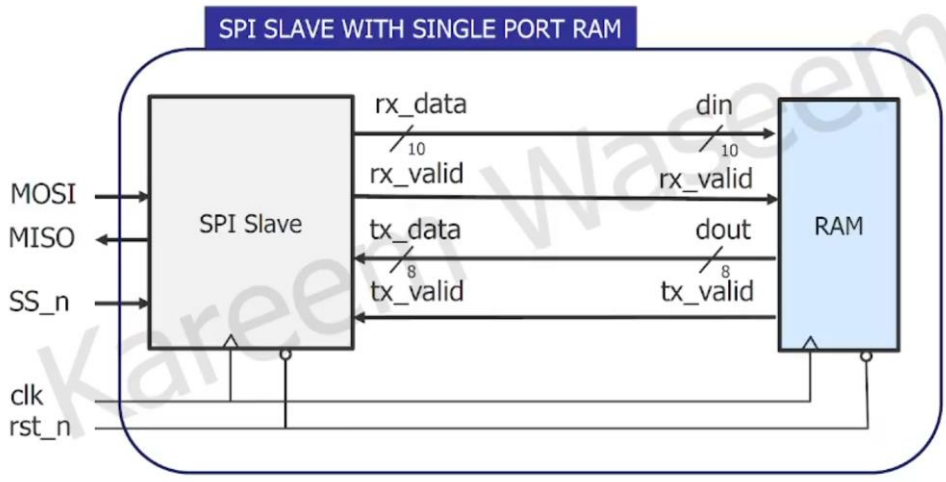
SPI - UVM Project

SPI Slave Wrapper with Single Port RAM

Presented For: Digital Verification Using SV & UVM Diploma

Under Supervision of: Eng. Kareem Waseem ❤️

💙 Digital Verification Engineers 😎😂
Ahmed Yahya Mohamed Abdullah
Magdy Ahmed Abbas Abdelhamid

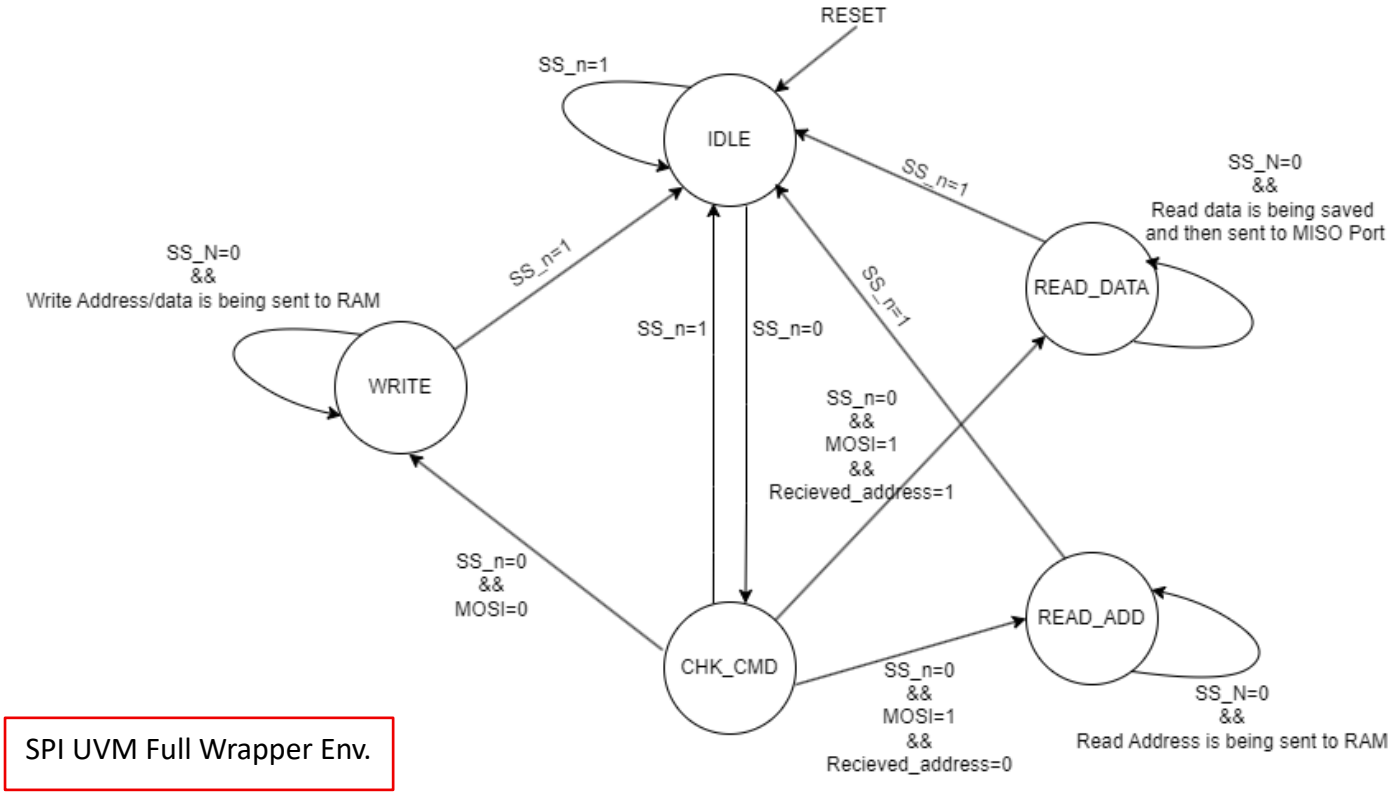


Solution Video: [Drive Link]

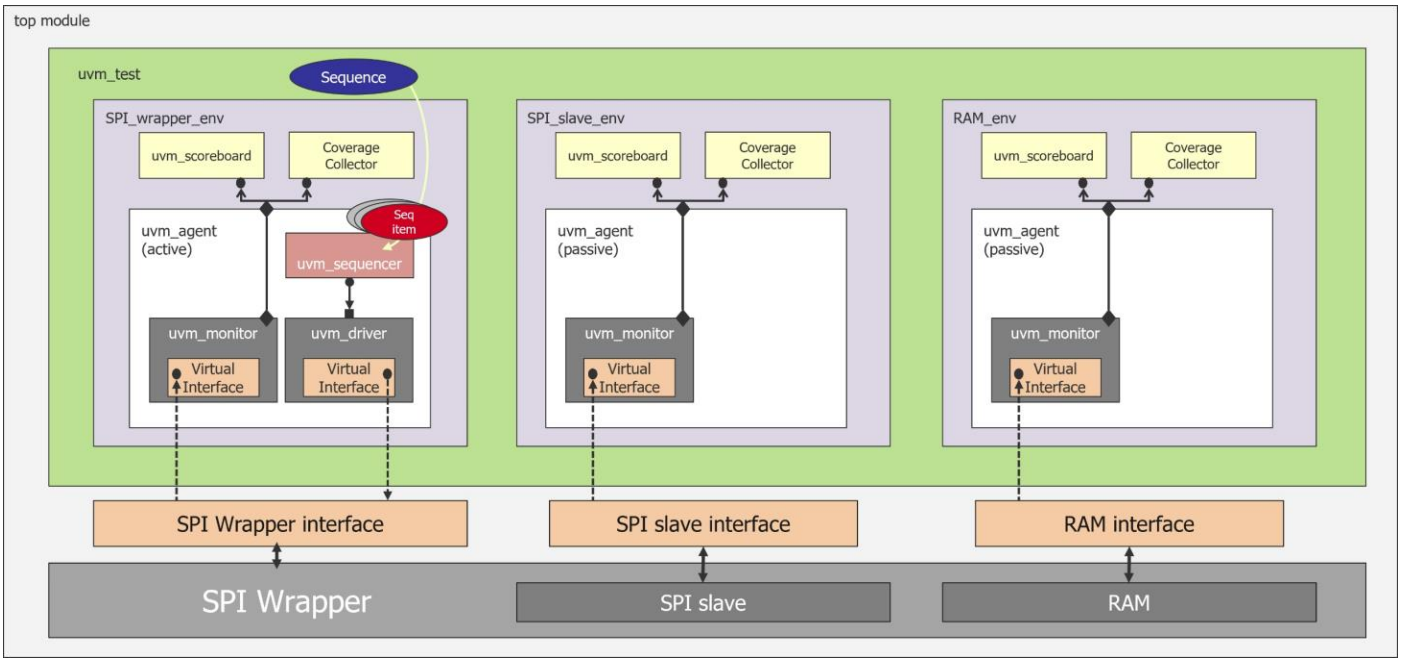
SPI Slave with Single Port RAM

RTL Design

FSM for SPI Interface



SPI UVM Full Wrapper Env.



Lower Modules

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

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

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

reg [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : begin
                    Wr_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b01 : begin
                    MEM[Wr_Addr] <= din[7:0];
                    tx_valid <= 0;
                end
                2'b10 : begin
                    Rd_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b11 : begin
                    dout <= MEM[Rd_Addr];
                    tx_valid <= 1;
                end
                default : tx_valid <= 0;
            endcase
        end
    end
end

endmodule
```

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

Wrapper Module

```
module 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);

endmodule
```

Bug Report

```
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
        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; // Make it inside the case statement to avoid change iff another operation comes
                                                                 // Not when the rx_valid fall
end

endmodule
```

RAM Bug 1

```
// Reading from Write Address
```

RAM Bug 2

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

Slave Bug 1

```
// Replace READ_ADD with READ_DATA
```

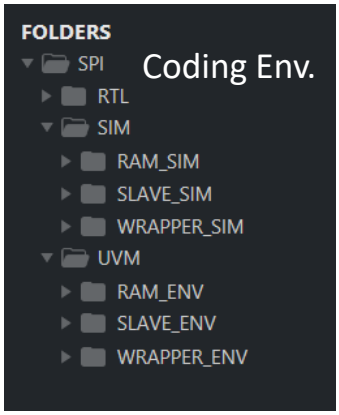
```
always @(posedge clk) begin
    if (~rst_n) begin   // counter must be zero in reset
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        MISO <= 0;
    end
```

Slave Bug 2

```
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 <= 8;
        end
    end
```

Slave Bug 3

```
// counter must be 9 not 8
```



Coding Env.

Part 1

Block Level

SPI Slave with Single Port RAM

Full Slave Environment

Slave Do File

src_files.list

```
../../RTL/SLAVE.sv
../../UVM/SLAVE_ENV/*.*v
```

```
vlib work
vlog -f src_files.list +cover -covercells +define+SIM
vsim -voptargs=+acc work.slave_top -cover -classdebug -uvmcontrol=all +UVM_VERBOSITY=UVM_MEDIUM
run 0
add wave /slave_top/DUT_SLAVE/*
coverage save SLAVE_top.ucdb -onexit -du work.SLAVE
run -all
coverage exclude -src ../../RTL/SLAVE.sv -line 80 -code b
# coverage report -detail -cvg -comments -output SFC_cov_rprt.txt {}
# quit -sim
# vcover report SLAVE_top.ucdb -details -annotate -all -output CC_SVA_cov_rprt.txt
# vcover report SLAVE_top.ucdb -du=SLAVE -recursive -assert -directive -cvg -codeAll -output cov_rprt_summary.txt
```

Slave Coverage

```
Recursive Coverage Report Summary Data by DU

================================================================================
=== Design Unit: work.SLAVE
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Assertions                      10        10         0   100.00%
    Branches                        34        34         0   100.00%
    Conditions                       4         4         0   100.00%
    Directives                      10        10         0   100.00%
    FSM States                       5         5         0   100.00%
    FSM Transitions                  8         8         0   100.00%
    Statements                      39        39         0   100.00%
    Toggles                        122       122         0   100.00%


