;
SPI_main_sequence main_seq;
SPI_reset_sequence reset_seq;
virtual SPI_interface SPI_vif ;

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

function void build_phase(uvm_phase phase);
super.build_phase(phase);
env= SPI_env::type_id::create("env",this);
SPI_config_obj = SPI_config::type_id::create("SPI_config_obj");
main_seq = SPI_main_sequence::type_id::create("main_seq");
reset_seq = SPI_reset_sequence::type_id::create("reset_seq");
if(!uvm_config_db#(virtual SPI_interface):: get(this,"","Config_key",
SPI_config_obj.SPI_vif ))begin
`uvm_fatal("build_phase","unable to get virtual interface")
end
uvm_config_db#(SPI_config)::set(this, "*", "Config_key", SPI_config_obj);
SPI_config_obj.is_active = UVM_ACTIVE;
endfunction

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

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

`uvm_info("run_phase","stimulus generation started",UVM_LOW)
main_seq.start(env.agt.sqr);
`uvm_info("run_phase","stimulus generation ended",UVM_LOW)
#5;
phase.drop_objection(this);
endtask


endclass
```

```
endpackage
```

SPI_Slave Top:

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

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

SPI_interface SPI_if(clk);

SLAVE DUT (SPI_if.MOSI,SPI_if.MISO,SPI_if.SS_n,SPI_if.clk,SPI_if.rst_n,
          SPI_if.rx_data,SPI_if.rx_valid,SPI_if.tx_data,SPI_if.tx_valid
);

spi_golden golden_ref
(SPI_if.MOSI,SPI_if.MISO_golden,SPI_if.SS_n,SPI_if.clk,SPI_if.rst_n,
        SPI_if.rx_data_golden,SPI_if.rx_valid_golden,SPI_if.tx_data,SPI_if.tx_val
id);

initial begin
    uvm_config_db#(virtual SPI_interface)::set(null, "uvm_test_top",
"Config_key", SPI_if);
    run_test("SPI_test");
end

endmodule
```

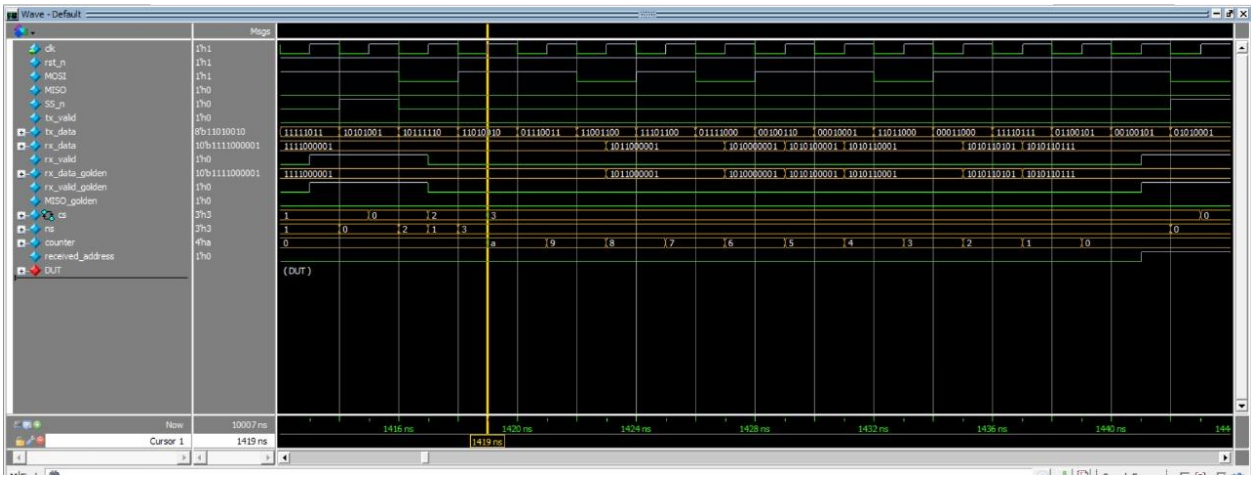
Run.do File:

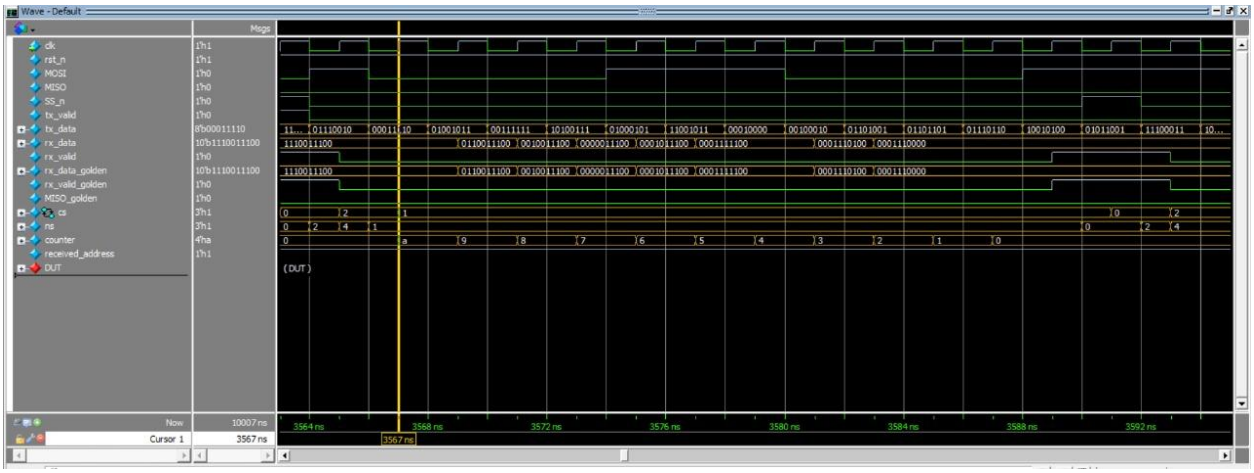
run.do | SPI_coverage_report.txt

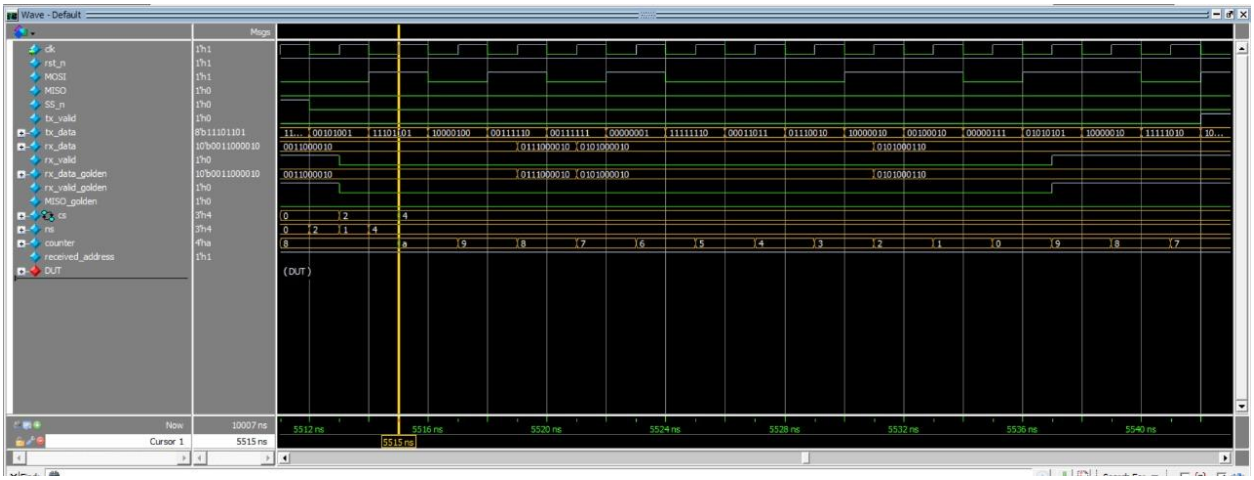
run.do

```
vlib work
vlog -work work SPI_slave.sv +define+SIM
vlog -work work SPI_golden.v
vlog -work work SPI_interface.sv
vlog -work work SPI_shared_pkg.sv
vlog -work work SPI_seq_item.sv
vlog -work work SPI_sequence.sv
vlog -work work SPI_config_object.sv
vlog -work work SPI_sequencer.sv
vlog -work work SPI_driver.sv
vlog -work work SPI_monitor.sv
vlog -work work SPI_agent.sv
vlog -work work SPI_scoreboard.sv
vlog -work work SPI_coverage.sv
vlog -work work SPI_env.sv
vlog -work work SPI_test.sv
vlog -work work SPI_top.sv

vsim -voptargs=+acc work.SPI_top -classdebug -uvmcontrol=all -cover -sv_seed random

add wave -position insertpoint sim:/SPI_top/SPI_if/*
add wave -position insertpoint sim:/SPI_top/DUT/*

coverage save SPI_coverage.ucdb -onexit
run -all

vcover report SPI_coverage.ucdb -details -annotate -all -output SPI_coverage_report.txt
```

QuestaSim Snippets:







Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/SPI_top/DUT/cover__read_data_to_idle	SVA	✓	Off	24	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__read_add_to_idle	SVA	✓	Off	71	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__write_to_idle	SVA	✓	Off	94	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__check_cmd_to_read_data	SVA	✓	Off	56	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__check_cmd_to_read_addr	SVA	✓	Off	139	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__check_cmd_to_write	SVA	✓	Off	198	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__idle_to_check_cmd	SVA	✓	Off	409	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__read_data_seq	SVA	✓	Off	10	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__read_add_seq	SVA	✓	Off	6	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__write_data_seq	SVA	✓	Off	6	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__write_add_seq	SVA	✓	Off	10	1	Unli...	1	100%		✓	0	0	0 ns	0
/SPI_top/DUT/cover__assert_reset	SVA	✓	Off	247	1	Unli...	1	100%		✓	0	0	0 ns	0

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comn
/SPI_coverage_pkg/SPI_coverage		100.00%							
TYPE cov_cg		100.00%	100	100.00...		✓	auto(0)		
CVP cov_cg::rx_data_cp		100.00%	100	100.00...		✓			
CVP cov_cg::SS_n_cp		100.00%	100	100.00...		✓			
CVP cov_cg::MOSI_cp		100.00%	100	100.00...		✓			
CROSS cov_cg::SSn_MOSI_cross		100.00%	100	100.00...		✓			
INST \/SPI_coverage_pkg::SPI_coverage::cov_cg		100.00%	100	100.00...		✓		0	
CVP rx_data_cp		100.00%	100	100.00...		✓			
CVP SS_n_cp		100.00%	100	100.00...		✓			
CVP MOSI_cp		100.00%	100	100.00...		✓			
CROSS SSn_MOSI_cross		100.00%	100	100.00...		✓			

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Acti
/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_sequence_pkg::SPI_main_sequence::body/#ublk#33843783#45/immed__48	Immediate	SVA	on	0	1	
/SPI_top/DUT/assert__assert_reset	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__write_add_seq	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__write_data_seq	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__read_add_seq	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__read_data_seq	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__idle_to_check_cmd	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__check_cmd_to_write	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__check_cmd_to_read_addr	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__check_cmd_to_read_data	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__write_to_idle	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__read_add_to_idle	Concurrent	SVA	on	0	1	
/SPI_top/DUT/assert__read_data_to_idle	Concurrent	SVA	on	0	1	

```
# ************************************************************************
# UVM_INFO SPI_test.sv(39) @ 2: uvm_test_top [run_phase] reset deasserted.
# UVM_INFO SPI_test.sv(41) @ 2: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO SPI_test.sv(43) @ 10002: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 10007: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_scoreboard.sv(63) @ 10007: uvm_test_top.env.sb [SCOREBOARD] ================= SCOREBOARD REPORT ==================
# UVM_INFO SPI_scoreboard.sv(64) @ 10007: uvm_test_top.env.sb [SCOREBOARD] Total Checked: 15009
# UVM_INFO SPI_scoreboard.sv(65) @ 10007: uvm_test_top.env.sb [SCOREBOARD] Passed:        15009
# UVM_INFO SPI_scoreboard.sv(66) @ 10007: uvm_test_top.env.sb [SCOREBOARD] Failed:        0
# UVM_INFO SPI_scoreboard.sv(67) @ 10007: uvm_test_top.env.sb [SCOREBOARD] ======================================================
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     5
# [TEST_DONE]     1
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10007 ns  Iteration: 54  Instance: /SPI_top
```

Coverage report:

RAM CODES:

Design after debugging + Assertions:

```
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  //  mkntsh mawgoda
        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:   dout <= MEM[Rd_Addr];  //rd instead of wr
          default: dout <= 0;
        endcase
      end
      tx_valid <= (din[9] && din[8] && rx_valid) ? 1'b1 : 1'b0;
    end
  end

//ASSERTIONS
  property reset_sva;
    @(posedge clk) !rst_n |-> ##1 (tx_valid == 0 && dout == 0);
  endproperty
  assert property (reset_sva);
  cover property (reset_sva);

  property tx_valid_low;
```

```
    @(posedge clk) disable iff(!rst_n) (din[9:8] inside {2'b00,2'b01,2'b10}) |->
##1 tx_valid==0;
  endproperty
  assert property (tx_valid_low);
  cover property (tx_valid_low);

  property tx_valid_high;
    @(posedge clk) disable iff (!rst_n) (din[9:8] == 2'b11) |=> ##[1:$] $rose(
        tx_valid
    ) ##[1:$] $fell(
        tx_valid
    );
  endproperty

  assert property (tx_valid_high);
  cover property (tx_valid_high);

  property write_operation;
    @(posedge clk) disable iff (!rst_n) (din[9:8] == 2'b00) |=> ##[1:$] (din[9:8]
== 2'b01);
  endproperty
  assert property (write_operation);
  cover property (write_operation);

  property read_operation;
    @(posedge clk) disable iff (!rst_n) (din[9:8] == 2'b10) |=> ##[1:$] (din[9:8]
== 2'b11);
  endproperty
  assert property (read_operation);
  cover property (read_operation);

endmodule
```

Bugs Report:

- Ram was reading from the write address.
- Missing begin, end.

