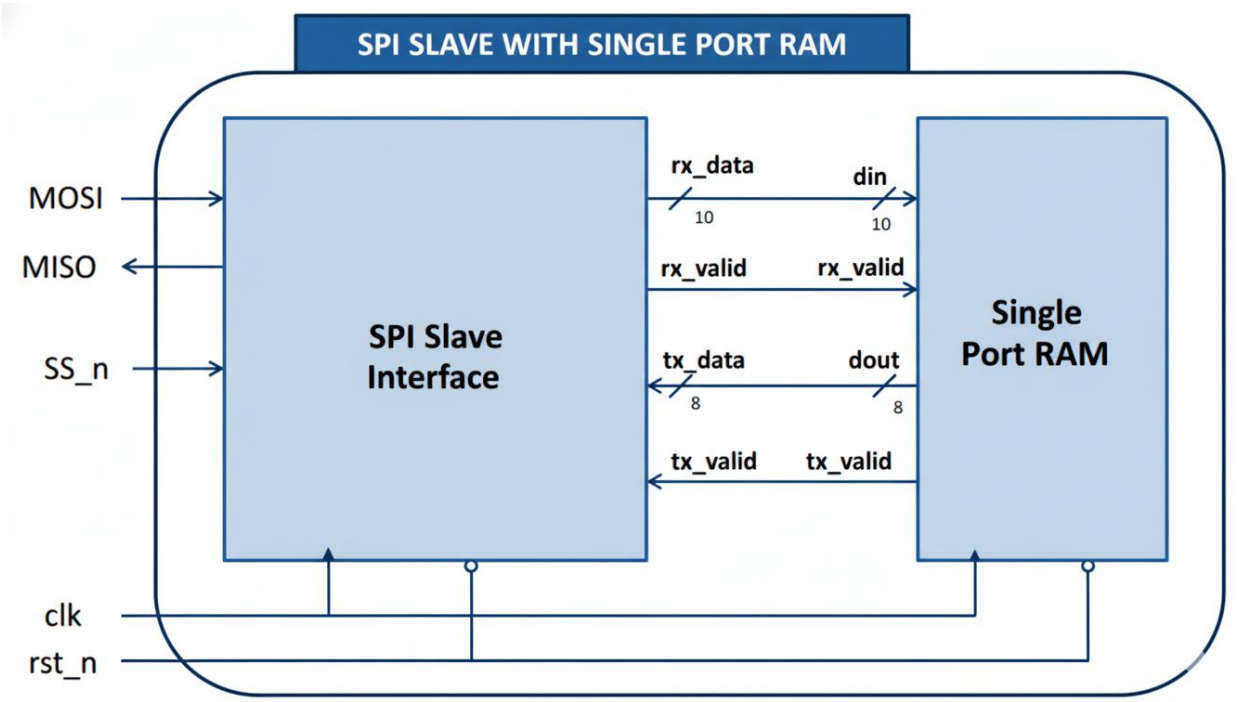
SPI Slave with Single Port RAM
UVM Project



TEAM MEMBERS

1. Rawan Mohamed Waziry
2. Marina Bebawy Nasr
3. Karim Maaty

CONTENTS

VERIFICATION PLAN

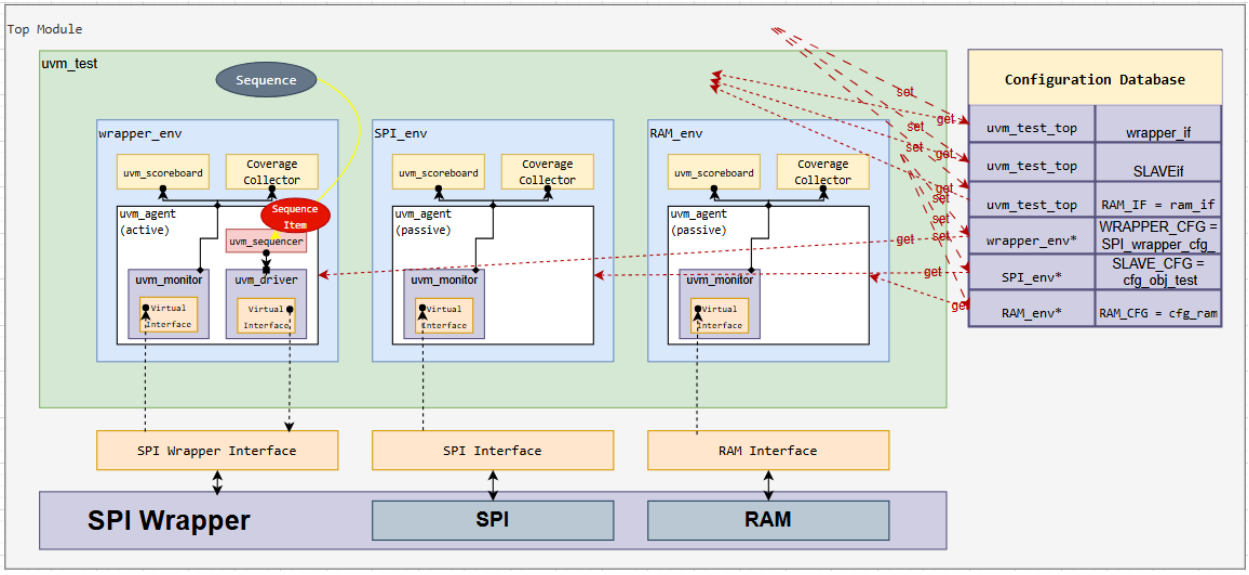
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SLAVE_1	When rst_n is asserted the outputs (MISO, rx_valid, and rx_data) should be low	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off most of the time	-	Output is checked against a golden model and using concurrent assertion
SLAVE_2	When the sequence 000 is sent serially to the MOSI after the SS_n falls, the first bit define the next state, the next two bits are sent on rx_data bus to define the command the last 8 bits are sent to the ram on rx_data bus to be stored in the internal write address bus	Randomized with constraints on the first 3 bits sent on mosi to have valid transitions only (000,001,110,111)	Covering all valid values and transitions of rx_data[9:8] and valid transitions of MOSI	Output is checked against a golden model and transistion of states is checked using concurrent assertion
SLAVE_3	When the sequence 001 is sent serially to the MOSI after the SS_n falls, the first bit define the next state, the next two bits are sent on rx_data bus to define the command, the last 8_bits are sent to the ram on rx_data bus to be stored in the wr_address previously held	Randomized with constraints on the first 3 bits sent on mosi to have valid transitions only (000,001,110,111)	Covering all valid values and transitions of rx_data[9:8] and valid transitions of MOSI	Output is checked against a golden model and transistion of states is checked using concurrent assertion
SLAVE_4	When the sequence 110 is sent serially to the MOSI after the SS_n falls, the first bit define the next state, the next two bits are sent on rx_data bus to define the command, and the last 8 bits are sent as data (on the rx_data bus) to be stored in the ram at	Randomized with constraints on the first 3 bits sent on mosi to have valid transitions only (000,001,110,111) and that read address is never followed by another read address operation	Covering all valid values and transitions of rx_data[9:8] and valid transitions of MOSI	Output is checked against a golden model and transistion of states is checked using concurrent assertion

	the internal read address.			
SLAVE_5	When the sequence 111 is sent serially to the MOSI after the SS_n falls, the first bit define the next state, the next two bits are sent on rx_data bus to define the command,the output MISO should take the value of the data stored in the previously held read address bus after being converted from parallel to serial	Randomized with constraints on the first 3 bits sent on mosi to have valid transitions only (000,001,110,111) and that read data is never followed by another read data operation	Covering all valid values and transitions of rx_data[9:8] and valid transitions of MOSI	Output is checked against a golden model and transistion of states is checked using concurrent assertion
SLAVE_6	When tx_valid is asserted, the slave is informed that data out is ready	Randomized under constraints to be asserted only in case of read data	-	Output is checked against a golden model
SLAVE_7	When communication is ended from the master side, SS_n is asserted and the slave return to the idle case	Randomized under constraints to be asserted at the end of operation only	Covering all valid transactions transitions and cross covering start of communication transition with all mosi valid transitions fo different operations	-
Wrapper_1	If the rst_n is asserted high, in the sequence of write_data_only the master will keep writing_addressess and data to the slave for the RAM to store	Randomized under constraints on the first 3 MOSI bits to be 000 (WRITE_ADD) and 001 (WRITE_DATA) only	-	Checked against a golden model
Wrapper_2	If the rst_n is asserted high, in the sequence of read_data_only the master will keep sending addresses and reading data to the slave for the RAM to send back to MASTER	Randomized under constraints on the first 3 MOSI bits to be 110 (READ_ADD) and 111 (READ_DATA) in the same order every time	-	Checked against a golden model
Wrapper_3	If the rst_n is asserted high, in the sequence of read_write_data the master will	Randomized under constraints: - if the current state	-	Checked against a golden model

	communicate with the slave varying between modes to test the SLAVE and the RAM	is READ_ADD then the next state would be READ_DATA. - if the current state is READ_DATA then the next state would be WRITE_ADD by 60% or READ_ADD by 40%. - if the current state is WRITE_ADD then the next state would be WRITE_DATA or WRITE_ADD. - if the cuttent state is WRITE_DATA then the next state would be WRITE_ADD by 40% or READ_ADD by 60%.		
RAM_1	On every reset assertion, outputs dout and tx_valid should be 0. Also the internals Wr_Addr, Rd_Addr are reset.	Directed at the beginning of the simulation, then randomized with a constraint on reset to be off most of the simulation time.	-	RAM outputs are checked against a golden model. Concurrent assertions on dout and tx_valid outputs verify the synchronous reset functionality.
RAM_2	For a write-only sequence, every Write Address operation shall always be followed by either Write Address or Write Data operation.	Randomization with constraints so that WRITE_ADDRESS is always followed by either WRITE_ADDRESS or WRITE_DATA operation.	Coverpoints for din[9:8]: • Check write data after write address Cross coverage: • Between all bins of din[9:8] and rx_valid signal when it is high	RAM outputs are checked against a golden model. Concurrent assertions verify that WRITE_ADDRESS is followed by either WRITE_ADDRESS or WRITE_DATA, and that tx_valid is low.
RAM_3	For a read-only sequence, every Read Address operation shall always be followed by Read Data. After a Read Data operation shall always	Randomization with constraints so that READ_ADDRESS is always followed by READ_DATA operation, and READ_DATA is	Coverpoint for din[9:8]: • Check read data after read address Cross coverage: • Between all	RAM outputs are checked against a golden model. Concurrent assertions verify that READ_ADDRESS is

	be followed by Read Address.	always followed by READ_ADDRESS operation.	bins of din[9:8] and rx_valid signal when it is high • Between din[9:8] when it equals read data and tx_valid when it is high	followed by READ_DATA, and READ_DATA is followed by READ_ADDRESS. tx_valid is low during the READ_ADDRESS operation and high during the READ_DATA operation.
RAM_4	For a randomized read/write sequence, ordering rules shall be enforced.	Randomization with constraints that: - Every WRITE_ADDRESS is always followed by either WRITE_ADDRESS or WRITE_DATA operation. - Every READ_ADDRESS is always followed by READ_DATA operation. - After READ_DATA: 60% WRITE_ADDRESS, 40% READ_ADDRESS. - After WRITE_DATA: 60% READ_ADDRESS, 40% WRITE_ADDRESS.	Coverpoints 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 Cross coverage: • Between all bins of din[9:8] and rx_valid signal when it is high • Between din[9:8] when it equals read data and tx_valid when it is high	RAM outputs are checked against a golden model. Concurrent assertions to verify the proper operation order and to ensure that tx_valid is high during the READ_DATA operation and low during all other operations.