Total Coverage By Design Unit (filtered view): 100.00%
```

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%
```

Slave Interface

```
interface slave_if (clk);
input clk;
logic       MOSI, rst_n, SS_n, tx_valid;
logic [7:0] tx_data;
logic [9:0] rx_data;
logic       rx_valid, MISO;
logic [9:0] rx_data_gm;
logic       rx_valid_gm, MISO_gm;
endinterface
```

Slave TOP

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

module slave_top ();
  bit clk;

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

  slave_if slave_vif (clk);

  SLAVE DUT_SLAVE (slave_vif.MOSI,slave_vif.MISO,slave_vif.SS_n,clk,slave_vif.rst_n,slave_vif.rx_data,slave_vif.rx_valid,slave_vif.tx_data,slave_vif.tx_valid);
  SLAVE_GM DUT_SLAVE_GM (clk,slave_vif.rst_n,slave_vif.MOSI,slave_vif.SS_n,slave_vif.tx_data,slave_vif.tx_valid,slave_vif.MISO_gm,slave_vif.rx_data_gm,slave_vif.rx_valid_gm);

  bind SLAVE SLAVE_SVA SLAVE_SVA_INST (slave_vif.MOSI,slave_vif.MISO,slave_vif.SS_n,clk,slave_vif.rst_n,slave_vif.rx_data,slave_vif.rx_valid,slave_vif.tx_data,slave_vif.tx_valid);

  initial begin
    uvm_config_db#(virtual slave_if)::set(null,"uvm_test_top","SLAVE_IF",slave_vif);
    run_test("slave_test");
  end
endmodule
```

Slave Config

```
package slave_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_config extends uvm_object;
        `uvm_object_utils(slave_config)
        virtual slave_if slave_vif;
        uvm_active_passive_enum slave_mode;
        function new(string name = "slave_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

Slave Golden

```
module SLAVE_GM (clk,rst_n,MOSI,SS_n,tx_data,tx_valid,MISO,rx_data,rx_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
                    if (received_address)
                        ns = READ_DATA;
                    else
                        ns = READ_ADD;
                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;
                    end
                end
            end
        endcase
    end
end
```

Slave Seq Item

```
package slave_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_seq_item extends uvm_sequence_item;
    `uvm_object_utils(slave_seq_item);

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

         logic [9:0] rx_data_gm;
         logic       rx_valid_gm, MISO_gm;

    // Typedef Enum to be descriptive with the operations to be done
    typedef enum bit [2:0] {WRITE_ADDR = 3'b000, WRITE_DATA = 3'b001, READ_ADDR = 3'b110, READ_DATA = 3'b111} e_state;

    rand bit        MOSI_BUS_RAND[];            // Dynamic Array to be randomized every cycle
         bit        MOSI_BUS[];                 // Dynamic Array to take randomized values only from MOSI_BUS_RAND every start of communication
         int        ss_n_cnt;                   // Counter to indicate Strat and End of communication
         int        count;                      // Counter to Serialize the MOSI_BUS bit by bit in MOSI
         int        WIDTH;                      // Integer to take only two values 13 (For all Operations except Read_Data) and 23 (For Read_Data)

    // States to model FSM concept
    rand e_state    slave_state;                // Randomized every cycle
         e_state    prev_state;                 // Not Randomized to take values of slave_state only on the start of communication


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

    function string convert2string();
        return $sformatf ("%s MOSI = %0b, SS_n = %0b, rst_n = %0d, tx_valid = %0b, tx_data = %0d, rx_data = %0d, rx_valid = %0b, MISO = %0b",
            super.convert2string(),MOSI,SS_n,rst_n,tx_valid,tx_data,rx_data,rx_valid,MISO);
    endfunction

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

    // 1- The reset signal (rst_n) shall be deasserted most of the time.
    constraint rst_n_c {rst_n dist {0:/2, 1:/98};}
```

```
// 2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles
constraint ss_n_c {
    if (!rst_n) {
        SS_n == 1;
    } else {
        // SS_n should be LOW during the data transfer (cycles 1-12 or 1-22) and HIGH only at the end (cycle 13 or 23)
        if (prev_state != READ_DATA) {          // For any operations except READ_DATA
            if (ss_n_cnt inside {[0:12]}) {
                SS_n == 0;
            } else { // ss_n_cnt == 13
                SS_n == 1;
            }
        } else {                                // For READ_DATA only
            if (ss_n_cnt inside {[0:22]}) {
                SS_n == 0;
            } else { // ss_n_cnt == 23
                SS_n == 1;
            }
        }
    }
}

// 3- Declare a randomized array to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI
//    when the SS_n falls are valid combinations only (000, 001, 110, 111)
constraint write_read_c {
    slave_state inside {WRITE_ADDR,WRITE_DATA,READ_ADDR,READ_DATA};
    if (prev_state == WRITE_ADDR)
        slave_state inside {WRITE_ADDR,WRITE_DATA};
    else if (prev_state == WRITE_DATA)
        slave_state dist {WRITE_ADDR:/40, READ_ADDR:/60};
    else if (prev_state == READ_ADDR)
        slave_state inside {WRITE_ADDR,READ_DATA};
    else
        slave_state dist {WRITE_ADDR:/60, READ_ADDR:/40};
}

// Constraint to Serialize MOSI_BUS on MOSI bit by bit
constraint mosi_c {
    if (rst_n) {
        if (ss_n_cnt != WIDTH){  // Serialize During communication only
            MOSI == MOSI_BUS[count];
        }
    }
}

// 4- The tx_valid signal to be high in case of read data
constraint tx_valid_c {
    if (rst_n && !SS_n) {
        if ((prev_state == READ_DATA) && (ss_n_cnt > 12))
            tx_valid == 1;
        else
            tx_valid == 0;
    }
}
```

```
    function void pre_randomize();
        if (!rst_n) begin
            count = 0;
        end
        else begin
            if (ss_n_cnt == 0) begin
                prev_state = slave_state;           // Will be updated when ss_n_cnt == 0 (start of communication) (after SS_n asserted and de-asserted)
                count = 0;                          // Serialize count git inialized with zero to start communication

                if (prev_state == READ_DATA) begin   // 23 Cycles For Read Data
                    WIDTH = 23;
                    MOSI_BUS = new[WIDTH];
                    MOSI_BUS_RAND = new[WIDTH];
                end
                else begin                          // 13 Cycles For any other state
                    WIDTH = 13;
                    MOSI_BUS = new[WIDTH];
                    MOSI_BUS_RAND = new[WIDTH];
                end
            end
            else begin
                count = count + 1;                  // Increment serializer counter during running of simulation
            end
        end
       // $display("ss_n_cnt = %0d, count = %0d, MOSI_BUS = %0p, MOSI = %0b, prev_state = %03b, slave_state = %03b",ss_n_cnt,count,MOSI_BUS,MOSI,prev_state,slave_state);
    endfunction

    function void post_randomize();
        if (!rst_n) begin
            ss_n_cnt = 0;
        end else begin
            if (ss_n_cnt == 0) begin          // At the Start of Simulation
                MOSI_BUS = MOSI_BUS_RAND;     // 1- Take Randomized values from MOSI_BUS_RAND to be assigned tp MOSI_BUS array
                for (int i=0; i<3; i++)       // 2- Loop for the 3 MSB to be like prev_state, which takes value of the required operation to be done
                    MOSI_BUS[i] = prev_state[2-i];
            end

            // Handle SS_n counter to control start and end of communication
            if ((ss_n_cnt != 13) && (prev_state != READ_DATA))
                ss_n_cnt += 1;
            else if ((ss_n_cnt != 23) && (prev_state == READ_DATA))
                ss_n_cnt += 1;
            else
                ss_n_cnt = 0;
        end
    endfunction
endclass
endpackage
```

Slave Sequence

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

class reset_sequence extends uvm_sequence #(slave_seq_item);
    `uvm_object_utils(reset_sequence);
    slave_seq_item rst_seq_item;

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

    task body();
        rst_seq_item = slave_seq_item::type_id::create("rst_seq_item");
        start_item(rst_seq_item);
        rst_seq_item.rst_n = 0;
        rst_seq_item.SS_n = 0;
        rst_seq_item.MOSI = 0;
        rst_seq_item.tx_valid = 0;
        rst_seq_item.tx_data = 0;
        finish_item(rst_seq_item);
    endtask
endclass

class main_sequence extends uvm_sequence #(slave_seq_item);
    `uvm_object_utils(main_sequence);
    slave_seq_item main_seq_item;

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

    task body();
        main_seq_item = slave_seq_item::type_id::create("main_seq_item");
        repeat(10000) begin
            start_item(main_seq_item);
            assert(main_seq_item.randomize());
            finish_item(main_seq_item);
        end
    endtask
endclass

endpackage
```

Slave Sequencer

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

class slave_sequencer extends uvm_sequencer #(slave_seq_item);
    `uvm_component_utils(slave_sequencer);

    function new(string name = "slave_sequencer", uvm_component parent = null);
        super.new(name,parent);
    endfunction
endclass
endpackage
```

Slave Driver

```
package slave_driver_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_driver extends uvm_driver #(slave_seq_item);
  `uvm_component_utils(slave_driver)
  virtual slave_if slave_vif;
  slave_seq_item stim_seq_item;

  function new (string name = "slave_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 = slave_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
        slave_vif.MOSI = stim_seq_item.MOSI;
        slave_vif.rst_n = stim_seq_item.rst_n;
        slave_vif.SS_n = stim_seq_item.SS_n;
        slave_vif.tx_valid = stim_seq_item.tx_valid;
        slave_vif.tx_data = stim_seq_item.tx_data;
        @(negedge slave_vif.clk);
        seq_item_port.item_done();
        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

Slave Monitor

```
package slave_monitor_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_monitor extends uvm_monitor;
    `uvm_component_utils(slave_monitor)
    virtual slave_if slave_vif;
    slave_seq_item rsp_seq_item;
    uvm_analysis_port #(slave_seq_item) mon_ap;

    function new(string name = "slave_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 = slave_seq_item::type_id::create("rsp_seq_item");
            @(negedge slave_vif.clk);
            rsp_seq_item.MOSI = slave_vif.MOSI;
            rsp_seq_item.rst_n = slave_vif.rst_n;
            rsp_seq_item.SS_n = slave_vif.SS_n;
            rsp_seq_item.tx_valid = slave_vif.tx_valid;
            rsp_seq_item.tx_data = slave_vif.tx_data;
            rsp_seq_item.rx_data = slave_vif.rx_data;
            rsp_seq_item.rx_valid = slave_vif.rx_valid;
            rsp_seq_item.MISO = slave_vif.MISO;
            rsp_seq_item.rx_data_gm = slave_vif.rx_data_gm;
            rsp_seq_item.rx_valid_gm = slave_vif.rx_valid_gm;
            rsp_seq_item.MISO_gm = slave_vif.MISO_gm;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