Before Debugging:

```
        else
            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 : dout <= MEM[Wr_Addr];
                    default : dout <= 0;
                endcase
            end
            tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
    end
```

After Debugging:

```
    if (rx_valid) begin  //  mkntsh mawgoda
      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:   dout <= MEM[Rd_Addr];  //rd instead of wr
        default: dout <= 0;
      endcase
    end
```

RAM GoldenModel:

```
module RAM_golden (clk,rst_n,din,rx_valid,dout,tx_valid);
parameter MEM_DEPTH =256 ;
parameter ADDR_SIZE =8 ;
input clk,rst_n,rx_valid;
input [9:0]din;
output reg [7:0]dout;
output reg tx_valid;

reg[7:0] write_addr;
reg [7:0] read_addr;
reg [7:0] mem [MEM_DEPTH-1:0];  //memory in hexa so (00)=8'b0 >>in mem.txt

always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
      dout<=8'b0;
      tx_valid <= 1'b0;
      read_addr<= 8'b0;
```

```
      write_addr<= 8'b0;
  end
  else  begin
    if (rx_valid) begin
      case (din[9:8])
        2'b00: begin
          write_addr<=din[7:0]; //hold
          tx_valid <= 1'b0;
        end
        2'b01: begin
          mem[write_addr]<=din[7:0]; //write
          tx_valid <= 1'b0;
        end
        2'b10: begin
          read_addr<=din[7:0] ;
          tx_valid <= 1'b0;
        end
        2'b11: begin
          dout <= mem[read_addr]; //read
          tx_valid <= 1'b1;
        end
        default: begin
          dout <= 0;
          tx_valid <= 0;
        end
      endcase
    end
    else begin
      tx_valid <= 1'b0;
    end
    //tx_valid <= (din[9] && din[8] && rx_valid) ? 1'b1 : 1'b0;
  end
end
endmodule
```

RAM interface:

```
interface RAM_interface(clk);
    input clk;
    logic rst_n, rx_valid;
    logic [9:0] din;
    logic [7:0] dout;
```

```
    logic tx_valid;
    logic [7:0] dout_golden;
    logic tx_valid_golden;
endinterface
```

RAM Config:

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

class RAM_config extends uvm_object;
`uvm_object_utils(RAM_config)
uvm_active_passive_enum is_active;
virtual RAM_interface RAM_if;

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

RAM Seq Item:

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

class RAM_seq_item_c extends uvm_sequence_item;
`uvm_object_utils(RAM_seq_item_c)

parameter MEM_DEPTH = 256;
parameter ADDR_SIZE = 8;
rand logic rst_n,rx_valid;
rand logic [9:0] din;
logic tx_valid;
logic [7:0] dout;
logic tx_valid_golden;
logic [7:0] dout_golden;

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

```
constraint rst_n_constraint {rst_n dist{1:=95 , 0:=5};}
constraint rx_valid_constraint {rx_valid dist{1:=95, 0:=5};}


function string convert2string();
    return $sformatf("%s rst_n=0b%0b, rx_valid=0b%0b, din=0b%0b, tx_valid=0b%0b,
dout=0b%0b,
    tx_valid_golden=0b%0b, dout_golden=0b%0b",
    super.convert2string(), rst_n, rx_valid, din, tx_valid, dout,
                        tx_valid_golden, dout_golden);
endfunction

function string convert2string_stimulus();
    return $sformatf("%s rst_n= 0b%0b, rx_valid=0b%b, din= 0b%b"
    ,super.convert2string(),rst_n, rx_valid,din);
endfunction
endclass
endpackage
```

RAM Read Sequence:

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

class RAM_rd_seq_c extends uvm_sequence #(RAM_seq_item_c);
`uvm_object_utils(RAM_rd_seq_c)

RAM_seq_item_c RAM_seq_item_obj;

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

task body();
    bit [1:0] prev_cmd = 2'b00;
        repeat(10000) begin
                RAM_seq_item_obj =
RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
                start_item(RAM_seq_item_obj);
                    if(prev_cmd == 2'b10) begin
```

```
                    assert(RAM_seq_item_obj.randomize() with {din[9:8] inside
{2'b10, 2'b11}; rx_valid == 1;});
                    end
                    else if (prev_cmd == 2'b11) begin
                    assert (RAM_seq_item_obj.randomize() with { din[9:8] ==
2'b10; rx_valid == 1; });
                    end
                    assert(!RAM_seq_item_obj.randomize())
                finish_item(RAM_seq_item_obj);

`uvm_info("READ_SEQ", $sformatf("READ command generated: %b",
RAM_seq_item_obj.din[9:8]), UVM_LOW)
                prev_cmd = RAM_seq_item_obj.din[9:8];
            end
        endtask
    endclass
endpackage
```

RAM Write Sequence:

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

class RAM_wr_seq_c extends uvm_sequence #(RAM_seq_item_c);
`uvm_object_utils(RAM_wr_seq_c)
RAM_seq_item_c RAM_seq_item_obj;

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

task body();
    bit [1:0] prev_cmd = 2'b00;
     repeat(10000) begin
        RAM_seq_item_obj = RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
            start_item(RAM_seq_item_obj);
                if(prev_cmd==2'b00) begin
                    assert(RAM_seq_item_obj.randomize() with {din[9:8] inside
{2'b00, 2'b01}; rx_valid == 1;});
                    if(!RAM_seq_item_obj.randomize()) begin
                        $display("ERROR: Randomization of write only failed at
iteration %0d", $time);
```

```
                        $display("Current values: %s",
RAM_seq_item_obj.convert2string_stimulus());
                    end
                end
            finish_item(RAM_seq_item_obj);
        prev_cmd = RAM_seq_item_obj.din[9:8];

`uvm_info("WRITE_SEQ", $sformatf("WRITE command generated: %b",
RAM_seq_item_obj.din[9:8]),UVM_LOW)
            end
        endtask
    endclass
endpackage
```

RAM Read&Write Sequence:

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

class RAM_rd_wr_seq_c extends uvm_sequence #(RAM_seq_item_c);
`uvm_object_utils(RAM_rd_wr_seq_c)
RAM_seq_item_c RAM_seq_item_obj;

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

task body();
    bit [1:0] prev_cmd;
        repeat(100) begin
                RAM_seq_item_obj =
RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
                if(prev_cmd==2'b00) begin
                    assert(RAM_seq_item_obj.randomize() with {din[9:8] inside
{2'b00,2'b01}; rx_valid==1;});
                end
                else if(prev_cmd==2'b01) begin
                    assert(RAM_seq_item_obj.randomize() with{ din[9:8] dist
{2'b00:= 40, 2'b10:=60}; rx_valid==1;});
                end
                else if(prev_cmd==2'b10) begin
```

```
                    assert(RAM_seq_item_obj.randomize() with {din[9:8] inside
{2'b11,2'b01}; rx_valid==1;});
                end
                else if(prev_cmd==2'b11) begin
                    assert(RAM_seq_item_obj.randomize() with{ din[9:8] dist
{2'b00:= 60, 2'b10:=40}; rx_valid==1;});
                end
                else begin //default
                assert(RAM_seq_item_obj.randomize() with {din[9:8] == 2'b00;
rx_valid == 1;});

                start_item(RAM_seq_item_obj);
                finish_item(RAM_seq_item_obj);

                assert(!RAM_seq_item_obj.randomize()) ;
`uvm_info("WRITE_READ_SEQ", $sformatf("CMD=%b (prev=%b) DATA/ADDR=%h",
RAM_seq_item_obj.din[9:8], prev_cmd, RAM_seq_item_obj.din[7:0]), UVM_LOW)
                prev_cmd = RAM_seq_item_obj.din[9:8];
            end
            end
        endtask
    endclass
endpackage
```

RAM Reset Seq:

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

class RAM_rst_seq_c extends uvm_sequence #(RAM_seq_item_c);
    `uvm_object_utils(RAM_rst_seq_c)
    RAM_seq_item_c RAM_seq_item_obj;

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

task body();
        RAM_seq_item_obj = RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
        start_item(RAM_seq_item_obj);
        RAM_seq_item_obj.rst_n=0;
```

```
        RAM_seq_item_obj.rx_valid=0;
        RAM_seq_item_obj.din=0;
        RAM_seq_item_obj.dout=0;
        RAM_seq_item_obj.tx_valid=0;
        finish_item(RAM_seq_item_obj);
endtask

endclass
endpackage
```

RAM Main Seq:

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

class RAM_main_seq_c extends uvm_sequence #(RAM_seq_item_c);
`uvm_object_utils(RAM_main_seq_c)

RAM_seq_item_c RAM_seq_item ;

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

task body();
        repeat(10000) begin
            RAM_seq_item = RAM_seq_item_c::type_id::create("RAM_seq_item");
            start_item(RAM_seq_item);
            assert (RAM_seq_item.randomize());
            finish_item(RAM_seq_item);
        end
        endtask
    endclass
endpackage
```

RAM Sequencer:

```
package RAM_sequencer_pkg;
import uvm_pkg::*;
```

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

class RAM_sequencer_c extends uvm_sequencer #(RAM_seq_item_c);
`uvm_component_utils(RAM_sequencer_c)

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

endclass
endpackage
```

RAM Coverage:

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

class RAM_coverage_c extends uvm_component;

`uvm_component_utils(RAM_coverage_c)
uvm_analysis_port #(RAM_seq_item_c) cov_export;
uvm_tlm_analysis_fifo #(RAM_seq_item_c) cov_fifo;
RAM_seq_item_c RAM_seq_item_obj;
virtual RAM_interface RAM_if;

covergroup RAM_coverage;
        din_cp: coverpoint RAM_seq_item_obj.din [9:8] {
                bins write_addr = {2'b00};
                bins write_data = {2'b01};
                bins read_addr  = {2'b10};
                bins read_data  = {2'b11};
            }
        transaction_order_cp: coverpoint RAM_seq_item_obj.din[9:8]{
                bins wa_to_wd = (2'b00 => 2'b01);
                bins ra_to_rd = (2'b10 => 2'b11);
                bins wa_wd_ra_rd = (2'b00 => 2'b01 => 2'b10 => 2'b11);
            }
        rx_valid_cp: coverpoint RAM_seq_item_obj.rx_valid {
                bins low = {0};
                bins high = {1};
            }
```

```
        cross_din_rx: cross din_cp, rx_valid_cp {
                ignore_bins read_opo = binsof(din_cp.read_data)&&
binsof(rx_valid_cp.low);
            }

        tx_valid_cp: coverpoint RAM_seq_item_obj.tx_valid {
                bins low = {0};
                bins high = {1};
            }

        cross_din_tx: cross din_cp,tx_valid_cp {
                ignore_bins loW_din =binsof(tx_valid_cp.low) &&
binsof(din_cp.read_data);
                ignore_bins high_wr_addr =binsof(tx_valid_cp.high) &&
binsof(din_cp.write_addr);
                ignore_bins high_wr_data =binsof(tx_valid_cp.high) &&
binsof(din_cp.write_data);
            }
        endgroup

function new(string name = "RAM_coverage_c", uvm_component parent = null);
            super.new(name,parent);
            RAM_coverage = 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);
            RAM_seq_item_obj =
RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
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(RAM_seq_item_obj);
                RAM_coverage.sample();
            end
```