UVM TESTBENCH DIAGRAM



- The testbench begins at the **Top module** instantiating the interfaces for each module, then instantiating the main modules: **Wrapper**, **SPI_slave**, **RAM** and their reference models to be compared with, then connecting the common wires of the interfaces to enable the communication between the modules, at the end each interface is stored in the **UVM Configuration Database** to be used by the **uvm_test** component.

- There are three main environments: **wrapper_env**, **SPI_env**, and **RAM_env**, all instantiated under the **uvm_test** component, the **uvm_test** gets each interface from the intefaces and stores them in **configuration objects** and sets them again in the database, each one is only visible by its environment to avoid corruption of the data.

- The **wrapper_env** acts as the active environment, generating and driving transactions through the SPI Wrapper interface, while the **SPI_env** and **RAM_env** are passive and monitor the corresponding SPI and RAM interfaces.

- There are four sequences to go through to check on the functionality of the modules, **reset_seq**, **write_only_seq**, **read_only_seq** and **read_write_seq** to ensure testing all the cases of the slave and the RAM.

- **Each sequence** running at the test level creates a **sequence item** which is randomized every time the **driver** requests for a **seq_item** and then sends to the **driver** through the **wrapper_env sequencer**, when driver finishes, it requests for another sequence item to be driven to the **wrapper** through the **wrapper_interface** and repeat the same cycle until the number of iterations set in the sequence ends.

- The DUT is driven via the SPI Wrapper interface. The **monitors** in the SPI and RAM environments capture the DUT responses and forward them to their respective **scoreboards** and **coverage collectors** for checking and analysis.

The **configuration database** connects all environments by providing virtual interfaces and configuration objects. The top-level test sets these configurations, and each environment retrieves its relevant data (interfaces and settings). This setup ensures that all environments are properly linked to their DUT interfaces enabling synchronized operation across the testbench.

Part One – SPI Slave

Code Snippets

Design

```
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;  // edit_1 replacing READ_ADD with
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; // edit_2 Counter should be 0 when the rst_n is asserted
    end
    else begin
        case (cs)
            IDLE : begin
```

```
            rx_valid <= 0;
        end
        CHK_CMD : begin
            counter <= 10;
        end
        WRITE : begin
            counter <= counter - 1;
            if (counter > 0) begin
                rx_data[counter-1] <= MOSI;
            end
            rx_valid <= (counter ==0 )? 1 : 0; //EDIT
        end
        READ_ADD : begin
            counter <= counter - 1;
            if (counter > 0) begin
                rx_data[counter-1] <= MOSI;
            end
            else begin
                received_address <= 1;
            end
            rx_valid <= (counter ==0 )? 1 : 0; //EDIT
        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
                // edit_3 when the counter is set to 8
                // the rx_data continues reading again.
                // Adding the condition of NOT rx_valid
                // will fix it
                if (counter > 0 && ~rx_valid) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    counter <= 8;
```

```
                    end
                end
            end
        endcase
    end
end

/*************************************************************************
**********************/
`ifdef SIM
    property p_idle_chk_trans;
        @(posedge clk) disable iff (~rst_n) ((!$stable(cs)) && $past(cs)==IDLE)
|-> (cs==CHK_CMD);
    endproperty
    assert property (p_idle_chk_trans);
    cover property (p_idle_chk_trans);

    property chk_to_states_trans;
        @(posedge clk) disable iff (~rst_n) ( (!$stable(cs)) && (cs==READ_ADD ||
cs==WRITE || cs==READ_DATA))
        |-> ($past(cs)==CHK_CMD);
    endproperty
    assert property (chk_to_states_trans);
    cover property (chk_to_states_trans);


    property p_to_idle_trans;
        @(posedge clk) disable iff (~rst_n) ((!$stable(cs)) && ($past(cs)==WRITE
|| $past(cs)==READ_ADD
        || $past(cs)==READ_DATA)) |-> (cs==IDLE);
    endproperty
    assert property (p_to_idle_trans);
    cover property (p_to_idle_trans);

`endif
/*************************************************************************
**********************/


endmodule
```

GOLDEN MODEL

```
module spi_slave_ref #(
    parameter IDLE      = 3'b000, // 0
    parameter WRITE     = 3'b001, // 1
    parameter CHK_CMD   = 3'b010, // 2
    parameter READ_ADD  = 3'b011, // 3
    parameter READ_DATA = 3'b100  // 4
) (
    input            MOSI,      // data sent from the master to the slave
    input            SS_n,      // signal by the master to enable the slave
    input            clk,
    input            rst_n,
    input      [7:0] tx_data,   // for data coming from the RAM
    input            tx_valid,  // enable to store the data coming from the RAM
    output reg       MISO,      // data sent from the slave to the master
    output reg [9:0] rx_data,   // for data to be sent to the RAM
    output reg       rx_valid   // enable for the RAM to store the data on
rx_data
);

    (* fsm_encoding = "gray" *)
    reg [2:0] cs, ns;   // current state and next state
    reg [5:0] counter;  // counter for serial to parallel data conversion
    reg       rd_addr;  // a signal to know if the address is received or not

    // next state logic
    always @(*) begin
        case (cs)
            IDLE:    ns = (SS_n) ? IDLE : CHK_CMD;
            CHK_CMD: begin
                if (~SS_n)
                  if (MOSI) ns = (rd_addr) ? READ_DATA : READ_ADD;
                  else ns = WRITE;
                else ns = IDLE;
            end
            WRITE:   ns = (SS_n == 1) ? IDLE : WRITE;
            READ_ADD: begin
                if (SS_n == 1) ns = IDLE;
                else ns = READ_ADD;
            end
            READ_DATA: begin
                if ( SS_n == 1) ns = IDLE;
                else ns = READ_DATA;
            end
            default: ns = IDLE;
```

```
        endcase
    end

    // current state memory and slave data handling
    always @(posedge clk) begin
        if (~rst_n) begin
            cs      <= IDLE;
            counter <= 3'b0;
            MISO    <= 0;
            rx_data <= 10'b0;
            rd_addr <= 0;
            rx_valid <= 0;
        end
        else begin
            cs <= ns;

            // First state
            if (cs == IDLE) begin
                counter <= 0;
                rx_valid <= 0;
            end

            // Second and third states
            else if (cs == WRITE || cs == READ_ADD) begin
                // Raising the rx_valid flag when all the data are converted
parallel
                // for one cycle for the ram to capture it
                rx_valid <= (counter == 10)? 1 : 0;

                // Converting the series data into parallel to send it to the ram
                if (counter < 10)
                    rx_data[9-counter] <= MOSI;
                else if (cs == READ_ADD && counter==10)
                    // Storing the info of receiving an address
                    // to enter the READ_DATA mode the next time
                    rd_addr <= 1;

                // Incrementing the counter every cycle
                counter <= counter + 1;
            end

            // Fourth state
            else if (cs == READ_DATA) begin
                if(~tx_valid) begin
                    // Receiving 2 bits operation bits from the master
```

```
                    // and 8 dummy bits
                    // total 11 clk cycles
                    if (counter < 10 && ~rx_valid) begin
                        rx_data[9-counter] <= MOSI;
                        counter <= counter + 1;
                    end
                    else if( counter == 10) begin
                        rx_valid <= 1;
                        counter <= 0;
                    end
                end
                else begin
                    // Receiving 8 data bits from the ram
                    // total 9 clk cycles
                    rx_valid <= 0;
                    if (counter < 8)
                        MISO <= tx_data[7-counter];
                    else if (counter == 8)
                        rd_addr <= 0;
                    counter <= counter + 1;
                end
            end
        end
    end
endmodule
```

INTERFACE

```
interface spi_slave_if(clk);
input logic clk;
logic  MOSI, rst_n, SS_n, tx_valid;
logic  [7:0] tx_data;
logic  [9:0] rx_data;
logic  rx_valid, MISO;
logic  rx_valid_exp, MISO_exp;
logic  [9:0] rx_data_exp;

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

endinterface
```

SVA

```
module spi_slave_sva(
input MOSI, clk, rst_n, SS_n, tx_valid,
input      [7:0] tx_data,
input [9:0] rx_data,
input       rx_valid, MISO
);

property p_rx_valid_wr_add;
    @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(~MOSI)[*3]) |-> ##10
($rose(rx_valid) && SS_n[->1]);
endproperty
assert property(p_rx_valid_wr_add);
cover property (p_rx_valid_wr_add);

property p_rx_valid_wr_data;
    @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(~MOSI)[*2] ##1 (MOSI)) |-
> ##10 ($rose(rx_valid) && SS_n[->1]);
endproperty
assert property(p_rx_valid_wr_data);
cover property (p_rx_valid_wr_data);

property p_rx_valid_rd_add;
    @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(MOSI)[*2] ##1 (~MOSI)) |-
> ##10 ($rose(rx_valid) && SS_n[->1]);
endproperty
assert property(p_rx_valid_rd_add);
cover property (p_rx_valid_rd_add);


property p_rx_valid_rd_data;
    @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(MOSI)[*3]) |-> ##10
($rose(rx_valid) && SS_n[->1]);
endproperty
assert property(p_rx_valid_rd_data);
cover property (p_rx_valid_rd_data);

property p_reset;
    @(posedge clk) (~rst_n) |=> (MISO==0 && rx_data==0 && rx_valid==0);
endproperty
assert property (p_reset);
cover property (p_reset);


endmodule
```

TOP MODULE

```
module spi_slave_top();
import uvm_pkg::*;
`include"uvm_macros.svh"
import spi_slave_test_pkg::*;

    bit clk;

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

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

    spi_slave_ref GM (SLAVEif.MOSI, SLAVEif.SS_n, SLAVEif.clk, SLAVEif.rst_n,
SLAVEif.tx_data,
    SLAVEif.tx_valid, SLAVEif.MISO_exp, SLAVEif.rx_data_exp,
SLAVEif.rx_valid_exp);

    bind SLAVE spi_slave_sva spi_sva_inst (SLAVEif.MOSI, SLAVEif.clk,
SLAVEif.rst_n, SLAVEif.SS_n, SLAVEif.tx_valid,
    SLAVEif.tx_data, SLAVEif.rx_data, SLAVEif.rx_valid, SLAVEif.MISO);

    initial begin
        uvm_config_db#(virtual
spi_slave_if)::set(null,"uvm_test_top","SLAVE_IF",SLAVEif);
        run_test("spi_slave_test");
    end

endmodule
```