Slave Agent

```
package slave_agent_pkg;
import slave_config_pkg::*;
import slave_driver_pkg::*;
import slave_monitor_pkg::*;
import slave_sequencer_pkg::*;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_agent extends uvm_agent;
`uvm_component_utils(slave_agent)
slave_sequencer sqr;
slave_monitor mon;
slave_driver drv;
slave_config slave_cfg;
uvm_analysis_port #(slave_seq_item) agt_ap;

function new (string name = "slave_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 #(slave_config)::get(this, "", "CFG_SLAVE",slave_cfg)) begin
        `uvm_fatal("build_phase","Unable to get configuration object")
    end
    if (slave_cfg.slave_mode == UVM_ACTIVE) begin
        sqr = slave_sequencer::type_id::create("sqr",this);
        drv = slave_driver::type_id::create("drv",this);
    end
    mon = slave_monitor::type_id::create("mon",this);
    agt_ap = new("agt_ap",this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    if (slave_cfg.slave_mode == UVM_ACTIVE) begin
        drv.slave_vif = slave_cfg.slave_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
    end
    mon.slave_vif = slave_cfg.slave_vif;
    mon.mon_ap.connect(agt_ap);
endfunction

endclass
endpackage
```

Slave Cov. Collector

```
package slave_coverage_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_coverage extends uvm_component;
        `uvm_component_utils(slave_coverage)
        uvm_analysis_port #(slave_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(slave_seq_item) cov_fifo;
        slave_seq_item seq_item_cov;

        covergroup cvr_grp;

            // 1- Add coverpoints on rx_data[9:8] to take all possible values and all possible transitions
            rx_data_cp : coverpoint seq_item_cov.rx_data[9:8]{
                bins wr_addr = {2'b00};
                bins wr_data = {2'b01};
                bins rd_addr = {2'b10};
                bins rd_data = {2'b11};
                bins write_trans = (2'b00 => 2'b01);
                bins read_trans  = (2'b10 => 2'b11);
            }
            // 2- Add coverpoints on SS_n to capture:
            // A. Check full transaction duration: 1 → 0 [*13] → 1 for normal operations.
            // B. Check extended transaction: 1 → 0 [*23] → 1 for READ_DATA
            SS_n_cp : coverpoint seq_item_cov.SS_n {
                bins ss_n_trans = (1=>0[*13]=>1);
                bins ss_n_trans_read_data = (1=>0[*23]=>1);
                bins ss_n_start_trans = (1=>0);
                bins ss_n_end_trans = (0=>1);
            }
            // 3- Add coverpoints on MOSI to validate correct transitions:
            // A. 000 (Write Address)
            // B. 001 (Write Data)
            // C. 110 (Read Address)
            // D. 111 (Read Data)
            MOSI_cp: coverpoint seq_item_cov.MOSI{
                bins wr_addr_trans = (0=>0=>0);
                bins wr_data_trans = (0=>0=>1);
                bins rd_addr_trans = (1=>1=>0);
                bins rd_data_trans = (1=>1=>1);
            }
            // 4- Cross coverage between SS_n and MOSI bins (Exclude irrelevant bins to focus on legal operation scenarios)
            MISO_cp: coverpoint seq_item_cov.MISO;
            ss_n_op_cross: cross SS_n_cp, MOSI_cp{
                ignore_bins bin_1 = binsof(SS_n_cp.ss_n_trans);
                ignore_bins bin_2 = binsof(SS_n_cp.ss_n_trans_read_data);
                ignore_bins bin_3 = binsof(SS_n_cp.ss_n_end_trans);
            }
            rstn_cp : coverpoint seq_item_cov.rst_n;
            rx_valid_cp : coverpoint seq_item_cov.rx_valid;
            tx_valid_cp : coverpoint seq_item_cov.tx_valid;
        endgroup

        function new (string name = "slave_coverage", uvm_component parent = null);
            super.new(name,parent);
            cvr_grp = 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);
                cvr_grp.sample();
            end
        endtask
    endclass
```

Slave Scoreboard

```
package slave_scoreboard_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(slave_scoreboard);
    uvm_analysis_export #(slave_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(slave_seq_item) sb_fifo;
    slave_seq_item seq_item_sb;

    int error_count;
    int correct_count;

    function new(string name = "slave_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);
                if ((seq_item_sb.rx_data != seq_item_sb.rx_data_gm) || (seq_item_sb.rx_valid != seq_item_sb.rx_valid_gm) || (seq_item_sb.MISO != seq_item_sb.MISO_gm)) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT: %s while the refecrence rx_data: %0d, rx_valid: %0b, MISO: %0b"
                        ,seq_item_sb.convert2string(),seq_item_sb.rx_data_gm,seq_item_sb.rx_valid_gm,seq_item_sb.MISO_gm));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct slave_out : %s",seq_item_sb.convert2string()),UVM_HIGH);
                     correct_count++;
                end
            end
    endtask

    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Slave: Total successful counts : %0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Slave: Total failed counts : %0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

Slave Environment

```
package slave_env_pkg;
import slave_agent_pkg::*;
import slave_scoreboard_pkg::*;
import slave_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class slave_env extends uvm_env;
  `uvm_component_utils(slave_env)
  slave_agent agt;
  slave_scoreboard sb;
  slave_coverage cov;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = slave_agent::type_id::create("agt",this);
    sb = slave_scoreboard::type_id::create("sb",this);
    cov = slave_coverage::type_id::create("cov",this);
  endfunction

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

Slave Test

```
package slave_test_pkg;
import slave_config_pkg::*;
import slave_env_pkg::*;
import slave_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_test extends uvm_test;
  `uvm_component_utils (slave_test)
  slave_env env_slave;
  slave_config slave_cfg;
  reset_sequence reset_seq;
  main_sequence main_seq;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env_slave = slave_env::type_id::create("env_slave",this);
    slave_cfg = slave_config::type_id::create("slave_cfg");
    reset_seq = reset_sequence::type_id::create("reset_seq");
    main_seq = main_sequence::type_id::create("main_seq");

    if (!uvm_config_db#(virtual slave_if)::get(this,"","SLAVE_IF",slave_cfg.slave_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the slave from the uvm_config_db")

    slave_cfg.slave_mode = UVM_ACTIVE;

    uvm_config_db#(slave_config)::set(this,"*","CFG_SLAVE",slave_cfg);
  endfunction

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","Reset Asserted",UVM_LOW)
    reset_seq.start(env_slave.agt.sqr);
    `uvm_info("run_phase","Reset Deasserted",UVM_LOW)

    `uvm_info("run_phase","Slave Main Stimulus Generation Started",UVM_LOW)
    main_seq.start(env_slave.agt.sqr);
    `uvm_info("run_phase","Slave Main Stimulus Generation Ended",UVM_LOW)

    phase.drop_objection(this);
  endtask
endclass

endpackage
```

SLAVE SVA

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

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

sequence write_add_seq;
    (!SS_n ##1 !MOSI ##1 !MOSI ##1 !MOSI);
endsequence

sequence write_data_seq;
    (!SS_n ##1 !MOSI ##1 !MOSI ##1 MOSI);
endsequence

sequence read_add_seq;
    (!SS_n ##1 MOSI ##1 MOSI ##1 !MOSI);
endsequence

sequence read_data_seq;
    (!SS_n ##1 MOSI ##1 MOSI ##1 MOSI);
endsequence

// 1- An assertion ensures that whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.
property rstn_property;
    @(posedge clk) !rst_n |=> (!MISO && !rx_valid && ~rx_data);
endproperty

// 2- An assertion checks that after any valid command sequence (write_add_seq (000), write_data_seq (001), read_add_seq(110), or read_data_seq(111)),
//    the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.
property tx_valid_property;
    @(posedge clk) disable iff(!rst_n) (!SS_n && tx_valid) |=> $fell(tx_valid) [->1];
endproperty

property rx_valid_property;
    @(posedge clk) disable iff(!rst_n) (write_add_seq or write_data_seq or read_add_seq or read_data_seq) |-> ##10 (rx_valid && $rose(SS_n)[->1]);
endproperty

rstn_assertion: assert property (rstn_property);
rstn_cover:     cover  property (rstn_property);

tx_valid_assertion: assert property (tx_valid_property);
tx_valid_cover:     cover  property (tx_valid_property);

rx_valid_assertion: assert property (rx_valid_property);
rx_valid_cover:     cover  property (rx_valid_property);

endmodule
```

```
// 3- Add the following assertions in the design to check correct FSM transitions
// • IDLE → CHK_CMD
// • CHK_CMD → WRITE or READ_ADD or READ_DATA
// • WRITE → IDLE
// • READ_ADD → IDLE
// • READ_DATA → IDLE

`ifdef SIM
    property FSM_Trans1;
        @(posedge clk) disable iff(!rst_n) (cs == IDLE && !SS_n) |=> (cs == CHK_CMD);
    endproperty

    property FSM_Trans2;
        @(posedge clk) disable iff(!rst_n) (cs == CHK_CMD && !SS_n && !MOSI) |=> (cs == WRITE);
    endproperty

    property FSM_Trans3;
        @(posedge clk) disable iff(!rst_n) (cs == CHK_CMD && !SS_n && MOSI && !received_address) |=> (cs == READ_ADD);
    endproperty

    property FSM_Trans4;
        @(posedge clk) disable iff(!rst_n) (cs == CHK_CMD && !SS_n && MOSI && received_address) |=> (cs == READ_DATA);
    endproperty

    property FSM_Trans5;
        @(posedge clk) disable iff(!rst_n) (cs == WRITE && SS_n) |=> (cs == IDLE);
    endproperty

    property FSM_Trans6;
        @(posedge clk) disable iff(!rst_n) (cs == READ_ADD && SS_n) |=> (cs == IDLE);
    endproperty

    property FSM_Trans7;
        @(posedge clk) disable iff(!rst_n) (cs == READ_DATA && SS_n) |=> (cs == IDLE);
    endproperty

    FSM_Trans_asser1: assert property (FSM_Trans1);
    FSM_Trans_asser2: assert property (FSM_Trans2);
    FSM_Trans_asser3: assert property (FSM_Trans3);
    FSM_Trans_asser4: assert property (FSM_Trans4);
    FSM_Trans_asser5: assert property (FSM_Trans5);
    FSM_Trans_asser6: assert property (FSM_Trans6);
    FSM_Trans_asser7: assert property (FSM_Trans7);

    FSM_Trans_cover1: cover property (FSM_Trans1);
    FSM_Trans_cover2: cover property (FSM_Trans2);
    FSM_Trans_cover3: cover property (FSM_Trans3);
    FSM_Trans_cover4: cover property (FSM_Trans4);
    FSM_Trans_cover5: cover property (FSM_Trans5);
    FSM_Trans_cover6: cover property (FSM_Trans6);
    FSM_Trans_cover7: cover property (FSM_Trans7);

`endif
```

Part 2

Block Level

SPI Slave with Single Port RAM

Full Ram Environment

RAM Do File

src_files.list

```
../../RTL/RAM.v
../../UVM/RAM_ENV/*.*v
```

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.ram_top -cover -classdebug -uvmcontrol=all +UVM_VERBOSITY=UVM_MEDIUM
run 0
add wave /ram_top/DUT_RAM/*
coverage save RAM_top.ucdb -onexit -du work.RAM
run -all
coverage exclude -src ../../RTL/RAM.v -line 28 -code s
coverage exclude -src ../../RTL/RAM.v -line 28 -code b
# coverage report -detail -cvg -comments -output SFC_cov_rprt.txt {}
# quit -sim
# vcover report RAM_top.ucdb -details -annotate -all -output CC_SVA_cov_rprt.txt
# vcover report RAM_top.ucdb -du=RAM -recursive -assert -directive -cvg -codeAll -output cov_rprt_summary.txt
```

RAM Coverage

```
Recursive Coverage Report Summary Data by DU

================================================================================
=== Design Unit: work.RAM
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Assertions                       5         5         0   100.00%
    Branches                         8         8         0   100.00%
    Directives                       5         5         0   100.00%
    Expressions                      3         3         0   100.00%
    Statements                      10        10         0   100.00%
    Toggles                        120       120         0   100.00%


Total Coverage By Design Unit (filtered view): 100.00%
```