```
endtask
    endclass
endpackage
```

RAM environment:

```
package RAM_env;
`include "uvm_macros.svh"
import uvm_pkg::*;
import RAM_Scoreboard_pkg::*;
import RAM_coverage_pkg::*;
import RAM_agent_pkg::*;

class RAM_env extends uvm_env;
`uvm_component_utils(RAM_env)

RAM_coverage_c RAM_cov;
RAM_agent RAM_agt;
RAM_Scoreboard RAM_sb;

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

function void build_phase(uvm_phase phase);
    super.build_phase(phase);
            RAM_cov = RAM_coverage_c::type_id::create("RAM_cov",this);
            RAM_agt =  RAM_agent::type_id::create("RAM_agt",this);
            RAM_sb = RAM_Scoreboard::type_id::create("RAM_sb",this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
            RAM_agt.agent_ap.connect(RAM_sb.sb_export);
            RAM_agt.agent_ap.connect(RAM_cov.cov_export);
endfunction
    endclass
endpackage
```

RAM Agent:

```
package RAM_agent_pkg;
`include "uvm_macros.svh"
```

```
import uvm_pkg::*;
import RAM_config_pkg::*;
import RAM_sequencer_pkg::*;
import RAM_seq_item_pkg::*;
import RAM_driver_pkg::*;
import RAM_monitor::*;

class RAM_agent extends uvm_agent;
`uvm_component_utils(RAM_agent)

RAM_config RAM_cfg;
RAM_sequencer_c RAM_seq;
RAM_driver_c RAM_dri;
RAM_monitor RAM_mon;
uvm_analysis_port #(RAM_seq_item_c) agent_ap;

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

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

    if (!uvm_config_db#(RAM_config)::get(this, "", "Config_key", RAM_cfg)) begin
                `uvm_fatal("Build phase", "unable to get config obj in agent")
            end

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

        if(RAM_cfg.is_active == UVM_ACTIVE) begin
        RAM_seq = RAM_sequencer_c::type_id::create("RAM_seq",this);
        RAM_dri = RAM_driver_c::type_id::create("RAM_dri",this);
    end
endfunction

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

     RAM_mon.RAM_if=RAM_cfg.RAM_if;
    RAM_mon.mon_ap.connect(agent_ap);

    if(RAM_cfg.is_active == UVM_ACTIVE) begin
        RAM_dri.RAM_if = RAM_cfg.RAM_if;
        RAM_dri.seq_item_port.connect(RAM_seq.seq_item_export);
```

```
    end
endfunction
    endclass
endpackage
```

RAM Driver:

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

class RAM_driver_c extends uvm_driver #(RAM_seq_item_c);
`uvm_component_utils(RAM_driver_c)
virtual RAM_interface RAM_if;
RAM_seq_item_c RAM_seq_item_obj;

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

task run_phase(uvm_phase phase);
    super.run_phase(phase);
        if (RAM_if == null) begin
                `uvm_info("DRIVER", "RAM_if is null in driver; waiting for
assignment", UVM_LOW)
            end
        wait (RAM_if != null);

    forever begin
            RAM_seq_item_obj =
RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
            seq_item_port.get_next_item(RAM_seq_item_obj);
            @(posedge RAM_if.clk);
            // Drive DUT signals only after ensuring RAM_if is valid
            RAM_if.rst_n    = RAM_seq_item_obj.rst_n;
            RAM_if.rx_valid = RAM_seq_item_obj.rx_valid;
            RAM_if.din      = RAM_seq_item_obj.din;

            seq_item_port.item_done();
`uvm_info("Run phase", RAM_seq_item_obj.convert2string_stimulus(), UVM_HIGH)
            end
        endtask
```

```
    endclass
endpackage
```

RAM Monitor:

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

class RAM_monitor extends uvm_monitor;
`uvm_component_utils(RAM_monitor)

virtual RAM_interface RAM_if;
RAM_seq_item_c RAM_seq_item_obj;
uvm_analysis_port #(RAM_seq_item_c) 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);
            // wait until the virtual interface handle is set by the agent/config
            if (RAM_if == null) begin
                `uvm_info("MON", "RAM_if is null in monitor; waiting for
assignment", UVM_LOW)
            end
            wait (RAM_if != null);
        forever begin
                RAM_seq_item_obj =
RAM_seq_item_c::type_id::create("RAM_seq_item_obj");
                @(posedge RAM_if.clk);
                RAM_seq_item_obj.rst_n = RAM_if.rst_n;
                RAM_seq_item_obj.rx_valid = RAM_if.rx_valid;
                RAM_seq_item_obj.din = RAM_if.din;

                //DUT outputs
```

```
                RAM_seq_item_obj.dout = RAM_if.dout;
                RAM_seq_item_obj.tx_valid = RAM_if.tx_valid;

                //Golden outputs
                RAM_seq_item_obj.dout = RAM_if.dout_golden;
                RAM_seq_item_obj.tx_valid = RAM_if.tx_valid_golden;

                mon_ap.write(RAM_seq_item_obj);
`uvm_info("run phase", RAM_seq_item_obj.convert2string(), UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

RAM ScoreBoard:

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

class RAM_Scoreboard extends uvm_scoreboard;
`uvm_component_utils(RAM_Scoreboard)

uvm_analysis_export #(RAM_seq_item_c) sb_export;
uvm_tlm_analysis_fifo #(RAM_seq_item_c) dut_fifo;
uvm_tlm_analysis_fifo #(RAM_seq_item_c) ref_fifo;

RAM_seq_item_c dut_item;

int error_count = 0;
int correct_count = 0;

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

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

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

```
      sb_export.connect(dut_fifo.analysis_export);
endfunction

task run_phase(uvm_phase phase);
    super.run_phase(phase);
      forever begin
        dut_fifo.get(dut_item);

        if ((dut_item.dout !== dut_item.dout) || (dut_item.tx_valid !==
dut_item.tx_valid)) begin
  `uvm_error("SCOREBOARD", $sformatf("Mismatch: DUT(dout=%0h, tx_valid=%0b) |
REF(dout=%0h, tx_valid=%0b)",
                                                dut_item.dout, dut_item.tx_valid,
dut_item.dout, dut_item.tx_valid))
          error_count++;
        end
        else correct_count++;
      end
    endtask


function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info("SCOREBOARD", "================== SCOREBOARD REPORT
==================", UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Total Checked: %0d", correct_count +
error_count), UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Passed:        %0d", correct_count),
UVM_NONE)
    `uvm_info("SCOREBOARD", $sformatf("Failed:        %0d", error_count),
UVM_NONE)
    `uvm_info("SCOREBOARD",
"=========================================================", UVM_NONE)
endfunction

  endclass
endpackage
```

RAM Test:

```
package RAM_test_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import RAM_env::*;
    import RAM_config_pkg::*;
    import RAM_main_seq_pkg::*;
    import RAM_rst_seq_pkg::*;
    import RAM_agent_pkg::*;
    import RAM_rd_seq_pkg::*;
    import RAM_rd_wr_seq_pkg::*;
    import RAM_wr_seq_pkg::*;


    class RAM_test extends uvm_test;
        `uvm_component_utils(RAM_test)
        RAM_env env;
        RAM_config RAM_cfg;
        RAM_main_seq_c RAM_main_seq;
        RAM_rst_seq_c RAM_rst_seq;
        virtual RAM_interface RAM_if;
        //---------->new
        RAM_rd_seq_c  RAM_rd_seq;
        RAM_rd_wr_seq_c RAM_rd_wr_seq;
        RAM_wr_seq_c  RAM_wr_seq;

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

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            env=RAM_env::type_id::create("env", this);
            RAM_cfg=RAM_config::type_id::create("RAM_cfg");
            RAM_main_seq=RAM_main_seq_c::type_id::create("RAM_main_seq");
            RAM_rst_seq=RAM_rst_seq_c::type_id::create("RAM_rst_seq");

RAM_rd_seq    = RAM_rd_seq_c::type_id::create("RAM_rd_seq");
RAM_rd_wr_seq = RAM_rd_wr_seq_c::type_id::create("RAM_rd_wr_seq");
RAM_wr_seq    = RAM_wr_seq_c::type_id::create("RAM_wr_seq");

    if(!uvm_config_db #(virtual RAM_interface)::get(this, "", "RAM_if",
RAM_cfg.RAM_if))
```

```
            `uvm_fatal("build_phase","unable to get config object from the env in
test pkg")
            uvm_config_db #(RAM_config)::set(this,"*", "Config_key", RAM_cfg);
            RAM_cfg.is_active=UVM_ACTIVE;
        endfunction

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("run_phase", "welcome", UVM_MEDIUM)
            RAM_rst_seq.start(env.RAM_agt.RAM_seq);
            RAM_main_seq.start(env.RAM_agt.RAM_seq);
            phase.drop_objection(this);

        endtask
    endclass
endpackage
```

RAM Top:

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

module RAM_top();
    bit clk;
    initial begin
        clk=0;
        forever #10 clk=~clk;
    end

    RAM_interface RAM_if (clk);

   RAM_golden goldenmodel(
        .din   (RAM_if.din),
        .clk     (RAM_if.clk),
        .rst_n   (RAM_if.rst_n),
        .rx_valid(RAM_if.rx_valid),
        .dout    (RAM_if.dout_golden),
        .tx_valid(RAM_if.tx_valid_golden)
    );

   RAM DUT(
```

```
        .din   (RAM_if.din),
        .clk     (RAM_if.clk),
        .rst_n   (RAM_if.rst_n),
        .rx_valid(RAM_if.rx_valid),
        .dout    (RAM_if.dout),
        .tx_valid(RAM_if.tx_valid)
    );

   //RAM_assertions SVA(RAM_if);

    initial begin
        uvm_config_db #(virtual RAM_interface)::set(null,"uvm_test_top",
"RAM_if", RAM_if);
        run_test("RAM_test");

    end
endmodule
```

DO file:

run.do

```
vlib work
vmap work work

vlog -work work RAM.sv +define+SIM
vlog -work work RAM_interface.sv
vlog -work work RAM_goldenmodel.sv
vlog -work work RAM_Config.sv
vlog -work work RAM_seq_item.sv
vlog -work work RAM_rd_seq.sv
vlog -work work RAM_wr_seq.sv
vlog -work work RAM_rd_wr_seq.sv
vlog -work work RAM_rst_seq.sv
vlog -work work RAM_main_seq.sv
vlog -work work RAM_driver.sv
vlog -work work RAM_mon.sv
vlog -work work RAM_agent.sv
vlog -work work RAM_sb.sv
vlog -work work RAM_coverage.sv
vlog -work work RAM_env.sv
vlog -work work RAM_sequencer.sv
vlog -work work RAM_test.sv
vlog -work work RAM_top.sv

vsim -voptargs=+acc work.RAM_top -classdebug -uvmcontrol=all -cover -sv_seed random

add wave -position insertpoint sim:/RAM_top/RAM_if/*
add wave -position insertpoint sim:/RAM_top/DUT/*