TEST

```
package spi_slave_test_pkg;
    import uvm_pkg::*;
    `include"uvm_macros.svh"
    import spi_slave_config_obj_pkg::*;
    import spi_slave_environment_pkg::*;
    import spi_slave_main_seq_pkg::*;
    import spi_slave_reset_seq_pkg::*;

    class spi_slave_test extends uvm_test;
        `uvm_component_utils(spi_slave_test)
        spi_slave_config_obj cfg_obj_test;
        spi_slave_environment env;
        spi_slave_reset_seq rst_seq;
        spi_slave_main_seq main_seq;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            cfg_obj_test=spi_slave_config_obj::type_id::create("cfg_obj_test");
            if(!uvm_config_db#(virtual
spi_slave_if)::get(this,"","SLAVE_IF",cfg_obj_test.config_vif))
                `uvm_fatal("build_phase","Test-Unable to get the virtual
interface");

            cfg_obj_test.is_active = UVM_ACTIVE;
            uvm_config_db
#(spi_slave_config_obj)::set(this,"*","CFG",cfg_obj_test);

            env=spi_slave_environment::type_id::create("env",this);
            rst_seq=spi_slave_reset_seq::type_id::create("rst_seq");
            main_seq=spi_slave_main_seq::type_id::create("main_seq");
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            rst_seq.start(env.agt.sqr);
            main_seq.start(env.agt.sqr);
            phase.drop_objection(this);
        endtask
    endclass
```

```
endpackage
```

ENVIRONMENT

```
package spi_slave_environment_pkg;
    import uvm_pkg::*;
    `include"uvm_macros.svh"
    import spi_slave_driver_pkg::*;
    import spi_slave_agent_pkg::*;
    import spi_slave_scoreboard_pkg::*;
    import spi_slave_coverage_pkg::*;

    class spi_slave_environment extends uvm_env;
        `uvm_component_utils(spi_slave_environment)
        spi_slave_agent agt;
        spi_slave_scoreboard sb;
        spi_slave_coverage cov;

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

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            agt=spi_slave_agent::type_id::create("agt",this);
            sb=spi_slave_scoreboard::type_id::create("sb",this);
            cov=spi_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
```

AGENT

```
package spi_slave_agent_pkg;
    import uvm_pkg::*;
    `include"uvm_macros.svh"
    import spi_slave_driver_pkg::*;
    import spi_slave_sequencer_pkg::*;
    import spi_slave_monitor_pkg::*;
    import spi_slave_seq_item_pkg::*;
    import spi_slave_config_obj_pkg::*;

    class spi_slave_agent extends uvm_agent;
        `uvm_component_utils(spi_slave_agent)
        spi_slave_driver drv;
        spi_slave_sequencer sqr;
        spi_slave_monitor mon;
        spi_slave_config_obj cfg;
        uvm_analysis_port #(spi_slave_seq_item) agt_ap;

        function new (string name="spi_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 #(spi_slave_config_obj)::get(this,"","CFG",cfg))
                `uvm_fatal("build_phase","Error:Can't get config object")

            if(cfg.is_active == UVM_ACTIVE) begin
                drv=spi_slave_driver::type_id::create("drv",this);
                sqr=spi_slave_sequencer::type_id::create("sqr",this);
            end

            mon=spi_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(cfg.is_active == UVM_ACTIVE) begin
                drv.driver_vif=cfg.config_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end

            mon.vif=cfg.config_vif;
            mon.mon_ap.connect(agt_ap);
```

```
        endfunction
    endclass
endpackage
```

DRIVER

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

    class spi_slave_driver extends uvm_driver #(spi_slave_seq_item);
        `uvm_component_utils(spi_slave_driver)
        virtual spi_slave_if driver_vif;
        spi_slave_seq_item drv_seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                drv_seq_item=spi_slave_seq_item::type_id::create("drv_seq_item");
                seq_item_port.get_next_item(drv_seq_item);
                driver_vif.rst_n = drv_seq_item.rst_n;
                driver_vif.SS_n = drv_seq_item.SS_n;
                driver_vif.MOSI = drv_seq_item.MOSI;
                driver_vif.tx_valid = drv_seq_item.tx_valid;
                driver_vif.tx_data = drv_seq_item.tx_data;
                @(negedge driver_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase",drv_seq_item.convert2string_stimulus(),UVM_
HIGH);

            end

        endtask


    endclass

endpackage
```

MONITOR

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

    class spi_slave_monitor extends uvm_monitor;
         `uvm_component_utils(spi_slave_monitor);
         virtual spi_slave_if vif;
         spi_slave_seq_item mon_seq_item;
         uvm_analysis_port #(spi_slave_seq_item) mon_ap;

         function new (string name="spi_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
                mon_seq_item=spi_slave_seq_item::type_id::create("mon_seq_item");
                @(negedge vif.clk);
                mon_seq_item.rst_n = vif.rst_n;
                mon_seq_item.SS_n = vif.SS_n;
                mon_seq_item.MOSI = vif.MOSI;
                mon_seq_item.tx_valid = vif.tx_valid;
                mon_seq_item.tx_data = vif.tx_data;
                mon_seq_item.rx_data = vif.rx_data;
                mon_seq_item.rx_valid = vif.rx_valid;
                mon_seq_item.MISO = vif.MISO;
                mon_seq_item.rx_data_exp = vif.rx_data_exp;
                mon_seq_item.rx_valid_exp = vif.rx_valid_exp;
                mon_seq_item.MISO_exp = vif.MISO_exp;

                mon_ap.write(mon_seq_item);
            end
         endtask

    endclass
endpackage
```

SCOREBOARD

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

        class spi_slave_scoreboard extends uvm_scoreboard;
            `uvm_component_utils(spi_slave_scoreboard)
            uvm_analysis_export #(spi_slave_seq_item) sb_export;
            uvm_tlm_analysis_fifo #(spi_slave_seq_item) sb_fifo;
            spi_slave_seq_item seq_item_sb;

            int error_count=0;
            int correct_count=0;

            function new (string name="spi_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.MISO !== seq_item_sb.MISO_exp) ||
(seq_item_sb.rx_valid !== seq_item_sb.rx_valid_exp)
                    || (seq_item_sb.rx_data !== seq_item_sb.rx_data_exp)) begin
                        error_count++;
                        `uvm_error("run_phase",$sformatf("Error:Transaction:%s",s
eq_item_sb.convert2string));
                    end else begin
                        correct_count++;
                        `uvm_info("run_phase",$sformatf("Error:Transaction:%s",se
q_item_sb.convert2string),UVM_HIGH);
```

```
                    end
                end

            endtask


        endclass

    endpackage
```

COVERAGE COLLECTOR

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

    class spi_slave_coverage extends uvm_component;
        `uvm_component_utils(spi_slave_coverage)
        uvm_analysis_export #(spi_slave_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(spi_slave_seq_item) cov_fifo;
        spi_slave_seq_item seq_item_cov;

        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

        covergroup cvr_grp;
            SS_n_cp: coverpoint seq_item_cov.SS_n {option.weight=0;}
            MOSI_cp: coverpoint seq_item_cov.MOSI {option.weight=0;}
```

```
            //SLAVE_2, SLAVE_3, SLAVE_4, SLAVE_5
            rx_data_cp: coverpoint seq_item_cov.rx_data [9:8] {
                bins rx_data_val[]={[0:3]};
                bins rx_data_trans[]=([0:3]=>[0:3]);
                ignore_bins ignore_trans = (0 => 3), (3 => 0), (1 => 2), (2 =>
1);
            }

            //SLAVE_7
            SS_n_trans_cp: coverpoint seq_item_cov.SS_n {
                bins full_transaction     = (1=>0[*13]=>1);
                bins extended_transaction = (1=>0[*23]=>1);
                bins start_of_comm = (1 => 0 => 0);
            }

            //SLAVE_2, SLAVE_3, SLAVE_4, SLAVE_5
            mosi_trans_cp: coverpoint seq_item_cov.MOSI {
                bins wr_add_trans  = (0 => 0 => 0);
                bins wr_data_trans = (0 => 0 => 1);
                bins rd_add_trans  = (1 => 1 => 0);
                bins rd_data_trans = (1 => 1 => 1);
            }

            //SLAVE_7
            ss_mosi_cross_cvr: cross SS_n_trans_cp, mosi_trans_cp {
                option.cross_auto_bin_max = 0;

                // write operations
                bins WRITE_DATA_OP = binsof(SS_n_trans_cp.start_of_comm) &&
                                     binsof(mosi_trans_cp.wr_data_trans);
                bins WRITE_ADD_OP  = binsof(SS_n_trans_cp.start_of_comm) &&
                                     binsof(mosi_trans_cp.wr_add_trans);

                // read operations
                bins READ_ADD_OP  = binsof(SS_n_trans_cp.start_of_comm) &&
                                    binsof(mosi_trans_cp.rd_add_trans);
                bins READ_DATA_OP = binsof(SS_n_trans_cp.start_of_comm) &&
                                    binsof(mosi_trans_cp.rd_data_trans);
            }
        endgroup

        function new (string name="spi_slave_coverage", uvm_component
parent=null);
            super.new(name,parent);
```

```
            cvr_grp=new();
        endfunction
    endclass

endpackage
```

SEQUENCE ITEM

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

    class spi_slave_seq_item extends uvm_sequence_item;
        `uvm_object_utils(spi_slave_seq_item)

        rand bit   rst_n, SS_n, tx_valid;
        bit MOSI;
        rand bit  [7:0] tx_data;
        logic  [9:0] rx_data;
        logic  rx_valid, MISO;
        logic  rx_valid_exp, MISO_exp;
        logic  [9:0] rx_data_exp;
        int cycles_counter=0;
        int mosi_index=10;
        int max_cnt;
        bit transaction_ended=0;
        bit [2:0] prev_op;


        rand bit [10:0] mosi_arr;

        function new (string name="spi_slave_seq_item");
            super.new(name);
            transaction_ended=0;
            wr_or_rd_data.constraint_mode(0);
        endfunction

        function string convert2string();
            return $sformatf ("%sreset=%b|SS_n=%b|MOSI=%b|tx_valid=%b|tx_data=%b|
            rx_data=%b|rx_data_exp=%b|rx_valid=%b|rx_valid_exp=%b|MISO=%b|MISO_ex
p=%b"
            ,super.convert2string,rst_n,SS_n,MOSI,tx_valid,tx_data,rx_data,rx_dat
a_exp,rx_valid,rx_valid_exp,MISO,MISO_exp);
        endfunction
```