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

Total Coverage By Instance (filtered view): 100.00%
```

RAM Interface

```
interface ram_if (clk);
input clk;
logic [9:0] din;
logic       rst_n, rx_valid;
logic [7:0] dout;
logic       tx_valid;
logic [7:0] dout_gm;
logic       tx_valid_gm;
endinterface
```

RAM TOP

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

module ram_top ();
  bit clk;

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

  ram_if ram_vif (clk);

  RAM DUT_RAM (ram_vif.din,clk,ram_vif.rst_n,ram_vif.rx_valid,ram_vif.dout,ram_vif.tx_valid);
  RAM_GM DUT_RAM_GM (ram_vif.din,clk,ram_vif.rst_n,ram_vif.rx_valid,ram_vif.dout_gm,ram_vif.tx_valid_gm);

  bind RAM RAM_SVA RAM_SVA_INST (ram_vif.din,clk,ram_vif.rst_n,ram_vif.rx_valid,ram_vif.dout,ram_vif.tx_valid);

  initial begin
    uvm_config_db#(virtual ram_if)::set(null,"uvm_test_top","RAM_IF",ram_vif);
    run_test("ram_test");
  end

  initial begin
    $readmemh ("RAM.dat",DUT_RAM.MEM);
  end
endmodule
```

RAM Config

```
package ram_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class ram_config extends uvm_object;
        `uvm_object_utils(ram_config)
        virtual ram_if ram_vif;
        uvm_active_passive_enum ram_mode;
        function new(string name = "ram_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

RAM Golden

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

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

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

reg [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : begin
                    Wr_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b01 : begin
                    MEM[Wr_Addr] <= din[7:0];
                    tx_valid <= 0;
                end
                2'b10 : begin
                    Rd_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b11 : begin
                    dout <= MEM[Rd_Addr];
                    tx_valid <= 1;
                end
                default : tx_valid <= 0;
            endcase
        end
    end
end

endmodule
```

RAM Seq Item

```
package ram_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_seq_item extends uvm_sequence_item;
    `uvm_object_utils(ram_seq_item);

    typedef enum bit [1:0] {WRITE_ADDR = 2'b00, WRITE_DATA = 2'b01, READ_ADDR = 2'b10, READ_DATA = 2'b11} e_state;

    rand  logic [9:0] din;
    rand  logic       rx_valid;
    rand  logic       rst_n;
          logic [7:0] dout;
          logic       tx_valid;
          logic [7:0] dout_gm;
          logic       tx_valid_gm;

    rand  e_state mem_state;
          e_state prev_state;

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

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

    function string convert2string_stimulus();
        return $sformatf ("din = %0b, rx_valid = %0d, rst_n = %0d",din,rx_valid,rst_n);
    endfunction

    // 1- The reset signal (rst_n) shall be deasserted most of the time.
    constraint rst_n_c {rst_n dist {0:/2, 1:/98};}

    // 2- The rx_valid signal shall be asserted most of time.
    constraint rx_valid_c {rx_valid dist {0:/10, 1:/90};}

    // 3- For a write-only sequence, every Write Address operation shall always be followed by either Write Address or Write Data operation.
    constraint write_c {
        if (rst_n && rx_valid) {
            if (prev_state == WRITE_ADDR)
                mem_state inside {WRITE_ADDR, WRITE_DATA};
            else
                mem_state == WRITE_DATA;
        }
    }

    // 4- For a read-only sequence, every Read Address operation shall always be followed by Read Data.
    //    After a Read Data operation shall always be followed by Read Address.
    constraint read_c {
        if (rst_n && rx_valid) {
            if (prev_state == READ_ADDR)
                mem_state == READ_DATA;
            else
                mem_state == READ_ADDR;
        }
    }
```

```
    // 5- For a randomized read/write sequence
    // A. After Write Address There will be followed by either Write Address or Write Data
    // B. After Write Data There will be distribution to be 60% Read Address and 40% Write Address
    // C. After Read Address There will be Read Data
    // D. After Read Data There will be distribution to be 60% Write Address and 40% Read Address
    constraint write_read_c {
        if (rst_n && rx_valid) {
            if (prev_state == WRITE_ADDR)
              mem_state inside {WRITE_ADDR, WRITE_DATA};
            else if (prev_state == WRITE_DATA)
              mem_state dist {READ_ADDR :/ 60, WRITE_ADDR :/ 40};
            else if (prev_state == READ_ADDR)
              mem_state == READ_DATA;
            else if (prev_state == READ_DATA)
              mem_state dist {WRITE_ADDR :/ 60, READ_ADDR :/ 40};
            else
              mem_state inside {WRITE_ADDR, READ_ADDR};
        }
    }

    constraint din_c {din[9:8] == mem_state;}

    function void post_randomize();
        if (rst_n && rx_valid)
            prev_state = mem_state;
    endfunction
endclass
endpackage
```

RAM Sequence

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

class reset_sequence extends uvm_sequence #(ram_seq_item);
    `uvm_object_utils(reset_sequence);
    ram_seq_item rst_seq_item;

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

    task body();
        rst_seq_item = ram_seq_item::type_id::create("rst_seq_item");
        start_item(rst_seq_item);
        rst_seq_item.rst_n = 0;
        rst_seq_item.din = 0;
        rst_seq_item.rx_valid = 0;
        finish_item(rst_seq_item);
    endtask
endclass

class write_sequence extends uvm_sequence #(ram_seq_item);
    `uvm_object_utils(write_sequence);
    ram_seq_item write_seq_item;

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

    task body();
        write_seq_item = ram_seq_item::type_id::create("write_seq_item");
        repeat(10000)begin
            write_seq_item.write_c.constraint_mode(1);
            write_seq_item.read_c.constraint_mode(0);
            write_seq_item.write_read_c.constraint_mode(0);
            start_item(write_seq_item);
            assert(write_seq_item.randomize());
            finish_item(write_seq_item);
        end
    endtask
endclass
```

```
class read_sequence extends uvm_sequence #(ram_seq_item);
    `uvm_object_utils(read_sequence);
    ram_seq_item read_seq_item;

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

    task body();
        read_seq_item = ram_seq_item::type_id::create("read_seq_item");
        repeat(10000)begin
            read_seq_item.write_c.constraint_mode(0);
            read_seq_item.read_c.constraint_mode(1);
            read_seq_item.write_read_c.constraint_mode(0);
            start_item(read_seq_item);
            assert(read_seq_item.randomize());
            finish_item(read_seq_item);
        end
    endtask
endclass

class write_read_sequence extends uvm_sequence #(ram_seq_item);
    `uvm_object_utils(write_read_sequence);
    ram_seq_item wr_rd_seq_item;

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

    task body();
        wr_rd_seq_item = ram_seq_item::type_id::create("wr_rd_seq_item");
        repeat(10000)begin
            wr_rd_seq_item.write_c.constraint_mode(0);
            wr_rd_seq_item.read_c.constraint_mode(0);
            wr_rd_seq_item.write_read_c.constraint_mode(1);
            start_item(wr_rd_seq_item);
            assert(wr_rd_seq_item.randomize());
            finish_item(wr_rd_seq_item);
        end
    endtask
endclass
endpackage
```

RAM Sequencer

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

class ram_sequencer extends uvm_sequencer #(ram_seq_item);
    `uvm_component_utils(ram_sequencer);

    function new(string name = "ram_sequencer", uvm_component parent = null);
        super.new(name,parent);
    endfunction
endclass
endpackage
```

RAM Driver

```
package ram_driver_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_driver extends uvm_driver #(ram_seq_item);
  `uvm_component_utils(ram_driver)
  virtual ram_if ram_vif;
  ram_seq_item stim_seq_item;

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

  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
        stim_seq_item = ram_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
        ram_vif.din = stim_seq_item.din;
        ram_vif.rst_n = stim_seq_item.rst_n;
        ram_vif.rx_valid = stim_seq_item.rx_valid;
        @(negedge ram_vif.clk);
        seq_item_port.item_done();
        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

RAM Monitor

```
package ram_monitor_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_monitor extends uvm_monitor;
    `uvm_component_utils(ram_monitor)
    virtual ram_if ram_vif;
    ram_seq_item rsp_seq_item;
    uvm_analysis_port #(ram_seq_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = ram_seq_item::type_id::create("rsp_seq_item");
            @(negedge ram_vif.clk);
            rsp_seq_item.din = ram_vif.din;
            rsp_seq_item.rx_valid = ram_vif.rx_valid;
            rsp_seq_item.rst_n = ram_vif.rst_n;
            rsp_seq_item.dout = ram_vif.dout;
            rsp_seq_item.tx_valid = ram_vif.tx_valid;
            rsp_seq_item.dout_gm = ram_vif.dout_gm;
            rsp_seq_item.tx_valid_gm = ram_vif.tx_valid_gm;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

RAM Agent

```
package ram_agent_pkg;
import ram_config_pkg::*;
import ram_driver_pkg::*;
import ram_monitor_pkg::*;
import ram_sequencer_pkg::*;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_agent extends uvm_agent;
`uvm_component_utils(ram_agent)
ram_sequencer sqr;
ram_monitor mon;
ram_driver drv;
ram_config ram_cfg;
uvm_analysis_port #(ram_seq_item) agt_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, "", "CFG_RAM",ram_cfg)) begin
        `uvm_fatal("build_phase","Unable to get configuration object")
    end
    if (ram_cfg.ram_mode == UVM_ACTIVE) begin
        sqr = ram_sequencer::type_id::create("sqr",this);
        drv = ram_driver::type_id::create("drv",this);
    end
    mon = ram_monitor::type_id::create("mon",this);
    agt_ap = new("agt_ap",this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    if (ram_cfg.ram_mode == UVM_ACTIVE) begin
        drv.ram_vif = ram_cfg.ram_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
    end
    mon.ram_vif = ram_cfg.ram_vif;
    mon.mon_ap.connect(agt_ap);
endfunction

endclass
endpackage
```

RAM Cov. Collector

```
package ram_coverage_pkg;
    import ram_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class ram_coverage extends uvm_component;
        `uvm_component_utils(ram_coverage)
        uvm_analysis_port #(ram_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(ram_seq_item) cov_fifo;
        ram_seq_item seq_item_cov;

        covergroup cvr_grp;

            // 1- Write Coverpoint to check transaction ordering for din[9:8]:
            // • Check din[9:8] takes 4 possible values
            // • Check write data after write address
            // • Check read data after read address
            // • Check write address => write data => read address => read data
            ram_order_cp : coverpoint seq_item_cov.din[9:8] {
                bins write_add        = {2'b00};
                bins write_data       = {2'b01};
                bins read_add         = {2'b10};
                bins read_data        = {2'b11};
                bins write_only_trans = (2'b00 => 2'b01);
                bins read_only_trans  = (2'b10 => 2'b11);
                bins write_then_read  = (2'b00 => 2'b01 => 2'b10 => 2'b11);
            }
            rstn_cp : coverpoint seq_item_cov.rst_n;
            rx_valid_cp : coverpoint seq_item_cov.rx_valid;
            tx_valid_cp : coverpoint seq_item_cov.tx_valid;

            // 2- Cross coverage:
            // • Between all bins of din[9:8] and rx_valid signal when it is high
            rx_valid_cross_ram_order : cross ram_order_cp, rx_valid_cp {
                ignore_bins ign_bin_1 = binsof(rx_valid_cp) intersect {0};
                ignore_bins ign_bin_2 = binsof(ram_order_cp.write_only_trans);
                ignore_bins ign_bin_3 = binsof(ram_order_cp.read_only_trans);
                ignore_bins ign_bin_4 = binsof(ram_order_cp.write_then_read);
            }

            // • Between din[9:8] when it equals read data and tx_valid when it is high
            tx_valid_cross_ram_order : cross ram_order_cp, tx_valid_cp {
                bins read_data = binsof(ram_order_cp.read_data) && binsof(tx_valid_cp) intersect {1};
                option.cross_auto_bin_max = 0;
            }
        endgroup

        function new (string name = "ram_coverage", uvm_component parent = null);
            super.new(name,parent);
            cvr_grp = 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);
                cvr_grp.sample();
            end
        endtask
    endclass
endpackage
```