coverage save RAM_coverage.ucdb -onexit
run -all

vcover report RAM_coverage.ucdb -details -annotate -all -output RAM_coverage_report.txt
```

QuestaSim Snippets:

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included
/RAM_top/DUT/cover__read_operation	SVA	✓	Off	1947	1	Unli...	1	100%		✓
/RAM_top/DUT/cover__write_operation	SVA	✓	Off	1883	1	Unli...	1	100%		✓
/RAM_top/DUT/cover__tx_valid_high	SVA	✓	Off	1647	1	Unli...	1	100%		✓
/RAM_top/DUT/cover__tx_valid_low	SVA	✓	Off	6846	1	Unli...	1	100%		✓
/RAM_top/DUT/cover__reset_sva	SVA	✓	Off	470	1	Unli...	1	100%		✓

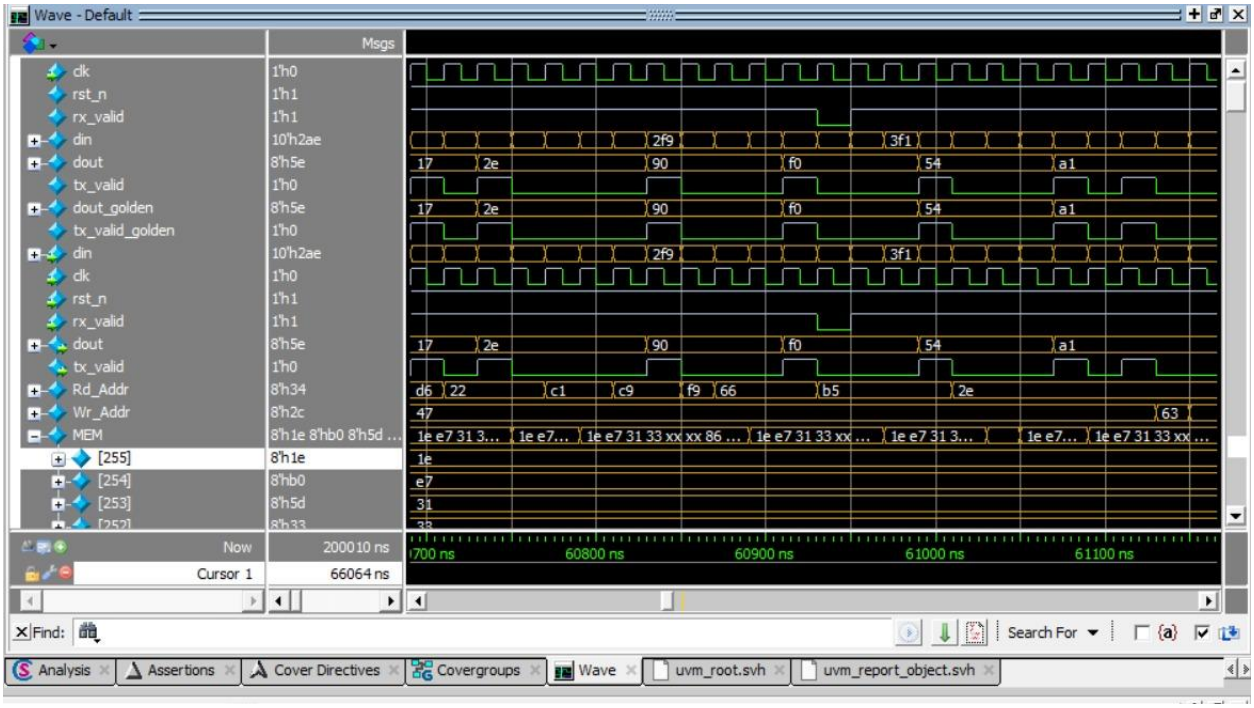
Covergroups

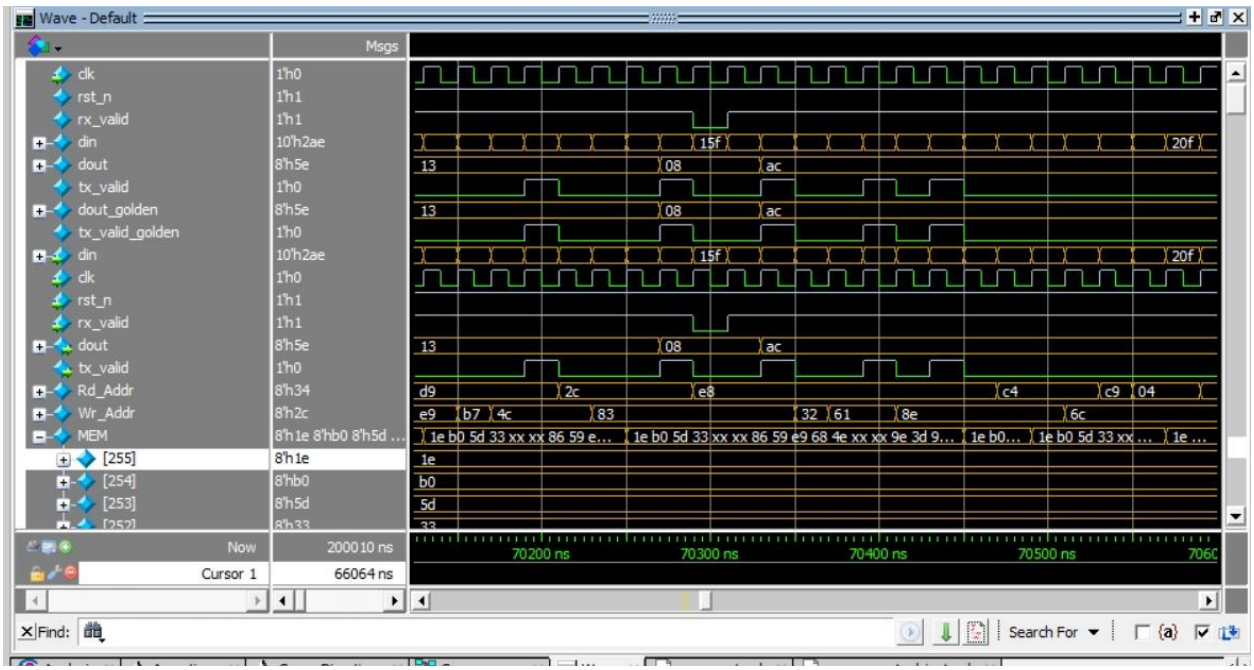
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instar
/RAM_coverage_pkg/RAM_coverage_c		100.00%					
TYPE RAM_coverage		100.00%	100	100.00...			
CVP RAM_coverage::din_cp		100.00%	100	100.00...			
CVP RAM_coverage::transaction_order_cp		100.00%	100	100.00...			
CVP RAM_coverage::rx_valid_cp		100.00%	100	100.00...			
CVP RAM_coverage::tx_valid_cp		100.00%	100	100.00...			
CROSS RAM_coverage::cross_din_rx		100.00%	100	100.00...			
CROSS RAM_coverage::cross_din_tx		100.00%	100	100.00...			
INST \/RAM_coverage_pkg::RAM_coverage_c::RAM_coverage		100.00%	100	100.00...			
CVP din_cp		100.00%	100	100.00...			
CVP transaction_order_cp		100.00%	100	100.00...			
CVP rx_valid_cp		100.00%	100	100.00...			
CVP tx_valid_cp		100.00%	100	100.00...			
CROSS cross_din_rx		100.00%	100	100.00...			
CROSS cross_din_tx		100.00%	100	100.00...			

Assertions

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	
/RAM_wr_seq_pkg::RAM_wr_seq_c::body/#ublk#267486743#19/immed__20	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#18/immed__19	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#21/immed__22	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#24/immed__25	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#27/immed__28	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#30/immed__31	Immediate	SVA	on	0	0	
/RAM_rd_wr_seq_pkg::RAM_rd_wr_seq_c::body/#ublk#18342935#30/immed__36	Immediate	SVA	on	0	0	
/RAM_rd_seq_pkg::RAM_rd_seq_c::body/#ublk#267512855#17/immed__26	Immediate	SVA	on	0	0	
/RAM_rd_seq_pkg::RAM_rd_seq_c::body/#ublk#267512855#17/#ublk#267512855#20/immed__2...	Immediate	SVA	on	0	0	
/RAM_rd_seq_pkg::RAM_rd_seq_c::body/#ublk#267512855#17/#ublk#267512855#23/immed__2...	Immediate	SVA	on	0	0	
/RAM_main_seq_pkg::RAM_main_seq_c::body/#ublk#267764039#16/immed__19	Immediate	SVA	on	0	1	
/RAM_top/DUT/assert__reset_sva	Concurrent	SVA	on	0	1	
/RAM_top/DUT/assert__tx_valid_low	Concurrent	SVA	on	0	1	
/RAM_top/DUT/assert__tx_valid_high	Concurrent	SVA	on	0	1	
/RAM_top/DUT/assert__write_operation	Concurrent	SVA	on	0	1	
/RAM_top/DUT/assert__read_operation	Concurrent	SVA	on	0	1	

```
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 200010: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_sb.sv(50) @ 200010: uvm_test_top.env.RAM_sb [SCOREBOARD] ================= SCOREBOARD REPORT =================
# UVM_INFO RAM_sb.sv(51) @ 200010: uvm_test_top.env.RAM_sb [SCOREBOARD] Total Checked: 10001
# UVM_INFO RAM_sb.sv(52) @ 200010: uvm_test_top.env.RAM_sb [SCOREBOARD] Passed:        10001
# UVM_INFO RAM_sb.sv(53) @ 200010: uvm_test_top.env.RAM_sb [SCOREBOARD] Failed:        0
# UVM_INFO RAM_sb.sv(54) @ 200010: uvm_test_top.env.RAM_sb [SCOREBOARD] ======================================================
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   10
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     5
# [TEST_DONE]     1
# [run_phase]     1
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 200010 ns  Iteration: 61  Instance: /RAM_top
```





Coverage report:

SPI_Slave_Wrapper:

Design + Assertions:

```
module SPI_Wrapper (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   RAM_instance   (rx_data_din,clk,rst_n,rx_valid,tx_data_dout,tx_valid);

SLAVE SLAVE_instance
(MOSI,MISO,SS_n,clk,rst_n,rx_data_din,rx_valid,tx_data_dout,tx_valid);

//ASSERTIONS
property assert_reset;
    @(posedge clk) (!rst_n) |=> (MISO=='0);
endproperty
assert property (assert_reset);
cover property (assert_reset);


property MISO_STABLE_NOT_READ;
  @(posedge clk) disable iff (!rst_n)
  $fell(SS_n) |=> (!MOSI) [*0:3] ##1 ($stable(MISO) throughout (!SS_n));
endproperty
assert property (MISO_STABLE_NOT_READ);
cover property (MISO_STABLE_NOT_READ);

endmodule
```

Wrapper Golden:

```
module SPI_Wrapper_golden ( MOSI, SS_n, clk, rst_n,MISO_golden);
input MOSI, SS_n, clk, rst_n;
output MISO_golden;
```

```
wire [9:0] rx_data_golden;//din
wire rx_valid_golden;
wire [7:0] dout_golden;//tx_data
wire tx_valid_golden;

spi_golden spi_golden (MOSI, MISO_golden,
SS_n,clk,rst_n,rx_data_golden,rx_valid_golden,
dout_golden[7:0], tx_valid_golden);

RAM_golden ram_golden (clk,rst_n,rx_data_golden, rx_valid_golden,dout_golden,
tx_valid_golden);

endmodule
```

Wrapper Interface:

```
interface SPI_Wrapper_interface (input bit clk);
logic rst_n;
logic MOSI;
logic MISO;
logic SS_n;
logic MISO_golden;
endinterface
```

Wrapper Shared Package:

```
package SPI_Wrapper_shared_pkg;
    bit [5:0] count = 0;
    bit read_data_flag=0;
    bit read_addr_flag=0;
    logic [10:0] arr_of_data=11'b0;
    bit is_read=0;
    bit have_address_to_read=0;
    int limit=13;
endpackage
```

Wrapper Config:

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

class SPI_Wrapper_config extends uvm_object;
`uvm_object_utils(SPI_Wrapper_config)

virtual SPI_Wrapper_interface wrapper_vif;
uvm_active_passive_enum is_active;

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

Wrapper Seq Item:

```
package SPI_Wrapper_seq_item_pkg;
  import SPI_shared_pkg::*;
  `include "uvm_macros.svh"
  import uvm_pkg::*;

  class SPI_Wrapper_seq_item extends uvm_sequence_item;
    `uvm_object_utils(SPI_Wrapper_seq_item)

    bit clk;
    rand logic MOSI, SS_n, rst_n, tx_valid;
    rand logic [7:0] tx_data;
    rand logic [10:0] MOSI_data;  // 11-bit array

    // DUT outputs
    logic [9:0] rx_data;
    logic MISO, rx_valid;