```
        function string convert2string_stimulus();
            return $sformatf ("%sreset=%b|SS_n=%b|MOSI=%b|tx_valid=%b|tx_data=%b"
            ,super.convert2string,rst_n,SS_n,MOSI,tx_valid,tx_data);
        endfunction

        function void post_randomize();
            if (~rst_n) begin
                transaction_ended=0;
                cycles_counter=0;
                mosi_index=10;
                mosi_arr[10:8]=0;
                wr_or_rd_data.constraint_mode(0);
                wr_or_rd_add.constraint_mode(1);
                mosi_arr.rand_mode(1);
            end else begin
                if(mosi_arr[10:8]==3'b111) begin
                    max_cnt=23;
                    tx_data.rand_mode(0);
                end else begin
                    max_cnt=13;
                    tx_data.rand_mode(1);
                end

              if(~SS_n)
                cycles_counter++;

                if (cycles_counter==max_cnt) begin
                    transaction_ended=1;
                    cycles_counter=0;
                    mosi_arr.rand_mode(1);

                    if (prev_op==3'b110) begin
                        wr_or_rd_add.constraint_mode(0);
                        wr_or_rd_data.constraint_mode(1);

                    end else if (prev_op==3'b111)begin
                        wr_or_rd_data.constraint_mode(0);
                        wr_or_rd_add.constraint_mode(1);
                    end
                    mosi_index=10;


                end
                else begin
                    transaction_ended=0;
```

```
                    prev_op=mosi_arr[10:8];
                    mosi_arr.rand_mode(0);
                end

                if(mosi_index>=0 && cycles_counter>=2 )begin
                       MOSI = mosi_arr[mosi_index];
                       mosi_index--;
                end
            end


        endfunction

        //SLAVE_1
        constraint rst_con {
           rst_n dist {0:=2, 1:=98};
        }

        //SLAVE_2,SLAVE_3,SLAVE_4
        constraint wr_or_rd_add {
             mosi_arr[10:8] inside {3'b000, 3'b001, 3'b110};
        }

        //SLAVE_2,SLAVE_3,SLAVE_5
        constraint wr_or_rd_data {
            mosi_arr[10:8] inside {3'b000, 3'b001, 3'b111};
        }



        //SLAVE_6
        constraint tx_read_data_con { if (mosi_arr[10:8]==3'b111 &&
cycles_counter>=13)
                                            tx_valid==1;
                                        else
                                            tx_valid==0;}

        //SLAVE_7
        constraint ss_n_con { if (transaction_ended==1)
                                SS_n==1;
                              else
                                SS_n==0;
        }
```

```
    endclass

endpackage
```

SEQUENCE

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

    class spi_slave_main_seq extends uvm_sequence #(spi_slave_seq_item);
        `uvm_object_utils(spi_slave_main_seq);
        spi_slave_seq_item seq_item;

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

        task body();
            seq_item = spi_slave_seq_item::type_id::create("seq_item");

            repeat(2000)begin
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
            end
        endtask
    endclass

endpackage
```

SEQUENCER

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

    class spi_slave_sequencer extends uvm_sequencer #(spi_slave_seq_item);
            `uvm_component_utils(spi_slave_sequencer);

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

    endclass

endpackage
```

CONFIGURATION OBJECT

```
package spi_slave_config_obj_pkg;
    import uvm_pkg::*;
    `include"uvm_macros.svh"
    class spi_slave_config_obj extends uvm_object;
        `uvm_object_utils(spi_slave_config_obj)
        virtual spi_slave_if config_vif;
        uvm_active_passive_enum is_active;

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

DO FILE

```
vlib work
vlog -f src_files.list +define+SIM +cover -covercells
vsim -voptargs=+acc work.spi_slave_top -classdebug -uvmcontrol=all -cover
add wave /spi_slave_top/SLAVEif/*
coverage save SLAVE.ucdb -onexit
coverage report -detail -cvg -directive -comments -output slave_report.txt {}
run 0
add wave -position insertpoint  \
```

```
sim:/uvm_root/uvm_test_top/main_seq.seq_item
do wave.do
run -all
#vcover report SLAVE.ucdb -details -annotate -all -output Slave_coverage.txt
```