RAM Scoreboard

```
package ram_scoreboard_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(ram_scoreboard);
    uvm_analysis_export #(ram_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(ram_seq_item) sb_fifo;
    ram_seq_item seq_item_sb;

    int error_count;
    int correct_count;

    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);
        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);
                if ((seq_item_sb.dout !== seq_item_sb.dout_gm) || (seq_item_sb.tx_valid !== seq_item_sb.tx_valid_gm)) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT: %s while the refecrence dout:
                                %0d, tx_valid: %0d",seq_item_sb.convert2string(),seq_item_sb.dout_gm,seq_item_sb.tx_valid_gm));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct ram_out : %s",seq_item_sb.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            end
    endtask

    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Ram: Total successful counts : %0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Ram: Total failed counts : %0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

RAM Environment

```
package ram_env_pkg;
import ram_agent_pkg::*;
import ram_scoreboard_pkg::*;
import ram_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class ram_env extends uvm_env;
  `uvm_component_utils(ram_env)
  ram_agent agt;
  ram_scoreboard sb;
  ram_coverage cov;

  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);
    agt = ram_agent::type_id::create("agt",this);
    sb = ram_scoreboard::type_id::create("sb",this);
    cov = ram_coverage::type_id::create("cov",this);
  endfunction

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

RAM Test

```
package ram_test_pkg;
import ram_config_pkg::*;
import ram_env_pkg::*;
import ram_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_test extends uvm_test;
  `uvm_component_utils (ram_test)
  ram_env env_ram;
  ram_config ram_cfg;
  reset_sequence reset_seq;
  write_sequence write_seq;
  read_sequence read_seq;
  write_read_sequence write_read_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 = ram_env::type_id::create("env_ram",this);
    ram_cfg = ram_config::type_id::create("ram_cfg");
    reset_seq = reset_sequence::type_id::create("reset_seq");
    write_seq = write_sequence::type_id::create("write_seq");
    read_seq = read_sequence::type_id::create("read_seq");
    write_read_seq = write_read_sequence::type_id::create("write_read_seq");

    if (!uvm_config_db#(virtual ram_if)::get(this,"","RAM_IF",ram_cfg.ram_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the ram from the uvm_config_db")

    ram_cfg.ram_mode = UVM_ACTIVE;

    uvm_config_db#(ram_config)::set(this,"*","CFG_RAM",ram_cfg);
  endfunction

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","Reset Asserted",UVM_LOW)
    reset_seq.start(env_ram.agt.sqr);
    `uvm_info("run_phase","Reset Deasserted",UVM_LOW)

    `uvm_info("run_phase","RAM Write Stimulus Generation Started",UVM_LOW)
    write_seq.start(env_ram.agt.sqr);
    `uvm_info("run_phase","RAM Write Stimulus Generation Ended",UVM_LOW)

    `uvm_info("run_phase","RAM Read Stimulus Generation Started",UVM_LOW)
    read_seq.start(env_ram.agt.sqr);
    `uvm_info("run_phase","RAM Read Stimulus Generation Ended",UVM_LOW)

    `uvm_info("run_phase","RAM Write & Read Stimulus Generation Started",UVM_LOW)
    write_read_seq.start(env_ram.agt.sqr);
    `uvm_info("run_phase","RAM Write & Read Stimulus Generation Ended",UVM_LOW)
    phase.drop_objection(this);
  endtask
endclass

endpackage
```

RAM SVA

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

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

sequence write_add_seq;
    ((din[9:8] == 2'b00) && rx_valid);
endsequence

sequence write_data_seq;
    ((din[9:8] == 2'b01) && rx_valid);
endsequence

sequence read_add_seq;
    ((din[9:8] == 2'b10) && rx_valid);
endsequence

sequence read_data_seq;
    ((din[9:8] == 2'b11) && rx_valid);
endsequence

// 1- An assertion ensures that whenever reset is asserted, the output signals (tx_valid and dout) are low
property rstn_property;
    @(posedge clk) !rst_n |=> (!tx_valid && ~dout);
endproperty

// 2- An assertion checks that during address or data input phases (write_add_seq, write_data_seq, read_add_seq),
//    the tx_valid signal must remain deasserted.
property not_tx_valid_property;
    @(posedge clk) disable iff(!rst_n) (write_add_seq or write_data_seq or read_add_seq) |=> !tx_valid;
endproperty

// 3- An assertion checks that after a read_data_seq occurs, the tx_valid signal must rise to indicate valid output
//    and after it rises by one clock cycle, it should eventually fall.
property tx_valid_fall_after_rise;
    @(posedge clk) disable iff(!rst_n) (read_data_seq) |=> $rose(tx_valid) |=> ($fell(tx_valid) [->1]);
endproperty

// 4- An assertion checks that every Write Address operation must be eventually followed by a Write Data operation.
property write_sequence;
    @(posedge clk) disable iff(!rst_n) write_add_seq |=> ((din[9:8] == 2'b01) && rx_valid) [->1];
endproperty

// 5- An assertion checks that every Read Address operation must be eventually followed by a Read Data operation.
property read_sequence;
    @(posedge clk) disable iff(!rst_n) read_add_seq |=> ((din[9:8] == 2'b11) && rx_valid) [->1];
endproperty

rstn_assertion: assert property (rstn_property);
rstn_cover:     cover  property (rstn_property);

not_tx_valid_assertion: assert property (not_tx_valid_property);
not_tx_valid_cover:     cover  property (not_tx_valid_property);

tx_valid_fall_after_rise_assertion: assert property (tx_valid_fall_after_rise);
tx_valid_fall_after_rise_cover:     cover  property (tx_valid_fall_after_rise);

write_seq_assertion: assert property (write_sequence);
write_seq_cover:     cover  property (write_sequence);

read_seq_assertion: assert property (read_sequence);
read_seq_cover:     cover  property (read_sequence);

endmodule
```

Part 3

System Level

SPI Slave with Single Port RAM

Wrapper Environment

Wrapper Do File

src_files.list

```
../../RTL/SLAVE.sv
../../RTL/RAM.v
../../RTL/WRAPPER.v
../../UVM/WRAPPER_ENV/*.*v
```

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.wrapper_top -cover -classdebug -uvmcontrol=all +UVM_VERBOSITY=UVM_MEDIUM
run 0
add wave /wrapper_top/DUT_WRAPPER/*
coverage save WRAPPER_top.ucdb -onexit -du work.WRAPPER
run -all
coverage exclude -src ../../RTL/RAM.v -line 28 -code s
coverage exclude -src ../../RTL/RAM.v -line 28 -code b
coverage exclude -src ../../RTL/SLAVE.sv -line 114 -code s
coverage exclude -src ../../RTL/SLAVE.sv -line 80 -code b
coverage exclude -src ../../RTL/SLAVE.sv -line 113 -code b
coverage exclude -src ../../RTL/SLAVE.sv -line 109 -code c
coverage report -detail -cvg -comments -output SFC_cov_rprt.txt {}
quit -sim
vcover report WRAPPER_top.ucdb -details -annotate -all -output CC_SVA_cov_rprt.txt
vcover report WRAPPER_top.ucdb -du=WRAPPER -recursive -assert -directive -cvg -codeAll -output cov_rprt_summary.txt
```

Wrapper Coverage

```
Recursive Coverage Report Summary Data by DU

================================================================================
=== Design Unit: work.WRAPPER
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Assertions                       2         2         0   100.00%
    Branches                        41        41         0   100.00%
    Conditions                       3         3         0   100.00%
    Directives                       2         2         0   100.00%
    Expressions                      3         3         0   100.00%
    FSM States                       5         5         0   100.00%
    FSM Transitions                  8         8         0   100.00%
    Statements                      48        48         0   100.00%
    Toggles                        208       208         0   100.00%


Total Coverage By Design Unit (filtered view): 100.00%
```

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 3

Total Coverage By Instance (filtered view): 100.00%
```

Wrapper Interfaces

```
interface ram_if (clk);
input clk;
logic [9:0] din;
logic       rst_n, rx_valid;
logic [7:0] dout;
logic       tx_valid;
logic [7:0] dout_gm;
logic       tx_valid_gm;
endinterface
```

```
interface slave_if (clk);
input clk;
logic       MOSI, rst_n, SS_n, tx_valid;
logic [7:0] tx_data;
logic [9:0] rx_data;
logic       rx_valid, MISO;
logic [9:0] rx_data_gm;
logic       rx_valid_gm, MISO_gm;
endinterface
```

```
interface wrapper_if(clk);
input clk;
logic MOSI, SS_n, clk, rst_n;
logic MISO;
logic MISO_gm;
endinterface
```

Wrapper Configs

```
package ram_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class ram_config extends uvm_object;
        `uvm_object_utils(ram_config)
        virtual ram_if ram_vif;
        uvm_active_passive_enum ram_mode;
        function new(string name = "ram_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

```
package slave_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_config extends uvm_object;
        `uvm_object_utils(slave_config)
        virtual slave_if slave_vif;
        uvm_active_passive_enum slave_mode;
        function new(string name = "slave_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

```
package wrapper_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_config extends uvm_object;
        `uvm_object_utils(wrapper_config)
        virtual wrapper_if wrapper_vif;
        uvm_active_passive_enum wrapper_mode;
        function new(string name = "wrapper_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

WRAPPER TOP

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

module wrapper_top ();
  bit clk;

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

  wrapper_if wrapper_vif (clk);
  ram_if ram_vif (clk);
  slave_if slave_vif (clk);