    // Golden model outputs
    logic [9:0] rx_data_golden;
    logic MISO_golden, rx_valid_golden;
```

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

    function string convert2string();
      return $sformatf(
          "rst_n=%0b, SS_n=%0b, MOSI_data=0x%0h, tx_data=0x%0h, tx_valid=%0b |
DUT: MISO=%0b rx_data=0x%0h rx_valid=%0b | GOLD: MISO=%0b rx_data=0x%0h
rx_valid=%0b",
          rst_n,
          SS_n,
          MOSI_data,
          tx_data,
          tx_valid,
          MISO,
          rx_data,
          rx_valid,
          MISO_golden,
          rx_data_golden,
          rx_valid_golden
      );
    endfunction

    function string convert2string_stimulus();
      return $sformatf(
          "rst_n=%0b, SS_n=%0b, MOSI_data=0x%0h, tx_data=0x%0h, tx_valid=%0b",
          rst_n,
          SS_n,
          MOSI_data,
          tx_data,
          tx_valid
      );
    endfunction

    constraint reset_constraint {
      rst_n dist {
        0 := 5,
        1 := 95
      };
    }

    constraint valid_MOSI_command {MOSI_data[10:8] inside {3'b000, 3'b001,
3'b110, 3'b111};}

    constraint ready_to_read {
```

```
      if (count >= 15)
      tx_valid == 1;
      else
      tx_valid == 0;
    }

    function void post_randomize();
      if (count == 0) arr_of_data = MOSI_data;
      is_read = (arr_of_data[10:8] == 3'b111) ? 1 : 0;
      limit = (is_read) ? 23 : 13;

      SS_n = (count == limit) ? 1 : 0;

      if (arr_of_data[10:8] == 3'b110) have_address_to_read = 1'b1;
      if (is_read || (!rst_n)) have_address_to_read = 1'b0;

      //
      if ((count > 0) && (count < 12)) begin
        MOSI = arr_of_data[11-count];
      end

      //count
      if (!rst_n) begin
        count = 0;
      end else begin
        if (count == limit) count = 0;
        else count++;
      end

    endfunction


  endclass
endpackage
```

Wrapper Sequence:

```
package SPI_Wrapper_sequence_pkg;
import uvm_pkg::*;
import SPI_Wrapper_seq_item_pkg::*;
`include "uvm_macros.svh"
```

```
class wrapper_reset_sequence extends uvm_sequence #(SPI_Wrapper_seq_item);
`uvm_object_utils(wrapper_reset_sequence)
SPI_Wrapper_seq_item seq_item;

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

task body();
    seq_item = SPI_Wrapper_seq_item::type_id::create("seq_item");
    start_item(seq_item);
    seq_item.rst_n = 0;
    seq_item.MOSI = 0;
    seq_item.SS_n = 1;
    seq_item.MOSI_data = 0;
    finish_item(seq_item);
endtask
endclass

class wrapper_write_only_sequence extends uvm_sequence #(SPI_Wrapper_seq_item);
    `uvm_object_utils(wrapper_write_only_sequence)
    SPI_Wrapper_seq_item seq_item;
    bit [2:0] last_op = 3'b000;

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

    task body();
        repeat(1000) begin
            seq_item = SPI_Wrapper_seq_item::type_id::create("seq_item");
            start_item(seq_item);

            if (last_op == 3'b000) begin
                assert(seq_item.randomize() with {
                    rst_n == 1;
                    MOSI_data[10:8] inside {3'b000, 3'b001};
                });
            end
            else begin
                assert(seq_item.randomize() with {
                    rst_n == 1;
                    MOSI_data[10:8] == 3'b000;
                });
```

```
            end

            finish_item(seq_item);
            last_op = seq_item.MOSI_data[10:8];
        end
    endtask
endclass

class wrapper_read_only_sequence extends uvm_sequence #(SPI_Wrapper_seq_item);
    `uvm_object_utils(wrapper_read_only_sequence)
    SPI_Wrapper_seq_item seq_item;
    bit [2:0] last_op = 3'b110;

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

    task body();
        repeat(1000) begin
            seq_item = SPI_Wrapper_seq_item::type_id::create("seq_item");
            start_item(seq_item);

            if (last_op == 3'b110) begin
                assert(seq_item.randomize() with {
                    rst_n == 1;
                    MOSI_data[10:8] inside {3'b110, 3'b111};
                });
            end
            else begin
                assert(seq_item.randomize() with {
                    rst_n == 1;
                    MOSI_data[10:8] == 3'b110;
                });
            end

            finish_item(seq_item);
            last_op = seq_item.MOSI_data[10:8];
        end
    endtask
endclass

class wrapper_write_read_sequence extends uvm_sequence #(SPI_Wrapper_seq_item);
`uvm_object_utils(wrapper_write_read_sequence)
    SPI_Wrapper_seq_item seq_item;
    bit [2:0] last_op = 3'b000;
```

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

    task body();
        repeat(1000) begin
            seq_item = SPI_Wrapper_seq_item::type_id::create("seq_item");
            start_item(seq_item);

            case(last_op)
                3'b000: begin
                    assert(seq_item.randomize() with {
                        rst_n == 1;
                        MOSI_data[10:8] inside {3'b000, 3'b001};
                    });
                end
                3'b001: begin
                    assert(seq_item.randomize() with {
                        rst_n == 1;
                        MOSI_data[10:8] dist {3'b110 := 60, 3'b000 := 40};
                    });
                end
                3'b110: begin
                    assert(seq_item.randomize() with {
                        rst_n == 1;
                        MOSI_data[10:8] inside {3'b110, 3'b111};
                    });
                end
                3'b111: begin
                    assert(seq_item.randomize() with {
                        rst_n == 1;
                        MOSI_data[10:8] dist {3'b000 := 60, 3'b110 := 40};
                    });
                end
                default: begin
                    assert(seq_item.randomize() with {rst_n == 1;});
                end
            endcase

            finish_item(seq_item);
            last_op = seq_item.MOSI_data[10:8];
        end
    endtask
endclass
```

```
endpackage
```

Wrapper Sequencer:

```
package SPI_Wrapper_sequencer_pkg;
import uvm_pkg::*;
import SPI_Wrapper_seq_item_pkg::*;
`include "uvm_macros.svh"

class SPI_Wrapper_sequencer extends uvm_sequencer #(SPI_Wrapper_seq_item);
    `uvm_component_utils(SPI_Wrapper_sequencer)

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

endclass

endpackage
```

Wrapper Environment:

```
package SPI_Wrapper_env_pkg;
import uvm_pkg::*;
import SPI_Wrapper_agent_pkg::*;
import SPI_Wrapper_scoreboard_pkg::*;
`include "uvm_macros.svh"

class SPI_Wrapper_env extends uvm_env;
`uvm_component_utils(SPI_Wrapper_env)

SPI_Wrapper_agent wrapper_agt;
SPI_Wrapper_scoreboard sb;


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

function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        wrapper_agt = SPI_Wrapper_agent::type_id::create("wrapper_agt", this);
        sb = SPI_Wrapper_scoreboard::type_id::create("sb", this);
```

```
endfunction

function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        wrapper_agt.agent_analport.connect(sb.sb_export);
endfunction
endclass
endpackage
```

Wrapper Agent:

```
package SPI_Wrapper_agent_pkg;
import uvm_pkg::*;
import SPI_Wrapper_sequencer_pkg::*;
import SPI_Wrapper_driver_pkg::*;
import SPI_Wrapper_monitor_pkg::*;
import SPI_Wrapper_config_pkg::*;
import SPI_Wrapper_seq_item_pkg::*;
`include "uvm_macros.svh"

class SPI_Wrapper_agent extends uvm_agent;
`uvm_component_utils(SPI_Wrapper_agent)

SPI_Wrapper_sequencer sqr;
SPI_Wrapper_driver driv;
SPI_Wrapper_monitor mon;
SPI_Wrapper_config wrapper_cfg;
uvm_analysis_port #(SPI_Wrapper_seq_item) agent_analport;

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

function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db#(SPI_Wrapper_config)::get(this, "", "Config_key",
wrapper_cfg))
                `uvm_fatal("build_phase", "unable to get wrapper config")

        if(wrapper_cfg.is_active==UVM_ACTIVE) begin
                sqr = SPI_Wrapper_sequencer::type_id::create("sqr", this);
                driv = SPI_Wrapper_driver::type_id::create("driv", this);
        end
```

```
        mon = SPI_Wrapper_monitor::type_id::create("mon", this);
        agent_analport = new("agent_analport", this);
endfunction

function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        mon.wrapper_vif = wrapper_cfg.wrapper_vif;
        mon.mon_ap.connect(agent_analport);
        if (wrapper_cfg.is_active == UVM_ACTIVE) begin
                driv.wrapper_vif = wrapper_cfg.wrapper_vif;
                driv.seq_item_port.connect(sqr.seq_item_export);
        end
endfunction

endclass
endpackage
```

Wrapper Driver:

```
package SPI_Wrapper_driver_pkg;
import uvm_pkg::*;
import SPI_Wrapper_seq_item_pkg::*;
import SPI_Wrapper_config_pkg::*;
`include "uvm_macros.svh"

class SPI_Wrapper_driver extends uvm_driver #(SPI_Wrapper_seq_item);
`uvm_component_utils(SPI_Wrapper_driver)

virtual SPI_Wrapper_interface wrapper_vif;
SPI_Wrapper_seq_item stim_seq_item;

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

task run_phase(uvm_phase phase);
        super.run_phase(phase);
forever begin
        stim_seq_item = SPI_Wrapper_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);

        wrapper_vif.rst_n = stim_seq_item.rst_n;
```

```
        wrapper_vif.SS_n = stim_seq_item.SS_n;
        wrapper_vif.MOSI = stim_seq_item.MOSI;

        @(negedge wrapper_vif.clk);

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

Wrapper Monitor:

```
package SPI_Wrapper_monitor_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)

virtual SPI_Wrapper_interface wrapper_vif;
SPI_Wrapper_seq_item rsp_seq_item;
uvm_analysis_port #(SPI_Wrapper_seq_item) mon_ap;

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

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

task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = SPI_Wrapper_seq_item::type_id::create("rsp_seq_item");
            @(negedge wrapper_vif.clk);
            rsp_seq_item.rst_n = wrapper_vif.rst_n;
            rsp_seq_item.SS_n = wrapper_vif.SS_n;
            rsp_seq_item.MOSI = wrapper_vif.MOSI;
            rsp_seq_item.MISO = wrapper_vif.MISO;
            rsp_seq_item.MISO_golden = wrapper_vif.MISO_golden;
```

```
            mon_ap.write(rsp_seq_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) sb_export;
uvm_tlm_analysis_fifo #(SPI_Wrapper_seq_item) sb_fifo;
SPI_Wrapper_seq_item seq_item_sb;

int error_count = 0;
int correct_count = 0;

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

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

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

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

```
        check_data(seq_item_sb);
         end
endtask

task check_data(SPI_Wrapper_seq_item item);
    if (item.MISO !== item.MISO_golden) begin
        error_count++;
    `uvm_error("SCOREBOARD", $sformatf("MISO Mismatch! Expected: %0b, Got: %0b |
%s",
                                        item.MISO_golden, item.MISO,
item.convert2string()))
        end
    else correct_count++;
endtask

function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("SCOREBOARD", "============ WRAPPER SCOREBOARD REPORT
============", UVM_NONE)
        `uvm_info("SCOREBOARD", $sformatf("Total Checked: %0d", correct_count +
error_count), UVM_NONE)
        `uvm_info("SCOREBOARD", $sformatf("Passed:        %0d", correct_count),
UVM_NONE)
        `uvm_info("SCOREBOARD", $sformatf("Failed:        %0d", error_count),
UVM_NONE)
        `uvm_info("SCOREBOARD",
"==================================================", UVM_NONE)
endfunction
endclass
endpackage
```

Wrapper Test:

```
package SPI_Wrapper_test_pkg;
  import uvm_pkg::*;
  import SPI_Wrapper_env_pkg::*;
  import SPI_Wrapper_config_pkg::*;
  import SPI_Wrapper_sequence_pkg::*;
  import SPI_env_pkg::*;
  import SPI_config_pkg::*;
  import RAM_env::*;
  import RAM_config_pkg::*;

  `include "uvm_macros.svh"
```

```
  class SPI_Wrapper_test extends uvm_test;
    `uvm_component_utils(SPI_Wrapper_test)

    SPI_Wrapper_env               wrapper_env;        // Active wrapper
environment
    SPI_env                       spi_environment;    // Passive SPI
environment
    RAM_env                       ram_environment;    // Passive RAM
environment

    SPI_Wrapper_config            wrapper_config_obj_test;
    SPI_config                    spi_config_obj_test;
    RAM_config                    ram_config_obj_test;

    virtual SPI_Wrapper_interface wrapper_vif;
    virtual SPI_interface         spi_test_vif;
    virtual RAM_interface         ram_test_vif;

    wrapper_reset_sequence        reset_seq;
    wrapper_write_only_sequence   write_seq;
    wrapper_read_only_sequence    read_seq;
    wrapper_write_read_sequence   write_read_seq;

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

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

      // Create all three environments
      wrapper_env = SPI_Wrapper_env::type_id::create("wrapper_env", this);
      spi_environment = SPI_env::type_id::create("spi_environment", this);
      ram_environment = RAM_env::type_id::create("ram_environment", this);