WAVE DO FILE

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider {SLAVE Interface}
add wave -noupdate /spi_slave_top/SLAVEif/clk
add wave -noupdate /spi_slave_top/SLAVEif/rst_n
add wave -noupdate /spi_slave_top/SLAVEif/SS_n
add wave -noupdate /spi_slave_top/SLAVEif/MOSI
add wave -noupdate /spi_slave_top/SLAVEif/rx_data
add wave -noupdate /spi_slave_top/SLAVEif/rx_data_exp
add wave -noupdate /spi_slave_top/SLAVEif/rx_valid
add wave -noupdate /spi_slave_top/SLAVEif/rx_valid_exp
add wave -noupdate /spi_slave_top/SLAVEif/tx_valid
add wave -noupdate /spi_slave_top/SLAVEif/tx_data
add wave -noupdate /spi_slave_top/SLAVEif/MISO
add wave -noupdate /spi_slave_top/SLAVEif/MISO_exp
add wave -noupdate /spi_slave_top/DUT/cs
add wave -noupdate /spi_slave_top/GM/cs
add wave -noupdate -divider Assertions
add wave -noupdate
/spi_slave_main_seq_pkg::spi_slave_main_seq::body/#ublk#223710487#17/immed__19
add wave -noupdate /spi_slave_top/DUT/assert__chk_to_states_trans
add wave -noupdate /spi_slave_top/DUT/assert__p_idle_chk_trans
add wave -noupdate /spi_slave_top/DUT/assert__p_to_idle_trans
add wave -noupdate /spi_slave_top/DUT/spi_sva_inst/assert__p_reset
add wave -noupdate /spi_slave_top/DUT/spi_sva_inst/assert__p_rx_valid_rd_add
add wave -noupdate /spi_slave_top/DUT/spi_sva_inst/assert__p_rx_valid_rd_data
add wave -noupdate /spi_slave_top/DUT/spi_sva_inst/assert__p_rx_valid_wr_add
add wave -noupdate /spi_slave_top/DUT/spi_sva_inst/assert__p_rx_valid_wr_data
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {3951 ns} 0}
quietly wave cursor active 1
configure wave -namecolwidth 218
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
```

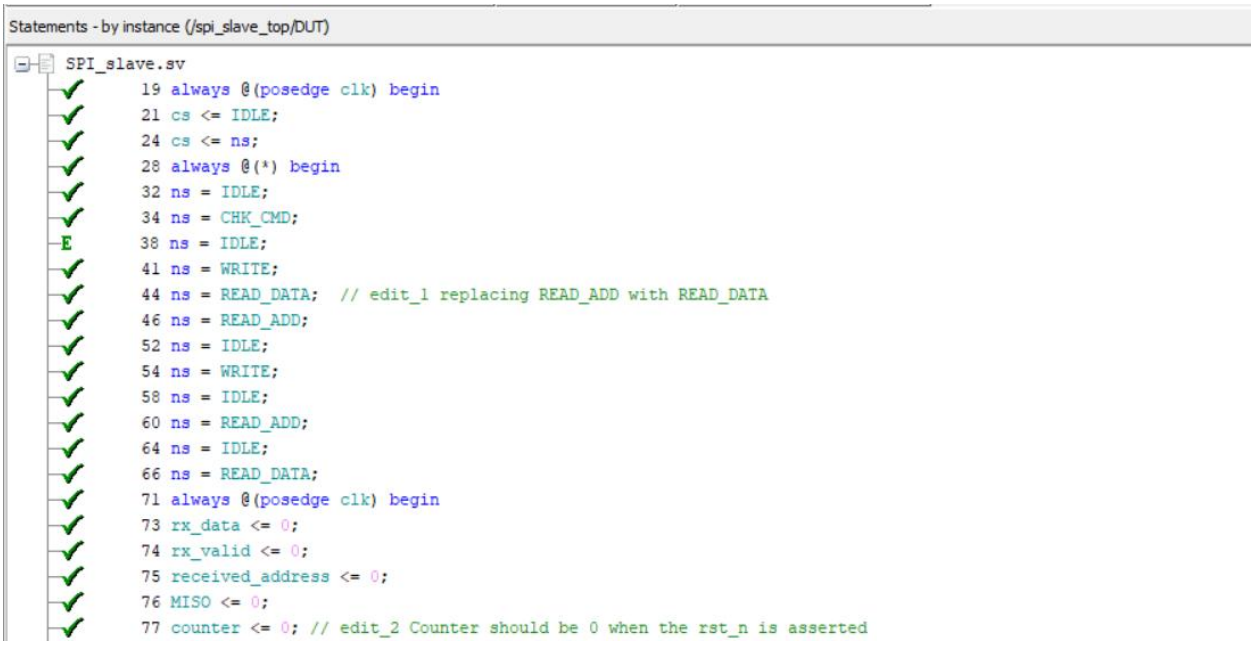
```
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {3934 ns} {4002 ns}
```

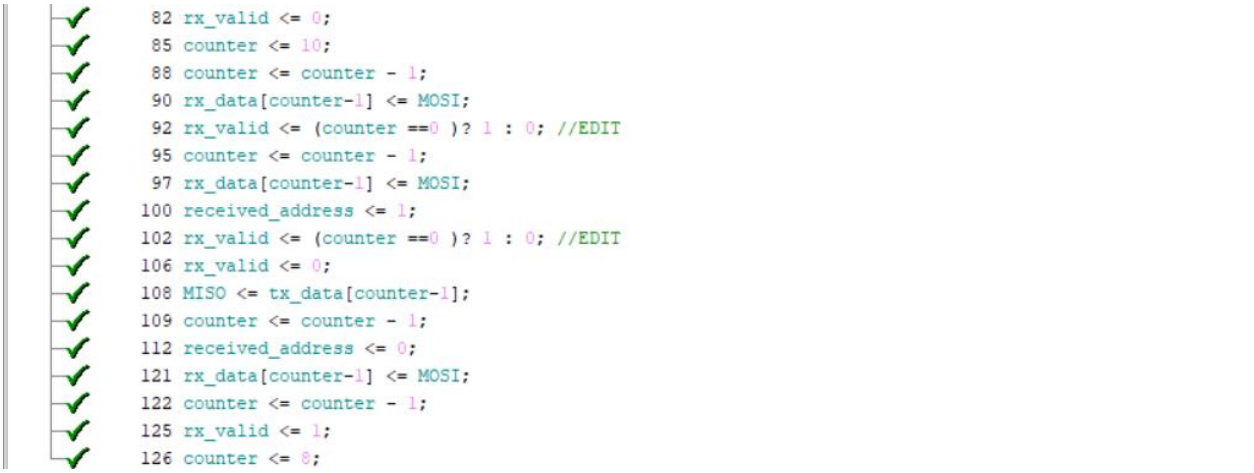
SOURCE FILES LIST

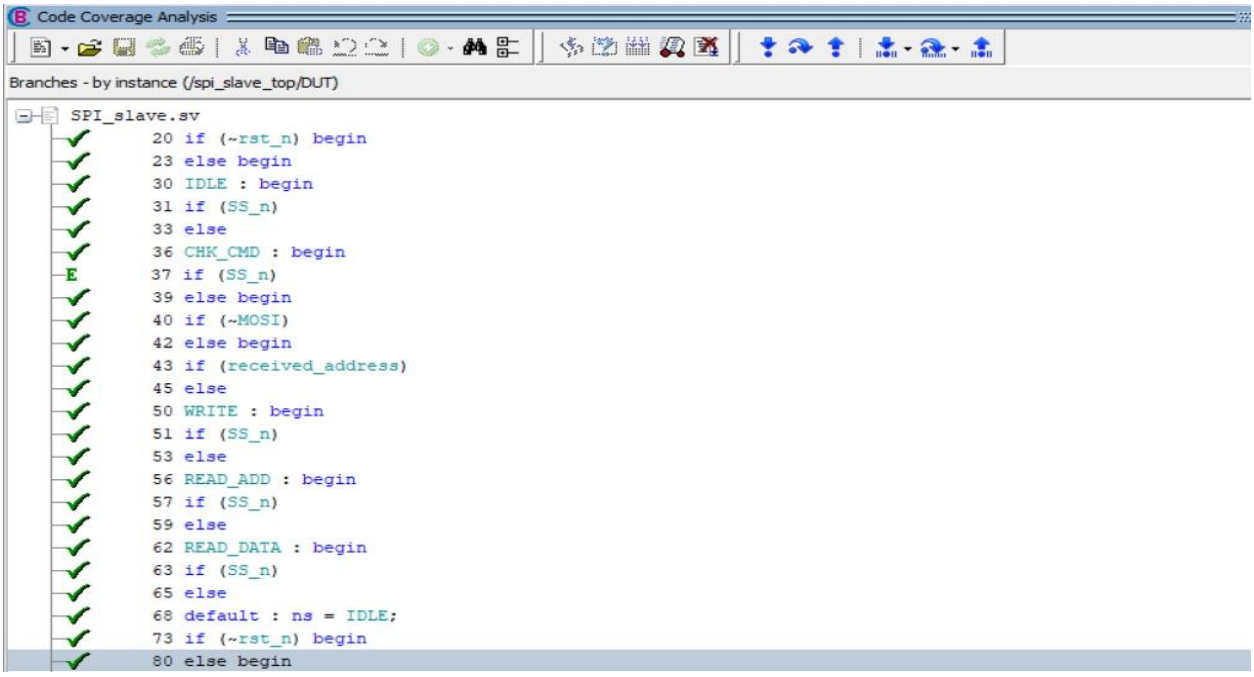
```
SPI_slave.sv
SPI_slave.v
SPI_slave_agent.sv
SPI_slave_cfg.sv
SPI_slave_cov.sv
SPI_slave_driver.sv
SPI_slave_if.sv
SPI_slave_sva.sv
SPI_slave_main_seq.sv
SPI_slave_monitor.sv
SPI_slave_rst_seq.sv
SPI_slave_scoreboard.sv
SPI_slave_seq_item.sv
SPI_slave_sequencer.sv
SPI_slave_test.sv
SPI_slave_top.sv
SPI_slave_env.sv
```

COVERAGE REPORTS

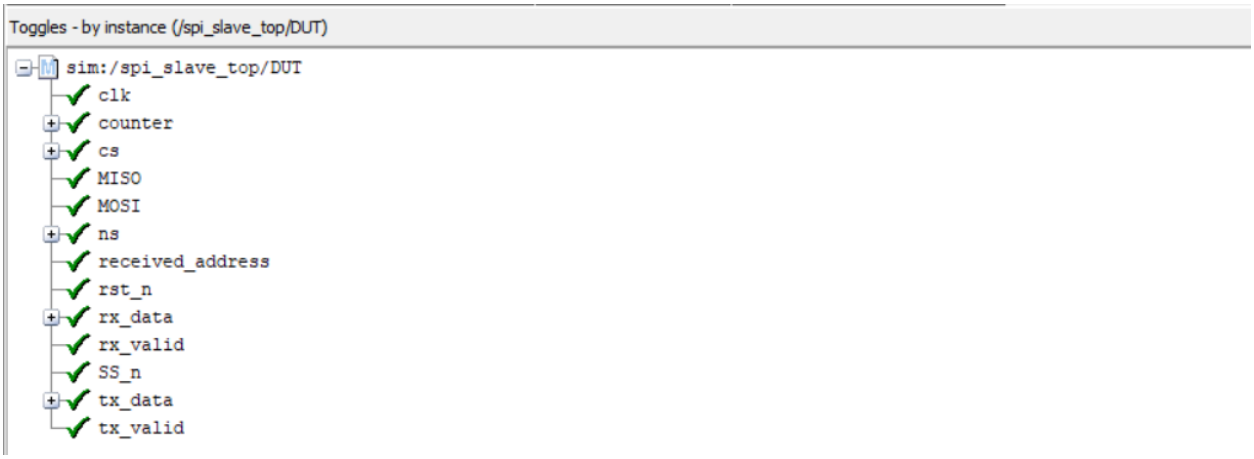
CODE COVERAGE REPORT



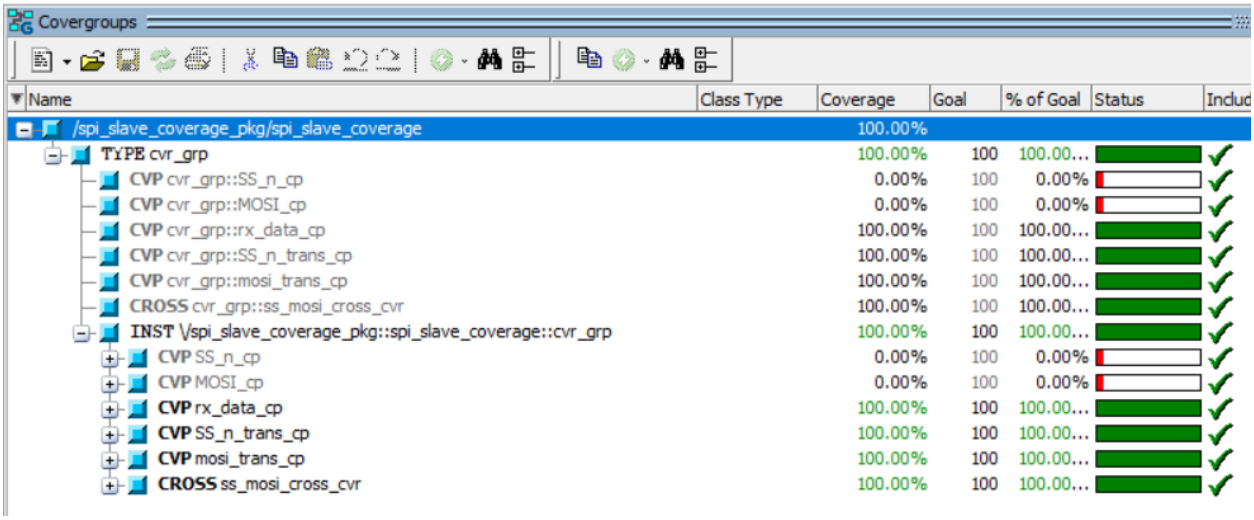



Code Coverage Analysis
Branches - by instance (/spi_slave_top/DUT)
SPI_slave.sv
20 if (~rst_n) begin
23 else begin
30 IDLE : begin
31 if (SS_n)
33 else
36 CHK_CMD : begin
E 37 if (SS_n)
39 else begin
40 if (~MOSI)
42 else begin
43 if (received_address)
45 else
50 WRITE : begin
51 if (SS_n)
53 else
56 READ_ADD : begin
57 if (SS_n)
59 else
62 READ_DATA : begin
63 if (SS_n)
65 else
68 default : ns = IDLE;
73 if (~rst_n) begin
80 else begin

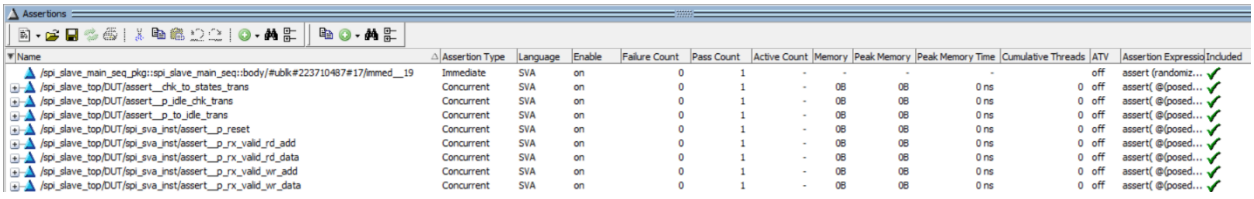

82 IDLE : begin
85 CHK_CMD : begin
88 WRITE : begin
90 if (counter > 0) begin
93 rx_valid <= (counter ==0)? 1 : 0; //EDIT
95 READ_ADD : begin
97 if (counter > 0) begin
100 else begin
103 rx_valid <= (counter ==0)? 1 : 0; //EDIT
105 READ_DATA : begin
106 if (tx_valid) begin
108 if (counter > 0) begin
112 else begin
116 else begin
121 if(counter > 0 && ~rx_valid) begin
125 else begin
E 131 default : begin

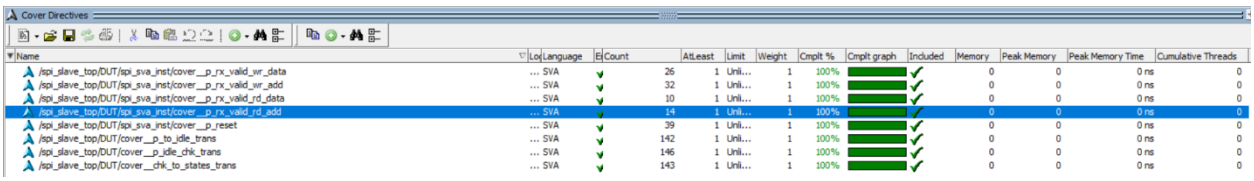


FUNCTIONAL COVERAGE REPORT



SEQUENTIAL DOMAIN COVERAGE REPORT





BUG REPORT

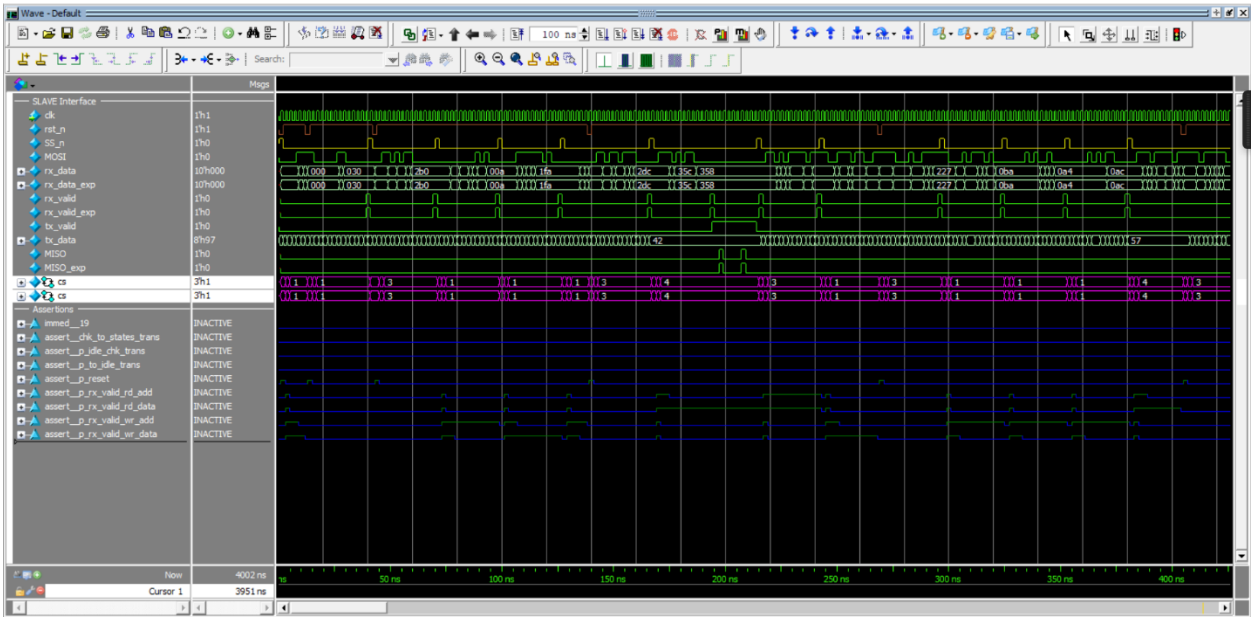
Case	Expected behaviour	Output	Edit

When entering READ_DATA or READ_ADD states, they will be replaced because their received flag condition is replaced.	Entering the READ_ADD state first then READ_DATA to get the stored data.	Entering the READ_DATA first which results in an output of unkonwns.	`if (received_address)`
			`ns = READ_DATA; // edit_1 replacing READ_ADD with READ_DATA`
			`else`
			`ns = READ_ADD;`

When the rst_n is asserted low the counter doesn't return back to zero	Counter == 0	Counter == the old value	`counter <= 0; // edit_2 Counter should be 0 when the rst_n is asserted`
rx_valid should be raised for only one clock cycle in all the cases except READ_DATA state	rx_valid is raised and fall again	rx_valid is raised for two clock cycles and fall again	`rx_valid <= (counter ==0 )? 1:0; //EDIT`

In READ_DATA state, when getting the data from the master and raising the rx_valid the counter is set to 8 and starts to decrements again reading values again from the MOSI	Counter == 8 until the tx_valid rises	Counter == 7 and the rx_valid reads an extra bit from the MOSI until the tx_valid rises	`// edit_3 when the counter is set to 8`
			`// the rx_data continues reading again.`
			`// Adding the condition of NOT rx_valid will fix it`
			`if (counter > 0 && ~rx_valid)`
			`begin`

WAVEFORM


Wave - Default
SLAVE Interface
clk
rst_n
SS_n
MOSI
rx_data
rx_data_exp
rx_valid
rx_valid_exp
tx_valid
tx_data
MISO
MISO_exp
cs
cs
Assertions
Now
Cursor 1

ASSERTIONS

Feature	Assertion
whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.	property p_reset; @(posedge clk) (~rst_n) |=> (MISO==0 && rx_data==0 && rx_valid==0); endproperty assert property (p_reset);
After any write_add command sequence (000), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.	property p_rx_valid_wr_add; @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(~MOSI)[*3]) \|-> ##10 ($rose(rx_valid) && SS_n[- >1]); endproperty assert property(p_rx_valid_wr_add); cover property (p_rx_valid_wr_add);
After any write_data command sequence (001), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.	property p_rx_valid_wr_data; @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(~MOSI)[*2] ##1 (MOSI)) \|-> ##10 ($rose(rx_valid) && SS_n[->1]); endproperty assert property(p_rx_valid_wr_data); cover property (p_rx_valid_wr_data);
After any read_add command sequence (110), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.	property p_rx_valid_rd_add; @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(MOSI)[*2] ##1 (~MOSI)) \|-> ##10 ($rose(rx_valid) && SS_n[->1]); endproperty assert property(p_rx_valid_rd_add); cover property (p_rx_valid_rd_add);

After any read_data command sequence (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 p_rx_valid_rd_data; @(posedge clk) disable iff (~rst_n) ($fell(SS_n)##1(MOSI)[*3]) |-> ##10 ($rose(rx_valid) && SS_n[->1]); endproperty assert property(p_rx_valid_rd_data); cover property (p_rx_valid_rd_data);
When the FSM transitions from the IDLE state, it must move only to the CHK_CMD state.	property p_idle_chk_trans; @(posedge clk) disable iff (~rst_n) ((!$stable(cs)) && $past(cs)==IDLE) |-> (cs==CHK_CMD); endproperty assert property (p_idle_chk_trans); cover property (p_idle_chk_trans);
After the FSM completes the CHK_CMD state, the next valid states are READ_ADD, WRITE, or READ_DATA.	property chk_to_states_trans; @(posedge clk) disable iff (~rst_n) ((!$stable(cs)) && (cs==READ_ADD \|\| cs==WRITE \|\| cs==READ_DATA)) |-> ($past(cs)==CHK_CMD); endproperty assert property (chk_to_states_trans); cover property (chk_to_states_trans);
After the FSM finishes any of the operational states WRITE, READ_ADD, or READ_DATA, it must transition back to the IDLE state.	property p_to_idle_trans; @(posedge clk) disable iff (~rst_n) ((!$stable(cs)) && ($past(cs)==WRITE || $past(cs)==READ_ADD || $past(cs)==READ_DATA)) |-> (cs==IDLE); endproperty assert property (p_to_idle_trans); cover property (p_to_idle_trans);

TRANSCRIPT


Transcript
#
(Specify +UVM_NO_RELNOTES to turn off this notice)
#
UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM] questa_uvm::init(all)
UVM_INFO @ 0: reporter [RNTST] Running test spi_slave_test...
** Warning: In instance '\/spi_slave_coverage_pkg::spi_slave_coverage::cvr_grp ' the 'option.per_instance' is set to '0' (false). Assignment to members 'weight', 'goal' and
 effect.
**
* Questa UVM Transaction Recording Turned ON.   *
* recording_detail has been set.   *
* To turn off, set 'recording_detail' to off: *
* uvm_config_db#(int)   ::set(null, "", "recording_detail", 0); *
* uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
**
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 4002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
#
--- UVM Report Summary ---
#
** Report counts by severity
UVM_INFO :   4
UVM_WARNING :   0
UVM_ERROR :   0
UVM_FATAL :   0
** Report counts by id
[Questa UVM]   2
[RNTST]   1
[TEST_DONE]   1
** Note: $finish   : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
Time: 4002 ns Iteration: 61 Instance: /spi_slave_top
Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

Part Two - RAM

CODE SNIPPETS

DESIGN