  WRAPPER DUT_WRAPPER (wrapper_vif.MOSI,wrapper_vif.MISO,wrapper_vif.SS_n,clk,wrapper_vif.rst_n);
  WRAPPER_GM DUT_WRAPPER_GM (wrapper_vif.MOSI,wrapper_vif.MISO_gm,wrapper_vif.SS_n,clk,wrapper_vif.rst_n);

  bind WRAPPER WRAPPER_SVA WRAPPER_SVA_INST (wrapper_vif.MOSI,wrapper_vif.MISO,wrapper_vif.SS_n,clk,wrapper_vif.rst_n);

  initial begin
    uvm_config_db#(virtual wrapper_if)::set(null,"uvm_test_top","WRAPPER_IF",wrapper_vif);
    uvm_config_db#(virtual ram_if)::set(null,"uvm_test_top","RAM_IF",ram_vif);
    uvm_config_db#(virtual slave_if)::set(null,"uvm_test_top","SLAVE_IF",slave_vif);
    run_test("wrapper_test");
  end

  initial begin
    $readmemh("RAM.dat",DUT_WRAPPER.RAM_instance.MEM);
    $readmemh("RAM.dat",DUT_WRAPPER_GM.RAM_gm_instance.MEM);
  end

  assign ram_vif.din        = DUT_WRAPPER.rx_data_din;
  assign ram_vif.rst_n      = DUT_WRAPPER.rst_n;
  assign ram_vif.rx_valid   = DUT_WRAPPER.rx_valid;
  assign ram_vif.dout       = DUT_WRAPPER.tx_data_dout;
  assign ram_vif.tx_valid   = DUT_WRAPPER.tx_valid;

  assign ram_vif.dout_gm     = DUT_WRAPPER_GM.tx_data_dout;
  assign ram_vif.tx_valid_gm = DUT_WRAPPER_GM.tx_valid;

  assign slave_vif.MOSI     = DUT_WRAPPER.MOSI;
  assign slave_vif.rst_n    = DUT_WRAPPER.rst_n;
  assign slave_vif.SS_n     = DUT_WRAPPER.SS_n;
  assign slave_vif.tx_valid = DUT_WRAPPER.tx_valid;
  assign slave_vif.tx_data  = DUT_WRAPPER.tx_data_dout;
  assign slave_vif.rx_data  = DUT_WRAPPER.rx_data_din;
  assign slave_vif.rx_valid = DUT_WRAPPER.rx_valid;
  assign slave_vif.MISO     = DUT_WRAPPER.MISO;

  assign slave_vif.rx_data_gm  = DUT_WRAPPER_GM.rx_data_din;
  assign slave_vif.rx_valid_gm = DUT_WRAPPER_GM.rx_valid;
  assign slave_vif.MISO_gm     = DUT_WRAPPER_GM.MISO;

endmodule
```

WRAPPER GOLDEN

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

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

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


RAM_GM   RAM_gm_instance   (rx_data_din,clk,rst_n,rx_valid,tx_data_dout,tx_valid);
SLAVE_GM SLAVE_gm_instance (clk,rst_n,MOSI,SS_n,tx_data_dout,tx_valid,MISO,rx_data_din,rx_valid);

endmodule
```

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

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

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

reg [7:0] MEM [255:0];

reg [7:0] Rd_Addr, Wr_Addr;

always @(posedge clk) begin
    if (~rst_n) begin
        dout <= 0;
        tx_valid <= 0;
        Rd_Addr <= 0;
        Wr_Addr <= 0;
    end
    else begin
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : begin
                    Wr_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b01 : begin
                    MEM[Wr_Addr] <= din[7:0];
                    tx_valid <= 0;
                end
                2'b10 : begin
                    Rd_Addr <= din[7:0];
                    tx_valid <= 0;
                end
                2'b11 : begin
                    dout <= MEM[Rd_Addr];
                    tx_valid <= 1;
                end
                default : tx_valid <= 0;
            endcase
        end
    end
end

endmodule
```

```
module SLAVE_GM (clk,rst_n,MOSI,SS_n,tx_data,tx_valid,MISO,rx_data,rx_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
                    if (received_address)
                        ns = READ_DATA;
                    else
                        ns = READ_ADD;
                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;
                    end
                end
            end
        endcase
    end
end
```

Wrapper Seq Item

```
package wrapper_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_seq_item extends uvm_sequence_item;
    `uvm_object_utils(wrapper_seq_item);

    rand logic       rst_n, SS_n;
    rand logic       MOSI;
         logic       MISO;
         logic       MISO_gm;

    // Typedef Enum to be descriptive with the operations to be done
    typedef enum bit [2:0] {WRITE_ADDR = 3'b000, WRITE_DATA = 3'b001, READ_ADDR = 3'b110, READ_DATA = 3'b111} e_state;

    rand bit        MOSI_BUS_RAND[];   // Dynamic Array to be randomized every cycle
         bit        MOSI_BUS[];        // Dynamic Array to take randomized values only from MOSI_BUS_RAND every start of communication
         int        ss_n_cnt;          // Counter to indicate Strat and End of communication
         int        count;             // Counter to Serialize the MOSI_BUS bit by bit in MOSI
         int        WIDTH;             // Integer to take only two values 13 (For all Operations except Read_Data) and 23 (For Read_Data)

    // States to model FSM concept
    rand e_state    wrapper_state;     // Randomized every cycle
         e_state    prev_state;        // Not Randomized to take values of wrapper_state only on the start of communication

    rand bit [1:0] mode_control;       // 0: write_only, 1: read_only, 2: write_read

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

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

    function string convert2string_stimulus();
        return $sformatf ("MOSI = %0b, SS_n = %0b, rst_n = %0d",MOSI,SS_n,rst_n);
    endfunction

    // 1- The reset signal (rst_n) shall be deasserted most of the time.
    constraint rst_n_c {rst_n dist {0:/2, 1:/98};}
```

```
// 2- The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles
constraint ss_n_c {
    if (!rst_n) {
        SS_n == 1;
    } else {
        // SS_n should be LOW during the data transfer (cycles 1-12 or 1-22) and HIGH only at the end (cycle 13 or 23)
        if (prev_state != READ_DATA) {          // For any operations except READ_DATA
            if (ss_n_cnt inside {[0:12]}) {
                SS_n == 0;
            } else { // ss_n_cnt == 13
                SS_n == 1;
            }
        } else {                                // For READ_DATA only
            if (ss_n_cnt inside {[0:22]}) {
                SS_n == 0;
            } else { // ss_n_cnt == 23
                SS_n == 1;
            }
        }
    }
}

// 3- Including Constraints 3,4,5 and 6 controlled by mode_select signal to Write-only, Read-only or Write_Read
constraint write_read_c {
    if (mode_control == 0){                         // Write only Sequence
        wrapper_state inside {WRITE_ADDR,WRITE_DATA};
        if (prev_state == WRITE_DATA)
            wrapper_state == WRITE_ADDR;
    }
    else if(mode_control == 1){                     // Read only Sequence
        wrapper_state inside {READ_ADDR,READ_DATA};
        if (prev_state == READ_DATA)
            wrapper_state == READ_ADDR;
    }
    else if(mode_control == 2){                     // Write/Read Sequence
        wrapper_state inside {WRITE_ADDR,WRITE_DATA,READ_ADDR,READ_DATA};
        if (prev_state == WRITE_ADDR)
            wrapper_state inside {WRITE_ADDR,WRITE_DATA};
        else if (prev_state == WRITE_DATA)
            wrapper_state dist {READ_ADDR :/ 60, WRITE_ADDR :/ 40};
        else if (prev_state == READ_ADDR)
            wrapper_state == READ_DATA;
        else
            wrapper_state dist {WRITE_ADDR :/ 60, READ_ADDR :/ 40};
    }
}
```

```
    // Constraint to Serialize MOSI_BUS on MOSI bit by bit
    constraint mosi_c {
        if (rst_n) {
            if (ss_n_cnt != WIDTH){  // Serialize During communication only
                MOSI == MOSI_BUS[count];
            }
        }
    }

    function void pre_randomize();
        if (!rst_n) begin
            count = 0;
        end
        else begin
            if (ss_n_cnt == 0) begin
                prev_state = wrapper_state; // Will be updated when ss_n_cnt == 0 (start of communication) (after SS_n asserted and de-asserted)
                count = 0;                  // Serialize count git inialized with zero to start communication

                if (prev_state == READ_DATA) begin    // 23 Cycles For Read Data
                    WIDTH = 23;
                    MOSI_BUS = new[WIDTH];
                    MOSI_BUS_RAND = new[WIDTH];
                end
                else begin                            // 13 Cycles For any other state
                    WIDTH = 13;
                    MOSI_BUS = new[WIDTH];
                    MOSI_BUS_RAND = new[WIDTH];
                end
            end
            else begin
                count = count + 1;                    // Increment serializer counter during running of simulation
            end
        end
        // $display("ss_n_cnt = %0d, count = %0d, MOSI_BUS = %0p, MOSI = %0b, prev_state = %03b, wrapper_state = %03b",ss_n_cnt,count,MOSI_BUS,MOS
    endfunction

    function void post_randomize();
        if (!rst_n) begin
            ss_n_cnt = 0;
        end else begin
            if (ss_n_cnt == 0) begin                  // At the Start of Simulation
                MOSI_BUS = MOSI_BUS_RAND;             // 1- Take Randomized values from MOSI_BUS_RAND to be assigned tp MOSI_BUS array
                for (int i=0; i<3; i++)   // 2- Loop for the 3 MSB to be like prev_state, which takes value of the required operation to be done
                    MOSI_BUS[i] = prev_state[2-i];
            end

            // Handle SS_n counter to control start and end of communication
            if ((ss_n_cnt != 13) && (prev_state != READ_DATA))
                ss_n_cnt += 1;
            else if ((ss_n_cnt != 23) && (prev_state == READ_DATA))
                ss_n_cnt += 1;
            else
                ss_n_cnt = 0;
        end
    endfunction

endclass
endpackage
```

Wrapper Sequence

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

class reset_sequence extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(reset_sequence);
    wrapper_seq_item rst_seq_item;

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

    task body();
        rst_seq_item = wrapper_seq_item::type_id::create("rst_seq_item");
        start_item(rst_seq_item);
        rst_seq_item.rst_n = 0;
        rst_seq_item.SS_n = 0;
        rst_seq_item.MOSI = 0;
        finish_item(rst_seq_item);
    endtask
endclass
```

```
class write_sequence extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(write_sequence);
    wrapper_seq_item write_seq_item;

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

    task body();
        write_seq_item = wrapper_seq_item::type_id::create("write_seq_item");
        repeat(10000) begin
            start_item(write_seq_item);
            assert(write_seq_item.randomize() with {write_seq_item.mode_control == 0;});
            finish_item(write_seq_item);
        end
    endtask
endclass

class read_sequence extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(read_sequence);
    wrapper_seq_item read_seq_item;

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

    task body();
        read_seq_item = wrapper_seq_item::type_id::create("read_seq_item");
        repeat(10000) begin
            start_item(read_seq_item);
            assert(read_seq_item.randomize() with {read_seq_item.mode_control == 1;});
            finish_item(read_seq_item);
        end
    endtask
endclass

class write_read_sequence extends uvm_sequence #(wrapper_seq_item);
    `uvm_object_utils(write_read_sequence);
    wrapper_seq_item wr_rd_seq_item;

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

    task body();
        wr_rd_seq_item = wrapper_seq_item::type_id::create("wr_rd_seq_item");
        repeat(10000) begin
            start_item(wr_rd_seq_item);
            assert(wr_rd_seq_item.randomize() with {wr_rd_seq_item.mode_control == 2;});
            finish_item(wr_rd_seq_item);
        end
    endtask
endclass
endpackage
```

Wrapper Sequencer

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

class ram_sequencer extends uvm_sequencer #(ram_seq_item);
    `uvm_component_utils(ram_sequencer);

    function new(string name = "ram_sequencer", uvm_component parent = null);
        super.new(name,parent);
    endfunction
endclass
endpackage
```

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

class slave_sequencer extends uvm_sequencer #(slave_seq_item);
    `uvm_component_utils(slave_sequencer);

    function new(string name = "slave_sequencer", uvm_component parent = null);
        super.new(name,parent);
    endfunction
endclass
endpackage
```

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

class wrapper_sequencer extends uvm_sequencer #(wrapper_seq_item);
    `uvm_component_utils(wrapper_sequencer);

    function new(string name = "wrapper_sequencer", uvm_component parent = null);
        super.new(name,parent);
    endfunction
endclass
endpackage
```

WRAPPER DRIVER

```
package ram_driver_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_driver extends uvm_driver #(ram_seq_item);
  `uvm_component_utils(ram_driver)
  virtual ram_if ram_vif;
  ram_seq_item stim_seq_item;