      // Create config objects
      wrapper_config_obj_test =
SPI_Wrapper_config::type_id::create("wrapper_config_obj_test");
      spi_config_obj_test = SPI_config::type_id::create("spi_config_obj_test");
      ram_config_obj_test = RAM_config::type_id::create("ram_config_obj_test");

      // Create sequences
      reset_seq = wrapper_reset_sequence::type_id::create("reset_seq");
      write_seq = wrapper_write_only_sequence::type_id::create("write_seq");
```

```
      read_seq = wrapper_read_only_sequence::type_id::create("read_seq");
      write_read_seq =
wrapper_write_read_sequence::type_id::create("write_read_seq");

      // Configure Wrapper (ACTIVE)
      if (!uvm_config_db#(virtual SPI_Wrapper_interface)::get(this, "",
"Config_key", wrapper_vif))
        `uvm_fatal("build_phase", "Wrapper interface not found in config_db")
      wrapper_config_obj_test.wrapper_vif = wrapper_vif;
      uvm_config_db#(SPI_Wrapper_config)::set(this, "*", "Config_key",
wrapper_config_obj_test);
      wrapper_config_obj_test.is_active = UVM_ACTIVE;

      // Configure SPI (PASSIVE)
      if (!uvm_config_db#(virtual SPI_interface)::get(this, "", "Config_key",
spi_test_vif))
        `uvm_fatal("build_phase", "SPI interface not found")
      spi_config_obj_test.SPI_vif = spi_test_vif;
      uvm_config_db#(SPI_config)::set(this, "*", "Config_key",
spi_config_obj_test);
      spi_config_obj_test.is_active = UVM_PASSIVE;

      // Configure RAM (PASSIVE)
      if (!uvm_config_db#(virtual RAM_interface)::get(this, "", "Config_key",
ram_test_vif))
        `uvm_fatal("build_phase", "RAM interface not found")
      ram_config_obj_test.RAM_if = ram_test_vif;
      uvm_config_db#(RAM_config)::set(this, "*", "Config_key",
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);

      `uvm_info("WRAPPER_TEST", "Starting SPI Wrapper Test", UVM_LOW)

      reset_seq.start(wrapper_env.wrapper_agt.sqr);
      write_seq.start(wrapper_env.wrapper_agt.sqr);
      read_seq.start(wrapper_env.wrapper_agt.sqr);
      write_read_seq.start(wrapper_env.wrapper_agt.sqr);

      `uvm_info("WRAPPER_TEST", "Test Completed", UVM_LOW)
```

```
      #100;
      phase.drop_objection(this);
    endtask
  endclass
endpackage
```

Wrapper Top:

```
module SPI_Wrapper_top ();

  import uvm_pkg::*;
  import SPI_Wrapper_test_pkg::*;

  `include "uvm_macros.svh"

  bit clk;

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

  SPI_Wrapper_interface wrapper_if (clk);
  SPI_interface spi_if (clk);
  RAM_interface ram_if (clk);

  SPI_Wrapper DUT
(wrapper_if.MOSI,wrapper_if.MISO,wrapper_if.SS_n,wrapper_if.clk,wrapper_if.rst_n)
;

  SPI_Wrapper_golden GOLDEN (wrapper_if.MOSI, wrapper_if.SS_n, wrapper_if.clk,
wrapper_if.rst_n, wrapper_if.MISO_golden);

  SLAVE DUT_SLAVE
(spi_if.MOSI,spi_if.MISO,spi_if.SS_n,spi_if.clk,spi_if.rst_n,spi_if.rx_data,spi_i
f.rx_valid,spi_if.tx_data,spi_if.tx_valid);

  spi_golden GM_SLAVE
(spi_if.MOSI,spi_if.MISO_golden,spi_if.SS_n,spi_if.clk,spi_if.rst_n,spi_if.rx_dat
a_golden,spi_if.rx_valid_golden,spi_if.tx_data,spi_if.tx_valid);

  RAM DUT_RAM
(ram_if.din,clk,ram_if.rst_n,ram_if.rx_valid,ram_if.dout,ram_if.tx_valid);
```

```
  RAM_golden GM_RAM
(clk,ram_if.rst_n,ram_if.din,ram_if.rx_valid,ram_if.dout_golden,ram_if.tx_valid_g
olden);

  assign spi_if.MOSI     = DUT.MOSI;
  assign spi_if.SS_n     = DUT.SS_n;
  assign spi_if.rst_n    = DUT.rst_n;
  assign spi_if.tx_valid = DUT.tx_valid;
  assign spi_if.tx_data  = DUT.tx_data_dout;

  assign ram_if.rx_valid = DUT.rx_valid;
  assign ram_if.rst_n    = DUT.rst_n;
  assign ram_if.din      = DUT.rx_data_din;

  initial begin
    uvm_config_db#(virtual SPI_Wrapper_interface)::set(null, "uvm_test_top",
"Config_key", wrapper_if);
    uvm_config_db#(virtual SPI_interface)::set(null, "uvm_test_top",
"Config_key", spi_if);
    uvm_config_db#(virtual RAM_interface)::set(null, "uvm_test_top",
"Config_key", ram_if);
    run_test("SPI_Wrapper_test");
  end

  initial begin
    $readmemh("ram.data", DUT.RAM_instance.MEM);
    $readmemh("ram.data", DUT_RAM.MEM);
    $readmemh("ram_golden.data", GM_RAM.mem);
    $readmemh("ram_golden.data", GOLDEN.ram_golden.mem);
  end
endmodule
```

Do File:

run.do

```
vlib work

vlog -sv -work work -f src_files.list +define+SIM

vsim -voptargs=+acc -classdebug -uvmcontrol=all -assertdebug work.SPI_Wrapper_top

#add wave -position insertpoint sim:/SPI_Wrapper_top/DUT/*
#add wave -position insertpoint sim:/SPI_Wrapper_top/DUT_RAM/*
#add wave -position insertpoint sim:/SPI_Wrapper_top/DUT_SLAVE/*

#add wave -position insertpoint sim:/SPI_Wrapper_top/GOLDEN/*
#add wave -position insertpoint sim:/SPI_Wrapper_top/GM_SLAVE/*
#add wave -position insertpoint sim:/SPI_Wrapper_top/GM_RAM/*

add wave -position insertpoint sim:/SPI_Wrapper_top/wrapper_if/*
add wave -position insertpoint sim:/SPI_Wrapper_top/spi_if/*
add wave -position insertpoint sim:/SPI_Wrapper_top/ram_if/*

run -all
```

Src Files:

```
RAM_agent.sv
RAM_Config.sv
RAM_coverage.sv
RAM_driver.sv
RAM_env.sv
RAM_goldenmodel.sv
RAM_interface.sv
RAM_main_seq.sv
RAM_mon.sv
RAM_rd_seq.sv
RAM_rd_wr_seq.sv
RAM_rst_seq.sv
RAM_seq_item.sv
RAM_sb.sv
RAM_sequencer.sv
RAM_wr_seq.sv
RAM.sv

SPI_agent.sv
SPI_config_object.sv
SPI_coverage.sv
SPI_driver.sv
SPI_golden.v
SPI_env.sv
SPI_interface.sv
SPI_monitor.sv
SPI_scoreboard.sv
SPI_seq_item.sv
SPI_sequence.sv
SPI_sequencer.sv
SPI_shared_pkg.sv
SPI_slave.sv