```
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[Rd_Addr];   // edit_1 Rd_Addr not Wr_Addr
            default : dout <= 0;
        endcase
        tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; // edit_2 -> only if reset
signal is deasserted
     end
     // tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; // edit_2
end
endmodule
```

GOLDEN MODEL

```
module RAM_ref #(
    parameter MEM_DEPTH = 256,
    parameter ADDR_SIZE = 8
)(
    input  [9:0] din,
    input  clk,
    input  rst_n,       // synchronous active low
    input  rx_valid,
    output reg [7:0] dout,
    output reg tx_valid
);
    reg [7:0] mem [MEM_DEPTH-1:0];
    reg [7:0] wr_addr, rd_addr;
    always @(posedge clk) begin
        if(~rst_n) begin
            dout     <= 0;
            tx_valid <= 0;
            wr_addr  <= 0;
            rd_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[rd_addr];
            endcase
            tx_valid <= ((din[9:8] == 2'b11) && rx_valid);
        end
    end
endmodule
```

INTERFACE

```
import ram_pkg::*;

interface RAM_if(input clk);

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

    logic  [9:0] din;
    logic  rst_n;       // synchronous active low
    logic  rx_valid;
```

```
    logic  tx_valid;
    logic  [7:0] dout;
    logic  [7:0] dout_ref;
    logic  tx_valid_ref;

endinterface : RAM_if
```

SVA

```
import ram_pkg::*;

module RAM_sva(
    input  [9:0] din,
    input  clk,
    input  rst_n,
    input  rx_valid,
    input reg [7:0] dout,
    input reg tx_valid
    );
// Reset
a_rst_dout: assert property(@(posedge clk) ~rst_n |=> dout == 0);
c_rst_dout: cover property(@(posedge clk) ~rst_n |=> dout == 0);
a_rst_tx_valid: assert property(@(posedge clk) ~rst_n |=> tx_valid == 0);
c_rst_tx_valid: cover property(@(posedge clk) ~rst_n |=> tx_valid == 0);

// tx_valid output
a_tx_low: assert property(@(posedge clk) disable iff(~rst_n)
    din[9:8] inside {WRITE_ADDRESS, WRITE_DATA, READ_ADDRESS} && rx_valid
    |=> ~tx_valid || $fell(tx_valid));
c_tx_low: cover property(@(posedge clk) disable iff(~rst_n)
    din[9:8] inside {WRITE_ADDRESS, WRITE_DATA, READ_ADDRESS} && rx_valid
    |=> ~tx_valid);

a_tx_high: assert property(@(posedge clk) disable iff(~rst_n)
    (din[9:8] == READ_DATA) && rx_valid |=> tx_valid ##1 $fell(tx_valid) [->1]);
c_tx_high: cover property(@(posedge clk) disable iff(~rst_n)
    (din[9:8] == READ_DATA) && rx_valid |=> tx_valid ##1 $fell(tx_valid) [->1]);

// RAM modes
a_write: assert property(@(posedge clk) disable iff(~rst_n)
    din[9:8] == WRITE_ADDRESS |=> (din[9:8] == WRITE_DATA) [->1]);
c_write: cover property(@(posedge clk) disable iff(~rst_n)
    din[9:8] == WRITE_ADDRESS |=> (din[9:8] == WRITE_DATA) [->1]);
```

```
a_read: assert property(@(posedge clk) disable iff(~rst_n)
    din[9:8] == READ_ADDRESS |=> (din[9:8] == READ_DATA) [->1]);
c_read: cover property(@(posedge clk) disable iff(~rst_n)
    din[9:8] == READ_ADDRESS |=> (din[9:8] == READ_DATA) [->1]);

endmodule : RAM_sva
```

TOP MODULE

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

module RAM_top();
    bit clk;
    initial begin
        clk=0;
        forever #1 clk=~clk;
    end

    RAM_if  ram_if(.clk(clk));
    RAM     DUT(.clk(ram_if.clk),.rst_n(ram_if.rst_n),.din(ram_if.din),
                .rx_valid(ram_if.rx_valid),.tx_valid(ram_if.tx_valid),
                .dout(ram_if.dout));
    RAM_ref ram_ref(.clk(ram_if.clk),.rst_n(ram_if.rst_n),.din(ram_if.din),
                .rx_valid(ram_if.rx_valid),.tx_valid(ram_if.tx_valid_ref),
                .dout(ram_if.dout_ref));
    bind RAM RAM_sva ram(.clk(ram_if.clk),.rst_n(ram_if.rst_n),.din(ram_if.din),
                .rx_valid(ram_if.rx_valid),.tx_valid(ram_if.tx_valid),
                .dout(ram_if.dout));

    initial begin
        uvm_config_db#(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF",
ram_if);
        run_test("ram_test");
    end
endmodule : RAM_top
```