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

  task run_phase (uvm_phase phase);
    super.run_phase(phase);
    forever begin
        stim_seq_item = ram_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
        ram_vif.din = stim_seq_item.din;
        ram_vif.rst_n = stim_seq_item.rst_n;
        ram_vif.rx_valid = stim_seq_item.rx_valid;
        @(negedge ram_vif.clk);
        seq_item_port.item_done();
        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

```
package slave_driver_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_driver extends uvm_driver #(slave_seq_item);
  `uvm_component_utils(slave_driver)
  virtual slave_if slave_vif;
  slave_seq_item stim_seq_item;

  function new (string name = "slave_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 = slave_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
        slave_vif.MOSI = stim_seq_item.MOSI;
        slave_vif.rst_n = stim_seq_item.rst_n;
        slave_vif.SS_n = stim_seq_item.SS_n;
        slave_vif.tx_valid = stim_seq_item.tx_valid;
        slave_vif.tx_data = stim_seq_item.tx_data;
        @(negedge slave_vif.clk);
        seq_item_port.item_done();
        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

```
package wrapper_driver_pkg;
import wrapper_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_driver extends uvm_driver #(wrapper_seq_item);
  `uvm_component_utils(wrapper_driver)
  virtual wrapper_if wrapper_vif;
  wrapper_seq_item stim_seq_item;

  function new (string name = "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 = wrapper_seq_item::type_id::create("stim_seq_item");
        seq_item_port.get_next_item(stim_seq_item);
        wrapper_vif.MOSI = stim_seq_item.MOSI;
        wrapper_vif.rst_n = stim_seq_item.rst_n;
        wrapper_vif.SS_n = stim_seq_item.SS_n;
        @(negedge wrapper_vif.clk);
        seq_item_port.item_done();
        `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

Wrapper Monitor

```
package ram_monitor_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_monitor extends uvm_monitor;
    `uvm_component_utils(ram_monitor)
    virtual ram_if ram_vif;
    ram_seq_item rsp_seq_item;
    uvm_analysis_port #(ram_seq_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = ram_seq_item::type_id::create("rsp_seq_item");
            @(negedge ram_vif.clk);
            rsp_seq_item.din = ram_vif.din;
            rsp_seq_item.rx_valid = ram_vif.rx_valid;
            rsp_seq_item.rst_n = ram_vif.rst_n;
            rsp_seq_item.dout = ram_vif.dout;
            rsp_seq_item.tx_valid = ram_vif.tx_valid;
            rsp_seq_item.dout_gm = ram_vif.dout_gm;
            rsp_seq_item.tx_valid_gm = ram_vif.tx_valid_gm;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

```
package slave_monitor_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_monitor extends uvm_monitor;
    `uvm_component_utils(slave_monitor)
    virtual slave_if slave_vif;
    slave_seq_item rsp_seq_item;
    uvm_analysis_port #(slave_seq_item) mon_ap;

    function new(string name = "slave_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 = slave_seq_item::type_id::create("rsp_seq_item");
            @(negedge slave_vif.clk);
            rsp_seq_item.MOSI = slave_vif.MOSI;
            rsp_seq_item.rst_n = slave_vif.rst_n;
            rsp_seq_item.SS_n = slave_vif.SS_n;
            rsp_seq_item.tx_valid = slave_vif.tx_valid;
            rsp_seq_item.tx_data = slave_vif.tx_data;
            rsp_seq_item.rx_data = slave_vif.rx_data;
            rsp_seq_item.rx_valid = slave_vif.rx_valid;
            rsp_seq_item.MISO = slave_vif.MISO;
            rsp_seq_item.rx_data_gm = slave_vif.rx_data_gm;
            rsp_seq_item.rx_valid_gm = slave_vif.rx_valid_gm;
            rsp_seq_item.MISO_gm = slave_vif.MISO_gm;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

```
package wrapper_monitor_pkg;
import wrapper_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_monitor extends uvm_monitor;
    `uvm_component_utils(wrapper_monitor)
    virtual wrapper_if wrapper_vif;
    wrapper_seq_item rsp_seq_item;
    uvm_analysis_port #(wrapper_seq_item) mon_ap;

    function new(string name = "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 = wrapper_seq_item::type_id::create("rsp_seq_item");
            @(negedge wrapper_vif.clk);
            rsp_seq_item.MOSI = wrapper_vif.MOSI;
            rsp_seq_item.rst_n = wrapper_vif.rst_n;
            rsp_seq_item.SS_n = wrapper_vif.SS_n;
            rsp_seq_item.MISO = wrapper_vif.MISO;
            rsp_seq_item.MISO_gm = wrapper_vif.MISO_gm;
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
        end
    endtask
endclass

endpackage
```

Wrapper Agent

```
package ram_agent_pkg;
import ram_config_pkg::*;
import ram_driver_pkg::*;
import ram_monitor_pkg::*;
import ram_sequencer_pkg::*;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_agent extends uvm_agent;
`uvm_component_utils(ram_agent)
ram_sequencer sqr;
ram_monitor mon;
ram_driver drv;
ram_config ram_cfg;
uvm_analysis_port #(ram_seq_item) agt_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, "", "CFG_RAM",ram_cfg)) begin
        `uvm_fatal("build_phase","Unable to get configuration object")
    end
    if (ram_cfg.ram_mode == UVM_ACTIVE) begin
        sqr = ram_sequencer::type_id::create("sqr",this);
        drv = ram_driver::type_id::create("drv",this);
    end
    mon = ram_monitor::type_id::create("mon",this);
    agt_ap = new("agt_ap",this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    if (ram_cfg.ram_mode == UVM_ACTIVE) begin
        drv.ram_vif = ram_cfg.ram_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
    end
    mon.ram_vif = ram_cfg.ram_vif;
    mon.mon_ap.connect(agt_ap);
endfunction

endclass
endpackage
```

```
package slave_agent_pkg;
import slave_config_pkg::*;
import slave_driver_pkg::*;
import slave_monitor_pkg::*;
import slave_sequencer_pkg::*;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_agent extends uvm_agent;
`uvm_component_utils(slave_agent)
slave_sequencer sqr;
slave_monitor mon;
slave_driver drv;
slave_config slave_cfg;
uvm_analysis_port #(slave_seq_item) agt_ap;

function new (string name = "slave_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 #(slave_config)::get(this, "", "CFG_SLAVE",slave_cfg)) begin
        `uvm_fatal("build_phase","Unable to get configuration object")
    end
    if (slave_cfg.slave_mode == UVM_ACTIVE) begin
        sqr = slave_sequencer::type_id::create("sqr",this);
        drv = slave_driver::type_id::create("drv",this);
    end
    mon = slave_monitor::type_id::create("mon",this);
    agt_ap = new("agt_ap",this);
endfunction

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    if (slave_cfg.slave_mode == UVM_ACTIVE) begin
        drv.slave_vif = slave_cfg.slave_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
    end
    mon.slave_vif = slave_cfg.slave_vif;
    mon.mon_ap.connect(agt_ap);
endfunction

endclass
endpackage
```

```
package wrapper_agent_pkg;
import wrapper_config_pkg::*;
import wrapper_driver_pkg::*;
import wrapper_monitor_pkg::*;
import wrapper_sequencer_pkg::*;
import wrapper_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_agent extends uvm_agent;
`uvm_component_utils(wrapper_agent)
wrapper_sequencer sqr;
wrapper_monitor mon;
wrapper_driver drv;
wrapper_config wrapper_cfg;
uvm_analysis_port #(wrapper_seq_item) agt_ap;

function new (string name = "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 #(wrapper_config)::get(this, "", "CFG_WRAPPER",wrapper_cfg)) begin
        `uvm_fatal("build_phase","Unable to get configuration object")
    end
    if (wrapper_cfg.wrapper_mode == UVM_ACTIVE) begin
        sqr = wrapper_sequencer::type_id::create("sqr",this);
        drv = wrapper_driver::type_id::create("drv",this);
    end
    mon = wrapper_monitor::type_id::create("mon",this);
    agt_ap = new("agt_ap",this);
endfunction

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

endclass
endpackage
```

Wrapper Cov. Collector

```
package ram_coverage_pkg;
    import ram_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class ram_coverage extends uvm_component;
        `uvm_component_utils(ram_coverage)
        uvm_analysis_port #(ram_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(ram_seq_item) cov_fifo;
        ram_seq_item seq_item_cov;

        covergroup cvr_grp;
            // 1- Write Coverpoint to check transaction ordering for din[9:8]:
            // • Check din[9:8] takes 4 possible values
            // • Check write data after write address
            // • Check read data after read address
            // • Check write address => write data => read address => read data
            ram_order_cp : coverpoint seq_item_cov.din[9:8] {
                bins write_add        = {2'b00};
                bins write_data       = {2'b01};
                bins read_add         = {2'b10};
                bins read_data        = {2'b11};
                bins write_only_trans = (2'b00 => 2'b01);
                bins read_only_trans  = (2'b10 => 2'b11);
            }
            rstn_cp : coverpoint seq_item_cov.rst_n;
            rx_valid_cp : coverpoint seq_item_cov.rx_valid;
            tx_valid_cp : coverpoint seq_item_cov.tx_valid;

            // 2- Cross coverage:
            // • Between all bins of din[9:8] and rx_valid signal when it is high
            rx_valid_cross_ram_order : cross ram_order_cp, rx_valid_cp {
                ignore_bins ign_bin_1 = binsof(rx_valid_cp) intersect {0};
                ignore_bins ign_bin_2 = binsof(ram_order_cp.write_only_trans);
                ignore_bins ign_bin_3 = binsof(ram_order_cp.read_only_trans);
            }

            // • Between din[9:8] when it equals read data and tx_valid when it is high
            tx_valid_cross_ram_order : cross ram_order_cp, tx_valid_cp {
                bins read_data = binsof(ram_order_cp.read_data) && binsof(tx_valid_cp) intersect {1};
                option.cross_auto_bin_max = 0;
            }
        endgroup

        function new (string name = "ram_coverage", uvm_component parent = null);
            super.new(name,parent);
            cvr_grp = 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);
                cvr_grp.sample();
            end
        endtask
    endclass
endpackage
```

```
package slave_coverage_pkg;
    import slave_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class slave_coverage extends uvm_component;
        `uvm_component_utils(slave_coverage)
        uvm_analysis_port #(slave_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(slave_seq_item) cov_fifo;
        slave_seq_item seq_item_cov;

        covergroup cvr_grp;
            // 1- Add coverpoints on rx_data[9:8] to take all possible values and all possible transitions
            rx_data_cp : coverpoint seq_item_cov.rx_data[9:8]{
                bins wr_addr = {2'b00};
                bins wr_data = {2'b01};
                bins rd_addr = {2'b10};
                bins rd_data = {2'b11};
                bins write_trans = (2'b00 => 2'b01);
                bins read_trans  = (2'b10 => 2'b11);
            }
            // 2- Add coverpoints on SS_n to capture:
            // A. Check full transaction duration: 1 → 0 [*13] → 1 for normal operations.
            // B. Check extended transaction: 1 → 0 [*23] → 1 for READ_DATA
            SS_n_cp : coverpoint seq_item_cov.SS_n {
                bins ss_n_trans = (1=>0[*13]=>1);
                bins ss_n_trans_read_data = (1=>0[*23]=>1);
                bins ss_n_start_trans = (1=>0);
                bins ss_n_end_trans = (0=>1);
            }
            // 3- Add coverpoints on MOSI to validate correct transitions:
            // A. 000 (Write Address)
            // B. 001 (Write Data)
            // C. 110 (Read Address)
            // D. 111 (Read Data)
            MOSI_cp: coverpoint seq_item_cov.MOSI{
                bins wr_addr_trans = (0=>0=>0);
                bins wr_data_trans = (0=>0=>1);
                bins rd_addr_trans = (1=>1=>0);
                bins rd_data_trans = (1=>1=>1);
            }
            // 4- Cross coverage between SS_n and MOSI bins (Exclude irrelevant bins to focus on legal operation scenarios)
            MISO_cp: coverpoint seq_item_cov.MISO;
            ss_n_op_cross: cross SS_n_cp, MOSI_cp{
                ignore_bins bin_1 = binsof(SS_n_cp.ss_n_trans);
                ignore_bins bin_2 = binsof(SS_n_cp.ss_n_trans_read_data);
                ignore_bins bin_3 = binsof(SS_n_cp.ss_n_end_trans);
            }
            rstn_cp : coverpoint seq_item_cov.rst_n;
            rx_valid_cp : coverpoint seq_item_cov.rx_valid;
            tx_valid_cp : coverpoint seq_item_cov.tx_valid;
        endgroup

        function new (string name = "slave_coverage", uvm_component parent = null);
            super.new(name,parent);
            cvr_grp = 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);
                cvr_grp.sample();
            end
        endtask
    endclass
endpackage
```

```
package wrapper_coverage_pkg;
    import wrapper_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class wrapper_coverage extends uvm_component;
        `uvm_component_utils(wrapper_coverage)
        uvm_analysis_port #(wrapper_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
        wrapper_seq_item seq_item_cov;

        covergroup cvr_grp;
            rstn_cp : coverpoint seq_item_cov.rst_n;
            SS_n_cp : coverpoint seq_item_cov.SS_n {
                bins ss_n_trans = (1=>0[*13]=>1);
                bins ss_n_trans_read_data = (1=>0[*23]=>1);
                bins ss_n_start_trans = (1=>0);
                bins ss_n_end_trans = (0=>1);
            }
            MOSI_cp: coverpoint seq_item_cov.MOSI {
                bins wr_addr_trans = (0=>0=>0);
                bins wr_data_trans = (0=>0=>1);
                bins rd_addr_trans = (1=>1=>0);
                bins rd_data_trans = (1=>1=>1);
            }
            MISO_cp: coverpoint seq_item_cov.MISO;
            ss_n_op_cross: cross SS_n_cp, MOSI_cp {
                ignore_bins bin_1 = binsof(SS_n_cp.ss_n_trans);
                ignore_bins bin_2 = binsof(SS_n_cp.ss_n_trans_read_data);
                ignore_bins bin_3 = binsof(SS_n_cp.ss_n_end_trans);
            }
        endgroup

        function new (string name = "wrapper_coverage", uvm_component parent = null);
            super.new(name,parent);
            cvr_grp = 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);
                cvr_grp.sample();
            end
        endtask
    endclass
endpackage
```