wrapper_agent.sv
wrapper_config_obj.sv
wrapper_driver.sv
wrapper_env.sv
wrapper_golden.sv
wrapper_interface.sv
wrapper_monitor.sv
wrapper_scoreboard.sv
wrapper_seq_item.sv
wrapper_sequence.sv
wrapper_sequencer.sv
wrapper_test.sv
wrapper_shared_pkg.sv
wrapper_top.sv
wrapper.sv
```

QuestaSnippets:

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/SPI_Wrapper_top/DUT/cover__MISO_STABLE_NOT_READ	SVA	✓	Off	193	1	Unli...	1	100%		✓	0	160	5 ns	386
/SPI_Wrapper_top/DUT/cover__assert_reset	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	1 ns	1
/SPI_Wrapper_top/DUT/RAM_instance/cover__read_operation	SVA	✓	Off	893	1	Unli...	1	100%		✓	0	8800	2991 ns	1786
/SPI_Wrapper_top/DUT/RAM_instance/cover__write_operation	SVA	✓	Off	902	1	Unli...	1	100%		✓	1040	13280	1915 ns	1817
/SPI_Wrapper_top/DUT/RAM_instance/cover__tx_valid_high	SVA	✓	Off	619	1	Unli...	1	100%		✓	720	4800	3847 ns	2485
/SPI_Wrapper_top/DUT/RAM_instance/cover__tx_valid_low	SVA	✓	Off	2422	1	Unli...	1	100%		✓	0	160	5 ns	2422
/SPI_Wrapper_top/DUT/RAM_instance/cover__reset_sva	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	1 ns	1
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__read_data_to_idle	SVA	✓	Off	82	1	Unli...	1	100%		✓	0	80	2073 ns	82
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__read_add_to_idle	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	2045 ns	1
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__write_to_idle	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	80	29 ns	109
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__check_cmd_to_read_...	SVA	✓	Off	83	1	Unli...	1	100%		✓	0	80	2049 ns	83
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__check_cmd_to_read_...	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	2021 ns	1
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__check_cmd_to_write	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	80	5 ns	109
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__idle_to_check_cmd	SVA	✓	Off	193	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__read_data_seq	SVA	✓	Off	30	1	Unli...	1	100%		✓	0	160	2101 ns	253
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__read_add_seq	SVA	✓	Off	53	1	Unli...	1	100%		✓	80	160	2045 ns	247
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__write_data_seq	SVA	✓	Off	44	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__write_add_seq	SVA	✓	Off	65	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT/SLAVE_instance/cover__assert_reset	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	1 ns	1
/SPI_Wrapper_top/DUT_SLAVE/cover__read_data_to_idle	SVA	✓	Off	82	1	Unli...	1	100%		✓	0	80	2073 ns	82
/SPI_Wrapper_top/DUT_SLAVE/cover__read_add_to_idle	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	2045 ns	1
/SPI_Wrapper_top/DUT_SLAVE/cover__write_to_idle	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	80	29 ns	109
/SPI_Wrapper_top/DUT_SLAVE/cover__check_cmd_to_read_data	SVA	✓	Off	83	1	Unli...	1	100%		✓	0	80	2049 ns	83
/SPI_Wrapper_top/DUT_SLAVE/cover__check_cmd_to_read_addr	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	2021 ns	1
/SPI_Wrapper_top/DUT_SLAVE/cover__check_cmd_to_write	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	80	5 ns	109
/SPI_Wrapper_top/DUT_SLAVE/cover__idle_to_check_cmd	SVA	✓	Off	193	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT_SLAVE/cover__read_data_seq	SVA	✓	Off	30	1	Unli...	1	100%		✓	0	160	2101 ns	253
/SPI_Wrapper_top/DUT_SLAVE/cover__read_add_seq	SVA	✓	Off	53	1	Unli...	1	100%		✓	80	160	2045 ns	247
/SPI_Wrapper_top/DUT_SLAVE/cover__write_data_seq	SVA	✓	Off	44	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT_SLAVE/cover__write_add_seq	SVA	✓	Off	65	1	Unli...	1	100%		✓	0	80	3 ns	193
/SPI_Wrapper_top/DUT_SLAVE/cover__assert_reset	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	1 ns	1
/SPI_Wrapper_top/DUT_RAM/cover__read_operation	SVA	✓	Off	893	1	Unli...	1	100%		✓	0	8800	2991 ns	1786
/SPI_Wrapper_top/DUT_RAM/cover__write_operation	SVA	✓	Off	902	1	Unli...	1	100%		✓	1040	13280	1915 ns	1817
/SPI_Wrapper_top/DUT_RAM/cover__tx_valid_high	SVA	✓	Off	619	1	Unli...	1	100%		✓	720	4800	3847 ns	2485
/SPI_Wrapper_top/DUT_RAM/cover__tx_valid_low	SVA	✓	Off	2422	1	Unli...	1	100%		✓	0	160	5 ns	2422
/SPI_Wrapper_top/DUT_RAM/cover__reset_sva	SVA	✓	Off	1	1	Unli...	1	100%		✓	0	80	1 ns	1

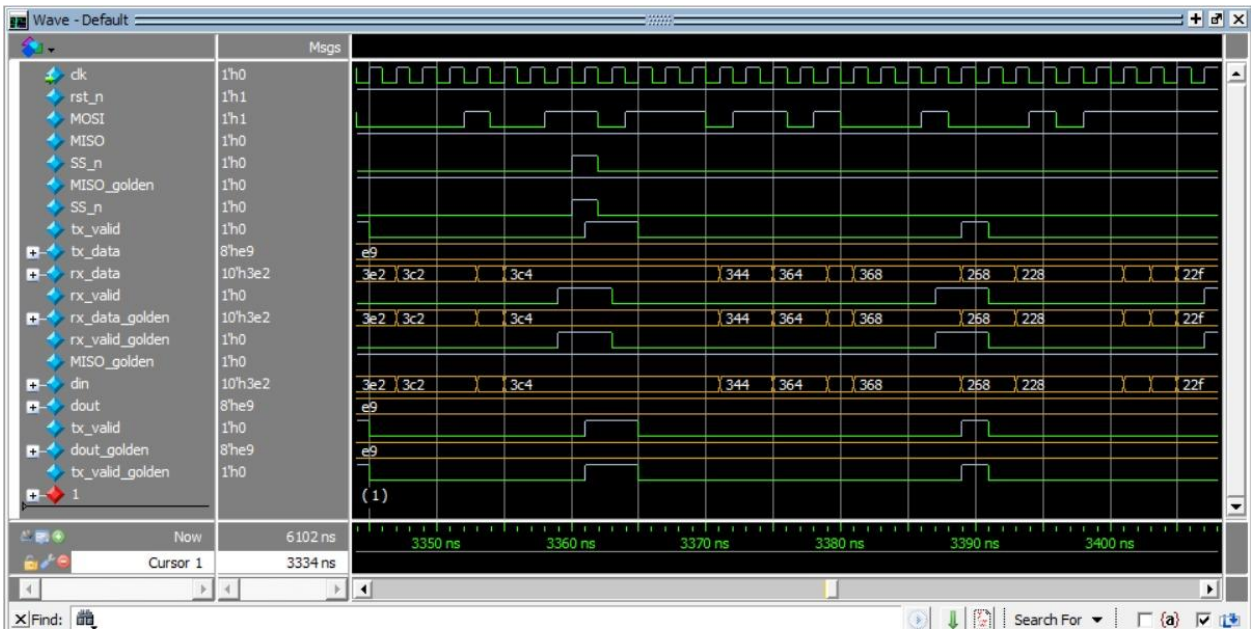
Covergroups

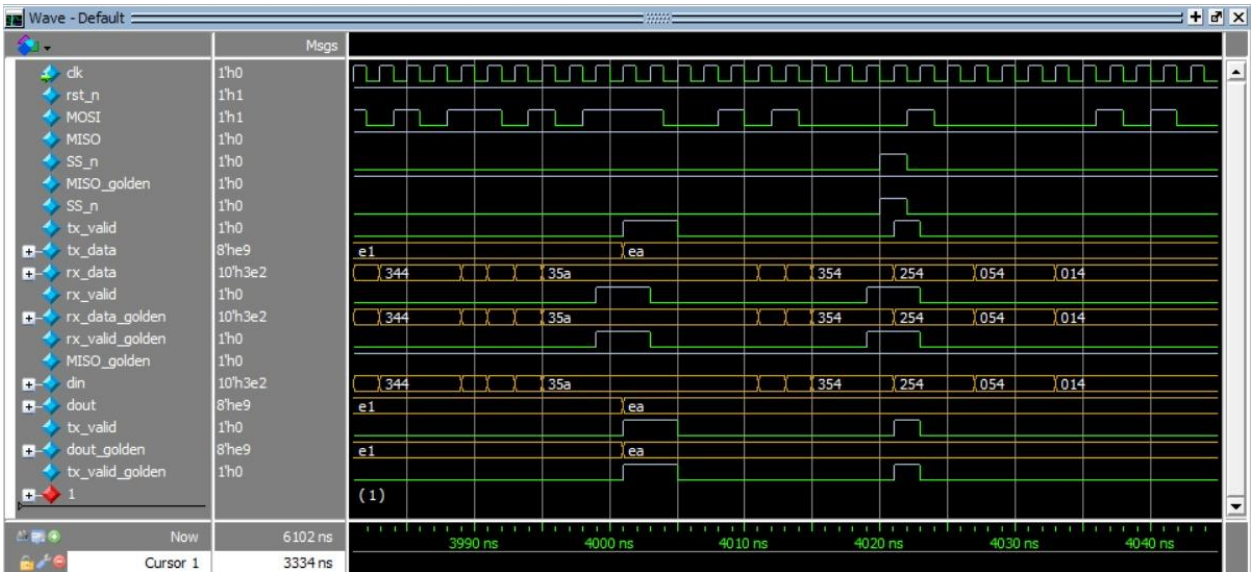
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	C
/RAM_coverage_pkg/RAM_coverage_c		100.00%							
TYPE RAM_coverage		100.00%	100	100.00...		✓	auto(0)		
CROSS RAM_coverage::cross_din_rx		100.00%	100	100.00...		✓			
CROSS RAM_coverage::cross_din_tx		100.00%	100	100.00...		✓			
CVP RAM_coverage::din_cp		100.00%	100	100.00...		✓			
CVP RAM_coverage::rx_valid_cp		100.00%	100	100.00...		✓			
CVP RAM_coverage::transaction_order_cp		100.00%	100	100.00...		✓			
CVP RAM_coverage::tx_valid_cp		100.00%	100	100.00...		✓			
INST \/RAM_coverage_pkg::RAM_coverage_c::RAM...		100.00%	100	100.00...		✓			0
/SPI_coverage_pkg/SPI_coverage		100.00%							
TYPE cov_cg		100.00%	100	100.00...		✓	auto(0)		
CROSS cov_cg::SSn_MOSI_cross		100.00%	100	100.00...		✓			
CVP cov_cg::MOSI_cp		100.00%	100	100.00...		✓			
CVP cov_cg::rx_data_cp		100.00%	100	100.00...		✓			
CVP cov_cg::SS_n_cp		100.00%	100	100.00...		✓			
INST \/SPI_coverage_pkg::SPI_coverage::cov_cg		100.00%	100	100.00...		✓			0

Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Co
/SPI_Wrapper_sequence_pkg::wrapper_read_only_sequence::body/#ublk#252081719#76/immed...	Immediate	SVA	on	0	329	
/SPI_Wrapper_sequence_pkg::wrapper_write_read_sequence::body/#ublk#252081719#106/imm...	Immediate	SVA	on	0	331	
/SPI_Wrapper_sequence_pkg::wrapper_write_read_sequence::body/#ublk#252081719#112/imm...	Immediate	SVA	on	0	162	
/SPI_Wrapper_sequence_pkg::wrapper_write_read_sequence::body/#ublk#252081719#118/imm...	Immediate	SVA	on	0	342	
/SPI_Wrapper_sequence_pkg::wrapper_write_read_sequence::body/#ublk#252081719#124/imm...	Immediate	SVA	on	0	165	
/SPI_Wrapper_sequence_pkg::wrapper_write_read_sequence::body/#ublk#252081719#130/imm...	Immediate	SVA	on	0	0	
/SPI_Wrapper_top/DUT/assert__assert_reset	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT/assert__MISO_STABLE_NOT_READ	Concurrent	SVA	on	0	193	
/SPI_Wrapper_top/DUT/RAM_instance/assert__reset_sva	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT/RAM_instance/assert__tx_valid_low	Concurrent	SVA	on	0	2422	
/SPI_Wrapper_top/DUT/RAM_instance/assert__tx_valid_high	Concurrent	SVA	on	0	619	
/SPI_Wrapper_top/DUT/RAM_instance/assert__write_operation	Concurrent	SVA	on	0	902	
/SPI_Wrapper_top/DUT/RAM_instance/assert__read_operation	Concurrent	SVA	on	0	893	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__assert_reset	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__write_add_seq	Concurrent	SVA	on	0	65	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__write_data_seq	Concurrent	SVA	on	0	44	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__read_add_seq	Concurrent	SVA	on	0	53	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__read_data_seq	Concurrent	SVA	on	0	30	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__idle_to_check_cmd	Concurrent	SVA	on	0	193	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__check_cmd_to_write	Concurrent	SVA	on	0	109	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__check_cmd_to_read_addr	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__check_cmd_to_read_data	Concurrent	SVA	on	0	83	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__write_to_idle	Concurrent	SVA	on	0	109	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__read_add_to_idle	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT/SLAVE_instance/assert__read_data_to_idle	Concurrent	SVA	on	0	82	
/SPI_Wrapper_top/DUT_SLAVE/assert__assert_reset	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT_SLAVE/assert__write_add_seq	Concurrent	SVA	on	0	65	
/SPI_Wrapper_top/DUT_SLAVE/assert__write_data_seq	Concurrent	SVA	on	0	44	
/SPI_Wrapper_top/DUT_SLAVE/assert__read_add_seq	Concurrent	SVA	on	0	53	
/SPI_Wrapper_top/DUT_SLAVE/assert__read_data_seq	Concurrent	SVA	on	0	30	
/SPI_Wrapper_top/DUT_SLAVE/assert__idle_to_check_cmd	Concurrent	SVA	on	0	193	
/SPI_Wrapper_top/DUT_SLAVE/assert__check_cmd_to_write	Concurrent	SVA	on	0	109	
/SPI_Wrapper_top/DUT_SLAVE/assert__check_cmd_to_read_addr	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT_SLAVE/assert__check_cmd_to_read_data	Concurrent	SVA	on	0	83	
/SPI_Wrapper_top/DUT_SLAVE/assert__write_to_idle	Concurrent	SVA	on	0	109	
/SPI_Wrapper_top/DUT_SLAVE/assert__read_add_to_idle	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT_SLAVE/assert__read_data_to_idle	Concurrent	SVA	on	0	82	
/SPI_Wrapper_top/DUT_RAM/assert__reset_sva	Concurrent	SVA	on	0	1	
/SPI_Wrapper_top/DUT_RAM/assert__tx_valid_low	Concurrent	SVA	on	0	2422	
/SPI_Wrapper_top/DUT_RAM/assert__tx_valid_high	Concurrent	SVA	on	0	619	
/SPI_Wrapper_top/DUT_RAM/assert__write_operation	Concurrent	SVA	on	0	902	
/SPI_Wrapper_top/DUT_RAM/assert__read_operation	Concurrent	SVA	on	0	893	

```
# UVM_INFO wrapper_test.sv(90) @ 6002: uvm_test_top [WRAPPER_TEST] Test Completed
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6102: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_sb.sv(50) @ 6102: uvm_test_top.ram_environment.RAM_sb [SCOREBOARD] ================== SCOREBOARD REPORT ==================
# UVM_INFO RAM_sb.sv(51) @ 6102: uvm_test_top.ram_environment.RAM_sb [SCOREBOARD] Total Checked: 3051
# UVM_INFO RAM_sb.sv(52) @ 6102: uvm_test_top.ram_environment.RAM_sb [SCOREBOARD] Passed:        3051
# UVM_INFO RAM_sb.sv(53) @ 6102: uvm_test_top.ram_environment.RAM_sb [SCOREBOARD] Failed:        0
# UVM_INFO RAM_sb.sv(54) @ 6102: uvm_test_top.ram_environment.RAM_sb [SCOREBOARD] ====================================================
# UVM_INFO SPI_scoreboard.sv(63) @ 6102: uvm_test_top.spi_environment.sb [SCOREBOARD] ================== SCOREBOARD REPORT ==================
# UVM_INFO SPI_scoreboard.sv(64) @ 6102: uvm_test_top.spi_environment.sb [SCOREBOARD] Total Checked: 9153
# UVM_INFO SPI_scoreboard.sv(65) @ 6102: uvm_test_top.spi_environment.sb [SCOREBOARD] Passed:        9153
# UVM_INFO SPI_scoreboard.sv(66) @ 6102: uvm_test_top.spi_environment.sb [SCOREBOARD] Failed:        0
# UVM_INFO SPI_scoreboard.sv(67) @ 6102: uvm_test_top.spi_environment.sb [SCOREBOARD] ========================================================
# UVM_INFO wrapper_scoreboard.sv(50) @ 6102: uvm_test_top.wrapper_env.sb [SCOREBOARD] ============ WRAPPER SCOREBOARD REPORT ============
# UVM_INFO wrapper_scoreboard.sv(51) @ 6102: uvm_test_top.wrapper_env.sb [SCOREBOARD] Total Checked: 3051
# UVM_INFO wrapper_scoreboard.sv(52) @ 6102: uvm_test_top.wrapper_env.sb [SCOREBOARD] Passed:        3051
# UVM_INFO wrapper_scoreboard.sv(53) @ 6102: uvm_test_top.wrapper_env.sb [SCOREBOARD] Failed:        0
# UVM_INFO wrapper_scoreboard.sv(54) @ 6102: uvm_test_top.wrapper_env.sb [SCOREBOARD] ====================================================
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   21
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]    15
# [TEST_DONE]     1
# [WRAPPER_TEST]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 6102 ns  Iteration: 54  Instance: /SPI_Wrapper_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```