TEST

```
package ram_test_pkg;
package ram_test_pkg;
    import uvm_pkg::*;
    import ram_enviroment_pkg::*;
    import ram_config_obj_pkg::*;
```

```
    import ram_sequence_pkg::*;

    `include "uvm_macros.svh"

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

        ram_enviroment env;
        ram_config_obj cfg_ram;
        ram_rst_seq rst_seq;
        ram_write_only_seq wr_seq;
        ram_read_only_seq rd_seq;
        ram_write_read_seq wr_rd_seq;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env = ram_enviroment::type_id::create("env",this);
            cfg_ram = ram_config_obj::type_id::create("cfg_ram");
            rst_seq = ram_rst_seq::type_id::create("rst_seq");
            wr_seq = ram_write_only_seq::type_id::create("wr_seq");
            rd_seq = ram_read_only_seq::type_id::create("rd_seq");
            wr_rd_seq = ram_write_read_seq::type_id::create("wr_rd_seq");
            if(!uvm_config_db#(virtual RAM_if)::get(this, "", "RAM_IF",
cfg_ram.ram_cfg_vif))
                `uvm_fatal("build_phase","Unable to get the ram virtual interface
- test");
            cfg_ram.is_active = UVM_ACTIVE;
            uvm_config_db#(ram_config_obj)::set(this, "*", "CFG_RAM", cfg_ram);
        endfunction

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

            // RAM_1
            `uvm_info("run_phase","Reset Asserted.",UVM_LOW);
            rst_seq.start(env.agent.sqr);
            `uvm_info("run_phase","Reset Deasserted.",UVM_LOW);

            // RAM_2
```

```
            `uvm_info("run_phase","Write Only Stimulus Generation
Started.",UVM_LOW);
            wr_seq.start(env.agent.sqr);
            `uvm_info("run_phase","Write Only Stimulus Generation
Ended.",UVM_LOW);

            // RAM_3
            `uvm_info("run_phase","Read Only Stimulus Generation
Started.",UVM_LOW);
            rd_seq.start(env.agent.sqr);
            `uvm_info("run_phase","Read Only Stimulus Generation
Ended.",UVM_LOW);

            // RAM_4
            `uvm_info("run_phase","Write/Read Stimulus Generation
Started.",UVM_LOW);
            wr_rd_seq.start(env.agent.sqr);
            `uvm_info("run_phase","Write/Read Stimulus Generation
Ended.",UVM_LOW);
            phase.drop_objection(this);
        endtask : run_phase
    endclass : ram_test
endpackage : ram_test_pkg
```

ENVIRONMENT

```
package ram_enviroment_pkg;
    import uvm_pkg::*;
    import ram_agent_pkg::*;
    import ram_coverage_pkg::*;
    import ram_scoreboard_pkg::*;

    `include "uvm_macros.svh"

    class ram_enviroment extends uvm_env;
        `uvm_component_utils(ram_enviroment);

        ram_agent agent;
        ram_scoreboard sb;
        ram_coverage cov;

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

```
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agent = ram_agent::type_id::create("agent",this);
            sb = ram_scoreboard::type_id::create("sb",this);
            cov = ram_coverage::type_id::create("cov",this);
        endfunction : build_phase

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

    endclass : ram_enviroment
endpackage : ram_enviroment_pkg
```

AGENT

```
package ram_agent_pkg;
    import uvm_pkg::*;
    import ram_driver_pkg::*;
    import ram_config_obj_pkg::*;
    import ram_monitor_pkg::*;
    import ram_sequencer_pkg::*;
    import ram_sequence_item_pkg::*;
    `include "uvm_macros.svh"

    class ram_agent extends uvm_agent;
        `uvm_component_utils(ram_agent);

        ram_driver      drv;
        ram_sequencer   sqr;
        ram_monitor     mon;
        ram_config_obj  cfg;
        uvm_analysis_port #(ram_sequence_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_obj)::get(this,"","CFG_RAM",cfg))
```

```
                `uvm_fatal("build_phase","Unable to get configuration object:
AGENT");
            if(cfg.is_active == UVM_ACTIVE) begin
                sqr = ram_sequencer::type_id::create("sqr",this);
                drv = ram_driver::type_id::create("srv",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(cfg.is_active == UVM_ACTIVE) begin
            drv.ram_driver_vif = cfg.ram_cfg_vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
        end
        mon.ram_monitor_vif = cfg.ram_cfg_vif;
                mon.mon_ap.connect(agt_ap);
    endfunction

endclass
endpackage
```

DRIVER

```
package ram_driver_pkg;
    import uvm_pkg::*;
    import ram_config_obj_pkg::*;
    import ram_sequence_item_pkg::*;
    `include "uvm_macros.svh"

    class ram_driver extends uvm_driver #(ram_sequence_item);
        `uvm_component_utils(ram_driver);

        virtual RAM_if ram_driver_vif;
        ram_sequence_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_sequence_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                ram_driver_vif.rst_n     =stim_seq_item.rst_n;
                ram_driver_vif.din       =stim_seq_item.din;
                ram_driver_vif.rx_valid  =stim_seq_item.rx_valid;
                @(negedge ram_driver_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM
_HIGH)
            end
        endtask : run_phase

    endclass : ram_driver
endpackage : ram_driver_pkg
```

MONITOR

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

    class ram_monitor extends uvm_monitor;
        `uvm_component_utils(ram_monitor)
        virtual RAM_if ram_monitor_vif;
        ram_sequence_item rsp_seq_item;
        uvm_analysis_port #(ram_sequence_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 : build_phase

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                rsp_seq_item =
ram_sequence_item::type_id::create("rsp_seq_item");
                @(negedge ram_monitor_vif.clk);
                rsp_seq_item.rst_n = ram_monitor_vif.rst_n;
```

```
                rsp_seq_item.din = ram_monitor_vif.din;
                rsp_seq_item.rx_valid = ram_monitor_vif.rx_valid;
                rsp_seq_item.tx_valid = ram_monitor_vif.tx_valid;
                rsp_seq_item.tx_valid_ref = ram_monitor_vif.tx_valid_ref;
                rsp_seq_item.dout = ram_monitor_vif.dout;
                rsp_seq_item.dout_ref = ram_monitor_vif.dout_ref;

                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
            end
        endtask : run_phase
    endclass : ram_monitor
endpackage : ram_monitor_pkg
```

SCOREBOARD

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

    class ram_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(ram_scoreboard)
        uvm_analysis_export #(ram_sequence_item) sb_export;
        uvm_tlm_analysis_fifo #(ram_sequence_item) sb_fifo;
        ram_sequence_item sb_seq_item;

        int correct_count = 0;
        int error_count = 0;

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

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

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

```
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(sb_seq_item);
                if(sb_seq_item.dout === sb_seq_item.dout_ref
                    && sb_seq_item.tx_valid == sb_seq_item.tx_valid_ref)
                        correct_count++;
                else
                        error_count++;
            end
        endtask : run_phase

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

    endclass
endpackage : ram_scoreboard_pkg
```

COVERAGE COLLECTOR

```
package ram_coverage_pkg;
    import uvm_pkg::*;
    import ram_sequence_item_pkg::*;
    import ram_pkg::*;
    `include "uvm_macros.svh"

    class ram_coverage extends uvm_component;
        `uvm_component_utils(ram_coverage)

        uvm_analysis_export #(ram_sequence_item) cov_export;
        uvm_tlm_analysis_fifo #(ram_sequence_item) cov_fifo;
        ram_sequence_item cov_seq_item;

        covergroup cvr_gp;
        din_cp: coverpoint cov_seq_item.din[9:8]{
            bins din_wr_add   = {WRITE_ADDRESS};
            bins din_wr_data  = {WRITE_DATA};
            bins din_rd_add   = {READ_ADDRESS};
            bins din_rd_data  = {READ_DATA};
```

```
            bins din_wr_trans = (WRITE_ADDRESS => WRITE_DATA);
            bins din_rd_trans = (READ_ADDRESS =>  READ_DATA);
            bins din_trans    = (WRITE_ADDRESS =>  WRITE_DATA => READ_ADDRESS =>
READ_DATA);
        }
        rx: coverpoint  cov_seq_item.rx_valid{
        option.weight = 0;
        }
        tx: coverpoint cov_seq_item.tx_valid{
        option.weight = 0;
        }
        rx_cross: cross din_cp, rx{
            ignore_bins rx_ignore = binsof(rx) intersect {0};
            ignore_bins trans_ignore = binsof(din_cp.din_wr_trans) ||
binsof(din_cp.din_rd_trans) || binsof(din_cp.din_trans);
        }
        tx_cross: cross din_cp, tx{
            option.cross_auto_bin_max = 0;
            bins read_data_tx_high = binsof(din_cp.din_rd_data) && binsof(tx)
intersect {1};
        }
        endgroup : cvr_gp

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

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

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

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

```
    endtask
endclass
endpackage
```

SEQUENCE ITEM

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

    class ram_sequence_item extends uvm_sequence_item;
        `uvm_object_utils(ram_sequence_item)
        rand logic  [9:0] din;
        rand logic  rst_n;
        rand logic  rx_valid;
        logic       tx_valid;
        logic       [7:0] dout;
        logic       [7:0] dout_ref;
        logic       tx_valid_ref;
        e_mode      prev;

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

        function string convert2string();
            return $sformatf("%s",
                super.convert2string());
        endfunction : convert2string

        function string convert2string_stimulus();
            return $sformatf(" ");
        endfunction

        constraint c_reset {
            rst_n dist {1:/98, 0:/2};
        }
        constraint c_rx {
            rx_valid dist {1:/90, 0:/10};
        }
        constraint c_write_only {
            (prev == WRITE_ADDRESS) ->
                (din[9:8] inside {WRITE_ADDRESS,WRITE_DATA});
```

```
            (prev == WRITE_DATA) -> (din[9:8] == WRITE_ADDRESS);
        }
        constraint c_read_only {
            (prev == READ_ADDRESS) -> (din[9:8] == READ_DATA);
            (prev == READ_DATA) -> (din[9:8] == READ_ADDRESS);
        }
        constraint c_write_read {
            (prev == WRITE_ADDRESS) ->
                (din[9:8] inside {WRITE_ADDRESS,WRITE_DATA});
            (prev == READ_ADDRESS) -> (din[9:8] == READ_DATA);
            (prev == WRITE_DATA) ->
                din[9:8] dist {READ_ADDRESS:/60, WRITE_ADDRESS:/40};
            (prev == READ_DATA) ->
                din[9:8] dist {WRITE_ADDRESS:/60, READ_ADDRESS:/40};
        }

        function void post_randomize();
            prev = e_mode'(din[9:8]);
        endfunction : post_randomize

endclass : ram_sequence_item
endpackage
```

SEQUENCE

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

    logic[1:0] prev_mode = WRITE_ADDRESS;

    class ram_rst_seq extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_rst_seq)
        ram_sequence_item seq_item;

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

        task body();
            seq_item = ram_sequence_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst_n = 0;
```

```
            finish_item(seq_item);
        endtask : body
    endclass : ram_rst_seq

    class ram_write_only_seq extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_write_only_seq)
        ram_sequence_item seq_item;