Wrapper Scoreboard

```
package ram_scoreboard_pkg;
import ram_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class ram_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(ram_scoreboard);
    uvm_analysis_export #(ram_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(ram_seq_item) sb_fifo;
    ram_seq_item seq_item_sb;

    int error_count;
    int correct_count;

    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);
        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);
                if ((seq_item_sb.dout !== seq_item_sb.dout_gm) || (seq_item_sb.tx_valid !== seq_item_sb.tx_valid_gm)) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT: %s while the refecrence dout: %0d, tx_valid: %0d"
                        ,seq_item_sb.convert2string(),seq_item_sb.dout_gm,seq_item_sb.tx_valid_gm));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct ram_out : %s",seq_item_sb.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            end
    endtask

    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Ram: Total successful counts : %0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Ram: Total failed counts : %0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

```
package slave_scoreboard_pkg;
import slave_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class slave_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(slave_scoreboard);
    uvm_analysis_export #(slave_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(slave_seq_item) sb_fifo;
    slave_seq_item seq_item_sb;

    int error_count;
    int correct_count;

    function new(string name = "slave_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);
                if ((seq_item_sb.rx_data != seq_item_sb.rx_data_gm) || (seq_item_sb.rx_valid != seq_item_sb.rx_valid_gm) || (seq_item_sb.MISO != seq_item_sb.MISO_gm)) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT: %s while the refecrence rx_data: %0d, rx_valid: %0b, MISO: %0b",
                        seq_item_sb.convert2string(),seq_item_sb.rx_data_gm,seq_item_sb.rx_valid_gm,seq_item_sb.MISO_gm));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct slave_out : %s",seq_item_sb.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            end
    endtask

    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Slave: Total successful counts : %0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Slave: Total failed counts : %0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

```
package wrapper_scoreboard_pkg;
import wrapper_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(wrapper_scoreboard);
    uvm_analysis_export #(wrapper_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;
    wrapper_seq_item seq_item_sb;

    int error_count;
    int correct_count;

    function new(string name = "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);
                if (seq_item_sb.MISO != seq_item_sb.MISO_gm) begin
                    `uvm_error("run_phase",$sformatf("Comparison failed, Transaction recieved by the DUT: %s
                        while the refecrence MISO: %0b",seq_item_sb.convert2string(),seq_item_sb.MISO_gm));
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",$sformatf("correct wrapper_out : %s",seq_item_sb.convert2string()),UVM_HIGH);
                    correct_count++;
                end
            end
    endtask

    function void report_phase (uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("Wrapper: Total successful counts : %0d",correct_count),UVM_MEDIUM);
        `uvm_info("report_phase",$sformatf("Wrapper: Total failed counts : %0d",error_count),UVM_MEDIUM);
    endfunction
endclass
endpackage
```

Wrapper Environment

```
package ram_env_pkg;
import ram_agent_pkg::*;
import ram_scoreboard_pkg::*;
import ram_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class ram_env extends uvm_env;
  `uvm_component_utils(ram_env)
  ram_agent agt;
  ram_scoreboard sb;
  ram_coverage cov;

  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);
    agt = ram_agent::type_id::create("agt",this);
    sb = ram_scoreboard::type_id::create("sb",this);
    cov = ram_coverage::type_id::create("cov",this);
  endfunction

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

```
package slave_env_pkg;
import slave_agent_pkg::*;
import slave_scoreboard_pkg::*;
import slave_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class slave_env extends uvm_env;
  `uvm_component_utils(slave_env)
  slave_agent agt;
  slave_scoreboard sb;
  slave_coverage cov;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = slave_agent::type_id::create("agt",this);
    sb = slave_scoreboard::type_id::create("sb",this);
    cov = slave_coverage::type_id::create("cov",this);
  endfunction

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

```
package wrapper_env_pkg;
import wrapper_agent_pkg::*;
import wrapper_scoreboard_pkg::*;
import wrapper_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class wrapper_env extends uvm_env;
  `uvm_component_utils(wrapper_env)
  wrapper_agent agt;
  wrapper_scoreboard sb;
  wrapper_coverage cov;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = wrapper_agent::type_id::create("agt",this);
    sb = wrapper_scoreboard::type_id::create("sb",this);
    cov = wrapper_coverage::type_id::create("cov",this);
  endfunction

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

WRAPPER TEST

```
package wrapper_test_pkg;
import ram_config_pkg::*;
import slave_config_pkg::*;
import wrapper_config_pkg::*;
import ram_env_pkg::*;
import slave_env_pkg::*;
import wrapper_env_pkg::*;
import wrapper_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class wrapper_test extends uvm_test;
  `uvm_component_utils (wrapper_test)
  wrapper_env env_wrapper;
  ram_env env_ram;
  slave_env env_slave;
  wrapper_config wrapper_cfg;
  ram_config ram_cfg;
  slave_config slave_cfg;
  reset_sequence reset_seq;
  write_sequence write_seq;
  read_sequence read_seq;
  write_read_sequence write_read_seq;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env_wrapper = wrapper_env::type_id::create("env_wrapper",this);
    env_ram = ram_env::type_id::create("env_ram",this);
    env_slave = slave_env::type_id::create("env_slave",this);
    wrapper_cfg = wrapper_config::type_id::create("wrapper_cfg");
    ram_cfg = ram_config::type_id::create("ram_cfg");
    slave_cfg = slave_config::type_id::create("slave_cfg");
    reset_seq = reset_sequence::type_id::create("reset_seq");
    write_seq = write_sequence::type_id::create("write_seq");
    read_seq = read_sequence::type_id::create("read_seq");
    write_read_seq = write_read_sequence::type_id::create("write_read_seq");

    if (!uvm_config_db#(virtual wrapper_if)::get(this,"","WRAPPER_IF",wrapper_cfg.wrapper_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the wrapper from the uvm_config_db")
    if (!uvm_config_db#(virtual ram_if)::get(this,"","RAM_IF",ram_cfg.ram_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the ram from the uvm_config_db")
    if (!uvm_config_db#(virtual slave_if)::get(this,"","SLAVE_IF",slave_cfg.slave_vif))
      `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the slave from the uvm_config_db")

    wrapper_cfg.wrapper_mode = UVM_ACTIVE;
    ram_cfg.ram_mode = UVM_PASSIVE;
    slave_cfg.slave_mode = UVM_PASSIVE;

    uvm_config_db#(wrapper_config)::set(this,"env_wrapper.*","CFG_WRAPPER",wrapper_cfg);
    uvm_config_db#(ram_config)::set(this,"env_ram.*","CFG_RAM",ram_cfg);
    uvm_config_db#(slave_config)::set(this,"env_slave.*","CFG_SLAVE",slave_cfg);
  endfunction
```

```
  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","Reset Asserted",UVM_LOW)
    reset_seq.start(env_wrapper.agt.sqr);
    `uvm_info("run_phase","Reset Deasserted",UVM_LOW)

    `uvm_info("run_phase","wrapper Write Stimulus Generation Started",UVM_LOW)
    write_seq.start(env_wrapper.agt.sqr);
    `uvm_info("run_phase","wrapper Write Stimulus Generation Ended",UVM_LOW)

    `uvm_info("run_phase","wrapper Read Stimulus Generation Started",UVM_LOW)
    read_seq.start(env_wrapper.agt.sqr);
    `uvm_info("run_phase","wrapper Read Stimulus Generation Ended",UVM_LOW)

    `uvm_info("run_phase","wrapper Write & Read Stimulus Generation Started",UVM_LOW)
    write_read_seq.start(env_wrapper.agt.sqr);
    `uvm_info("run_phase","wrapper Write & Read Stimulus Generation Ended",UVM_LOW)
    phase.drop_objection(this);
  endtask
endclass

endpackage
```

WRAPPER SVA

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

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

sequence write_add_seq;
    (!SS_n ##1 !MOSI ##1 !MOSI ##1 !MOSI);
endsequence

sequence write_data_seq;
    (!SS_n ##1 !MOSI ##1 !MOSI ##1 MOSI);
endsequence

sequence read_add_seq;
    (!SS_n ##1 MOSI ##1 MOSI ##1 !MOSI);
endsequence

sequence read_data_seq;
    (!SS_n ##1 MOSI ##1 MOSI ##1 MOSI);
endsequence

// 1- An assertion ensures that whenever reset is asserted, the outputs (MISO) are all inactive.
property rstn_property;
    @(posedge clk) !rst_n |=> (!MISO);
endproperty

// 2- An assertion to make sure that the MISO remains with a stable value as long as it is not a read data operation
property miso_property;
    @(posedge clk) disable iff(!rst_n) write_add_seq or write_data_seq or read_add_seq |=> $stable(MISO) [->1];
endproperty

rstn_assertion: assert property (rstn_property);
rstn_cover:     cover  property (rstn_property);

miso_assertion: assert property (miso_property);
miso_cover:     cover  property (miso_property);

endmodule
```