Wave - Default
Msgs
clk 1'h0
rst_n 1'h1
MOSI 1'h1
MISO 1'h0
SS_n 1'h0
MISO_golden 1'h0
SS_n 1'h0
tx_valid 1'h0
tx_data 8'he9
rx_data 10'h3e2
rx_valid 1'h0
rx_data_golden 10'h3e2
rx_valid_golden 1'h0
MISO_golden 1'h0
din 10'h3e2
dout 8'he9
tx_valid 1'h0
dout_golden 8'he9
tx_valid_golden 1'h0
1
Now 6102 ns
Cursor 1 3334 ns
3350 ns
3360 ns
3370 ns
3380 ns
3390 ns
3400 ns
Find:
Search For


Wave - Default
Msgs
clk 1'h0
rst_n 1'h1
MOSI 1'h1
MISO 1'h0
SS_n 1'h0
MISO_golden 1'h0
SS_n 1'h0
tx_valid 1'h0
tx_data 8'he9
rx_data 10'h3e2
rx_valid 1'h0
rx_data_golden 10'h3e2
rx_valid_golden 1'h0
MISO_golden 1'h0
din 10'h3e2
dout 8'he9
tx_valid 1'h0
dout_golden 8'he9
tx_valid_golden 1'h0
1
Now 6102 ns
Cursor 1 3334 ns
3990 ns
4000 ns
4010 ns
4020 ns
4030 ns
4040 ns

Assertions Table of SPI Slave

Feature	Assertion Description
Reset behavior	Whenever reset (!rst_n) is asserted, MISO, rx_data, and rx_valid should all be low: @(posedge clk) (!rst_n |-> (!MISO && rx_data == '0 && rx_valid == '0))
Write address sequence	When SS_n falls and 3 cycles of !MOSI occur, after 10 cycles, rx_valid and SS_n should be high: ($fell(SS_n) ##1(!MOSI)[*3]) |-> ##10 (rx_valid && SS_n)
Write data sequence	When SS_n falls, followed by 2 cycles of !MOSI and 1 cycle of MOSI, after 10 cycles rx_valid and SS_n must be high: ($fell(SS_n) ##1(!MOSI)[*2] ##1(MOSI)) |-> ##10 (rx_valid && SS_n)
Read address sequence	When SS_n falls, followed by 2 cycles of MOSI and 1 cycle of !MOSI, after 10 cycles rx_valid and SS_n must be high: ($fell(SS_n) ##1(MOSI)[*2] ##1(!MOSI)) |-> ##10 (rx_valid && SS_n)
Read data sequence	When SS_n falls and 3 cycles of MOSI occur, after 10 cycles, rx_valid and SS_n should be high: ($fell(SS_n) ##1(MOSI)[*3]) |-> ##10 (rx_valid && SS_n)
IDLE → CHK_CMD	When current state cs == IDLE and SS_n is low, next state should be CHK_CMD: (cs == IDLE && !SS_n) |-> (ns == CHK_CMD)

CHK_CMD → WRITE	**When cs == CHK_CMD, SS_n low, and MOSI == 0, next state should be WRITE: (cs == CHK_CMD && !SS_n && !MOSI) |-> (ns == WRITE)**
CHK_CMD → READ_ADDR	**When cs == CHK_CMD, SS_n low, MOSI == 1, and address not yet received, next state should be READ_ADD: (cs == CHK_CMD && !SS_n && MOSI && !received_address) |-> (ns == READ_ADD)**
CHK_CMD → READ_DATA	**When cs == CHK_CMD, SS_n low, MOSI == 1, and address already received, next state should be READ_DATA: (cs == CHK_CMD && !SS_n && MOSI && received_address) |-> (ns == READ_DATA)**
WRITE → IDLE	**When cs == WRITE and SS_n high, next state should go to IDLE: (cs == WRITE && SS_n) |-> (ns == IDLE)**
READ_ADD → IDLE	**When cs == READ_ADD and SS_n high, next state should go to IDLE: (cs == READ_ADD && SS_n) |-> (ns == IDLE)**
READ_DATA → IDLE	**When cs == READ_DATA and SS_n high, next state should go to IDLE: (cs == READ_DATA && SS_n) |-> (ns == IDLE)**

Assertion Table of Ram:

Feature	Assertion
Whenever the reset (rst_n) is asserted, MISO is low	@(posedge clk) (!rst_n |-> ~MISO);
During reset, tx_valid and dout are low	@(posedge clk) !rst_n |-> ##1 (tx_valid == 0 && dout == 0);
When din[9:8] indicates a non-read operation (00, 01, or 10), tx_valid must remain low	@(posedge clk) disable iff(!rst_n) (din[9:8] inside {2'b00,2'b01,2'b10}) |-> ##1 (tx_valid == 0);
When din[9:8] indicates a read operation (11), tx_valid must pulse high then low	@(posedge clk) disable iff(!rst_n) (din[9:8] == 2'b11) |=> ##[1:$] $rose(tx_valid) ##[1:$] $fell(tx_valid);
Write sequence must occur in order: write address (00) followed by write data (01)	@(posedge clk) disable iff(!rst_n) (din[9:8] == 2'b00) |=> ##[1:$] (din[9:8] == 2'b01);
Read sequence must occur in order: read address (10) followed by read data (11)	@(posedge clk) disable iff(!rst_n) (din[9:8] == 2'b10) |=> ##[1:$] (din[9:8] == 2'b11);
MISO remains stable whenever the SPI is not performing a read operation (din[9:8] != 2'b11)	@(posedge clk) disable iff(!rst_n) (din[9:8] != 2'b11) |=> $stable(MISO);

Assertions Table of Wrapper:

Feature	Assertion
Whenever the reset (rst_n) is asserted, MISO is low	@(posedge clk) (!rst_n) |=> (MISO == '0);
When chip select (SS_n) falls, and MOSI indicates a non-read command, MISO remains stable while SS_n is low	@(posedge clk) disable iff(!rst_n) $fell(SS_n) \|=> (!MOSI)[*0:3] ##1 ($stable(MISO) throughout (!SS_n));

Verification Plan:

	Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
3	**reset check**	When the reset signal is activated, the MISO output should be zero	Directed reset assertion using wrapper_reset_sequence	Covering the reset transition for MISO output	In the scoreboard, comparing the outputs with the expected of a Golden model
4	**IDLE**	When reset is deactivated and SS_n is High, the cs should be IDLE	Randomizing the SS_n and constraining it to be low for 13 cycle for all cases except for the READ_DATA, the SS_n would be low for 23 cycle	Covering the Transition of the SS_n after 13 cycle except for the READ_DATA would be for 23 cycle	In the scoreboard, comparing the outputs with the expected of a Golden model
5	**MISO Stability**	MISO should remain stable as long as it is not a read data operation	Generate write operations (000, 001, 110) and verify MISO stays constant	Cover MISO stability during WRITE, CHK_CMD, READ_ADD states	In the scoreboard, comparing the outputs with the expected of a Golden model
6	**Write Address Flow**	Complete write address transaction: SS_n low → 3'b000 command → 8-bit address → SS_n high after 13 cycles	Use wrapper_write_only_sequence to generate 000 commands with proper timing	Cover transition (3'b000) and SS_n timing (1=> 0[*13] => 1)	Verify SPI rx_data[9:8]=2'b00 reaches RAM din[9:8]=2'b00 and RAM stores address correctly
7	**Write Data Flow**	Complete write data transaction: SS_n low → 3'b001 command → 8-bit data → SS_n high after 13 cycles	Use wrapper_write_only_sequence to generate 001 commands following 000	Cover transition (3'b000 => 3'b001) and verify sequential write	Verify SPI rx_data reaches RAM, data written to memory at stored address
8	**Read Address Flow**	Complete read address transaction: SS_n low → 3'b110 command → 8-bit address → SS_n high after 13 cycles	Use wrapper_read_only_sequence to generate 110 commands	Cover transition (3'b110) and SS_n timing for read address	Verify SPI rx_data[9:8]=2'b10 reaches RAM din[9:8]=2'b10 and RAM stores read address
9	**Read Data Flow**	Complete read data transaction: SS_n low → 3'b111 command → RAM retrieves data → MISO outputs 8 bits → SS_n high after 23 cycles	Use wrapper_read_only_sequence to generate 111 commands following 110	Cover transition (3'b110 => 3'b111) and SS_n extended timing (1=> 0[*23] => 1)	Verify RAM tx_valid triggers, RAM dout reaches SPI tx_data, MISO outputs correct data serially
10	**Full Write-Read Sequence**	Complete transaction: Write Address → Write Data → Read Address → Read Data	Use wrapper_write_read_sequence with exact sequence: 000 → 001 → 110 → 111	Cover full transition bin: (3'b000 => 3'b001 => 3'b110 => 3'b111)	Verify written data matches read data, complete end-to-end flow validation
11	**SPI to RAM Interface**	rx_data from SPI correctly transferred to RAM din, rx_valid propagated	Generate all command types and monitor interface signals	Cover all command types reaching RAM: 2'b00, 2'b01, 2'b10, 2'b11	Verify rx_data[9:0] = din[9:0] and rx_valid timing matches
12	**RAM to SPI Interface**	tx_data from RAM correctly transferred to MISO, tx_valid triggers MISO output	Generate read operations and monitor RAM outputs	Cover tx_valid assertion during READ_DATA state	Verify dout[7:0] = tx_data[7:0] and MISO serializes correctly
13	**SS_n Timing - Write**	SS_n must remain low for exactly 13 clock cycles during write operations (000, 001,	Generate write operations with SS_n timing constraints	Cover normal_transaction bin: (1=> 0[*13] => 1)	Assert SS_n duration = 13 cycles for non-read-data operations
14	**SS_n Timing - Read**	SS_n must remain low for exactly 23 clock cycles during read data operation (111)	Generate read data operations with SS_n extended timing	Cover extended_transaction bin: (1=> 0[*23] => 1)	Assert SS_n duration = 23 cycles for read data operation
15	**Command Cross**	Ensure all valid command combinations with SS_n timing are tested	Randomize commands with proper SS_n timing constraints	Cross coverage: cmd_cp × SS_n_cp, with illegal bins for invalid	Verify no illegal combinations occur (e.g., read_data with 13-cycle SS_n)
16	**Sequential Write Operations**	Multiple consecutive write operations: 000 → 001 → 000 → 001	Use wrapper_write_only_sequence with repeated patterns	Cover bins: wa_to_wd, wa_to_wa transitions	Verify multiple addresses and data written correctly to RAM
17	**Sequential Read Operations**	Multiple consecutive read operations: 110 → 111 → 110 → 111	Use wrapper_read_only_sequence with repeated patterns	Cover bins: ra_to_rd, ra_to_ra transitions	Verify multiple read addresses and data retrieved correctly
18	**Invalid Command Handling**	System behavior with invalid MOSI patterns (not 000, 001, 110, 111)	Constraint excludes invalid commands (design assumes valid only)	N/A - design constraint ensures only valid commands	Design assumes valid commands
19	**MOSI Command Detection**	SPI correctly decodes 3-bit commands from MOSI serial data	Generate all valid commands: 000, 001, 110, 111	Cover all command bins in cmd_cp coverpoint	Verify rx_data[9:8] matches expected command after 11 serial bits
20	**Counter Operation**	SPI counter decrements correctly from 10 to 0 during data reception	Monitor internal counter during all transaction types	Track counter behavior across all states	Verify counter resets to 10 in CHK_CMD and decrements properly
21	**Received Address Flag**	received_address flag set correctly after READ_ADD, used for READ_DATA state transition	Generate read sequences and monitor internal flag	Cover READ_ADD to READ_DATA transition	Verify flag enables READ_DATA state and clears after read completes
22	**RAM Memory Integrity**	Data written to RAM at address X can be read back from address X	Write known data to specific addresses, then read back	Cover address space sampling (min, max, random addresses)	Compare written data vs read data for same address
23	**Reset During Transaction**	System recovers correctly if reset asserted during active transaction	Assert reset at random times during SS_n low periods	Cover reset during each state: WRITE, READ_ADD, READ_DATA	Verify all outputs zero, state returns to IDLE, no data corruption