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

        task body();
            repeat(1000) begin
                seq_item = ram_sequence_item::type_id::create("seq_item");
                start_item(seq_item);
                    seq_item.prev = e_mode'(prev_mode);
                    seq_item.c_write_only.constraint_mode(1);
                    seq_item.c_read_only.constraint_mode(0);
                    seq_item.c_write_read.constraint_mode(0);
                    assert(seq_item.randomize());
                    prev_mode = seq_item.din[9:8];
                finish_item(seq_item);
            end
        endtask : body
    endclass : ram_write_only_seq

    class ram_read_only_seq extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_read_only_seq)
        ram_sequence_item seq_item;

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

        task body();
            repeat(1000) begin
                seq_item = ram_sequence_item::type_id::create("seq_item");
                start_item(seq_item);
                    seq_item.prev = e_mode'(prev_mode);
                    seq_item.c_write_only.constraint_mode(0);
                    seq_item.c_read_only.constraint_mode(1);
                    seq_item.c_write_read.constraint_mode(0);
                    assert(seq_item.randomize());
                    prev_mode = seq_item.din[9:8];
```

```
                finish_item(seq_item);
            end
        endtask : body
    endclass : ram_read_only_seq

    class ram_write_read_seq extends uvm_sequence #(ram_sequence_item);
        `uvm_object_utils(ram_write_read_seq)
        ram_sequence_item seq_item;

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

        task body();
            repeat(1000) begin
                seq_item = ram_sequence_item::type_id::create("seq_item");
                start_item(seq_item);
                    seq_item.prev = e_mode'(prev_mode);
                    seq_item.c_read_only.constraint_mode(0);
                    seq_item.c_write_only.constraint_mode(0);
                    seq_item.c_write_read.constraint_mode(1);
                    assert(seq_item.randomize());
                    prev_mode = seq_item.din[9:8];
                finish_item(seq_item);
            end
        endtask : body
    endclass : ram_write_read_seq

endpackage : ram_sequence_pkg
```

SEQUENCER

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

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

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

```
endpackage : ram_sequencer_pkg
```

Configuration Object

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

    class ram_config_obj extends uvm_object;
        `uvm_object_utils(ram_config_obj);

        virtual RAM_if ram_cfg_vif;
        uvm_active_passive_enum is_active;

        function new(string name = "ram_config");
            super.new(name);
        endfunction : new
    endclass : ram_config_obj

endpackage : ram_config_obj_pkg
```

Shared Package

```
package ram_pkg;
    typedef enum logic[1:0] {WRITE_ADDRESS = 2'b00, WRITE_DATA = 2'b01,
                             READ_ADDRESS = 2'b10, READ_DATA = 2'b11} e_mode;
endpackage : ram_pkg
```

Do File

```
quit -sim
vdel -all
vlib work
vlog -f src_files.list +cover
vsim -voptargs=+acc work.RAM_top -classdebug -uvmcontrol=all -cover
run 0
add wave -group "RAM Interface" /RAM_top/ram_if/*
add wave -group "Scoreboard Counters"
sim:/uvm_root/uvm_test_top/env/sb/error_count \
sim:/uvm_root/uvm_test_top/env/sb/correct_count
add wave -group "Assertions" /RAM_top/DUT/ram/a_rst_dout\
/RAM_top/DUT/ram/a_rst_tx_valid \
/RAM_top/DUT/ram/a_tx_low \
/RAM_top/DUT/ram/a_tx_high \
/RAM_top/DUT/ram/a_write \
/RAM_top/DUT/ram/a_read
```

```
run -all
coverage exclude -src RAM.v -line 26 -code s
coverage exclude -src RAM.v -line 26 -code b
coverage save RAM_top.ucdb

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

SOURCE FILES LIST

```
RAM.v
RAM_ref.v
ram_pkg.sv
RAM_if.sv
RAM_sva.sv
ram_config_obj.sv
ram_sequence_item.sv
ram_sequence.sv
ram_sequencer.sv
ram_driver.sv
ram_monitor.sv
ram_agent.sv
ram_scoreboard.sv
ram_coverage.sv
ram_enviroment.sv
ram_test.sv
ram_top.sv
```

COVERAGE REPORTS

CODE COVERAGE REPORT

Statement Coverage

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

Branch Coverage

```
=== Instance: /RAM_top/DUT
=== Design Unit: work.RAM
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         7         7         0   100.00%
```

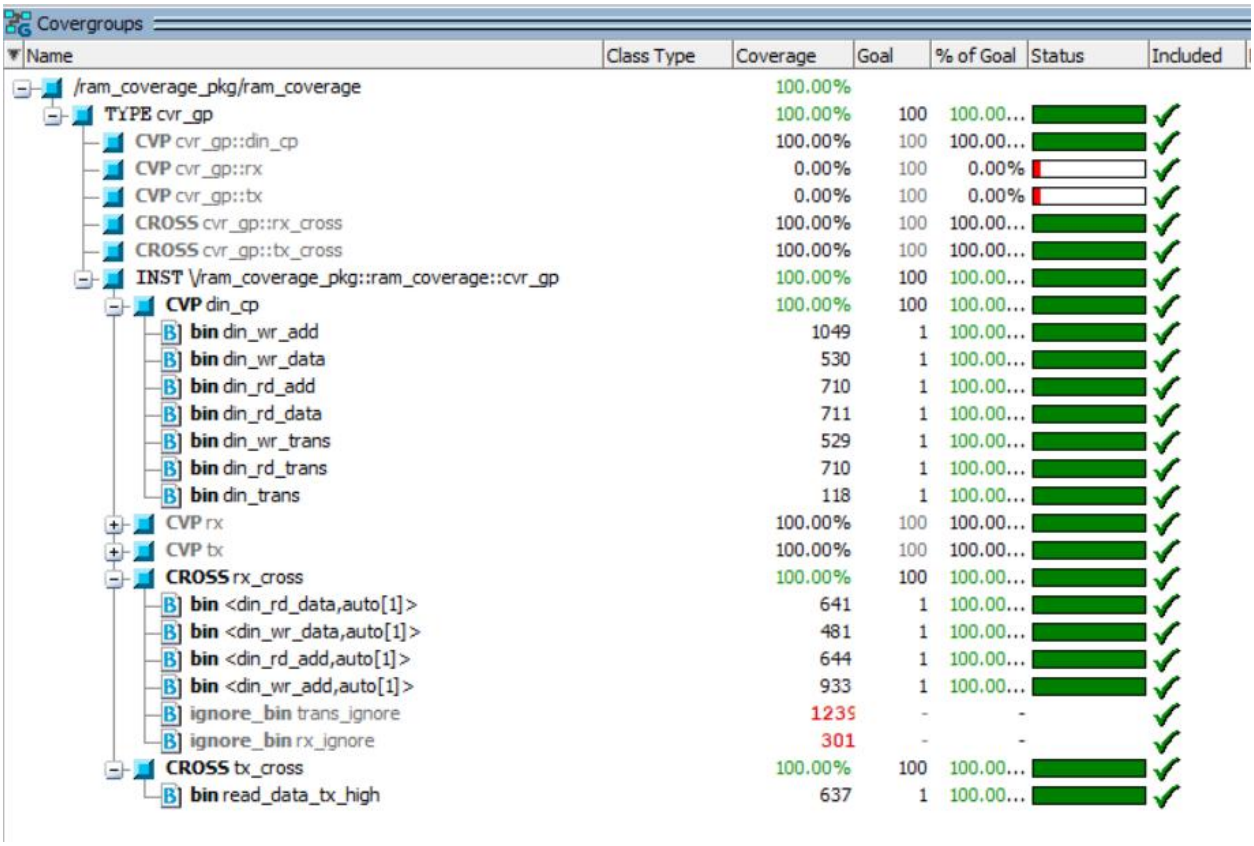
Toggle Coverage

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

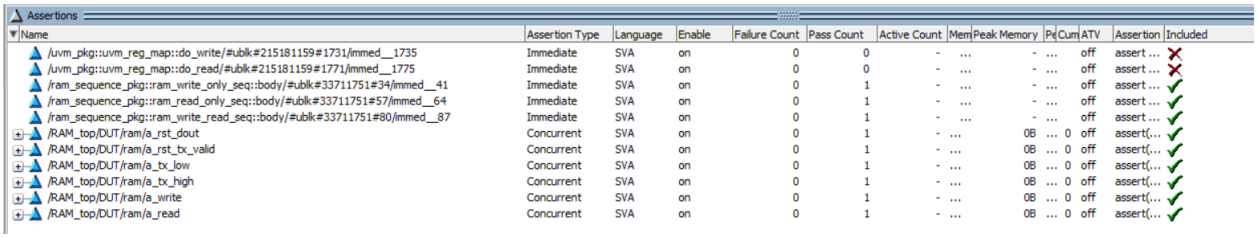
Exclusions:

1. Default branch in branch coverage, as all cases are covered explicitly.
2. Default branch in statement coverage, as all cases are covered explicitly.

FUNCTIONAL COVERAGE REPORT

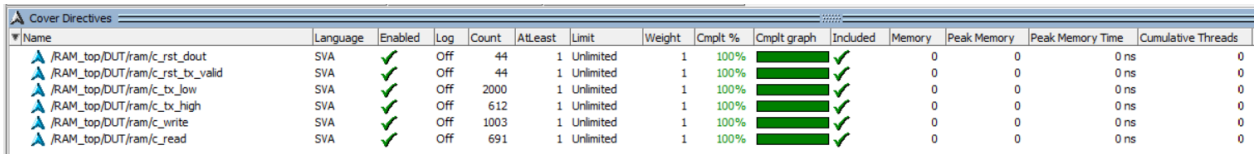


SEQUENTIAL DOMAIN COVERAGE REPORT



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Mem	Peak Memory	Pe	Cum	ATV	Assertion	Included
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	...	-	...		off	assert ...	✗
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	...	-	...		off	assert ...	✗
/ram_sequence_pkg::ram_write_only_seq::body/#ublk#33711751#34/immed__41	Immediate	SVA	on	0	1	-	...	-	...		off	assert ...	✓
/ram_sequence_pkg::ram_read_only_seq::body/#ublk#33711751#57/immed__64	Immediate	SVA	on	0	1	-	...	-	...		off	assert ...	✓
/ram_sequence_pkg::ram_write_read_seq::body/#ublk#33711751#80/immed__87	Immediate	SVA	on	0	1	-	...	-	...		off	assert ...	✓
/RAM_top/DUT/ram/a_rst_dout	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓
/RAM_top/DUT/ram/a_rst_tx_valid	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓
/RAM_top/DUT/ram/a_tx_low	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓
/RAM_top/DUT/ram/a_tx_high	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓
/RAM_top/DUT/ram/a_write	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓
/RAM_top/DUT/ram/a_read	Concurrent	SVA	on	0	1	-	...	0B	...	0	off	assert(...	✓



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/RAM_top/DUT/ram/c_rst_dout	SVA	✓	Off	44	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/RAM_top/DUT/ram/c_rst_tx_valid	SVA	✓	Off	44	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/RAM_top/DUT/ram/c_tx_low	SVA	✓	Off	2000	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/RAM_top/DUT/ram/c_tx_high	SVA	✓	Off	612	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/RAM_top/DUT/ram/c_write	SVA	✓	Off	1003	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/RAM_top/DUT/ram/c_read	SVA	✓	Off	691	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0

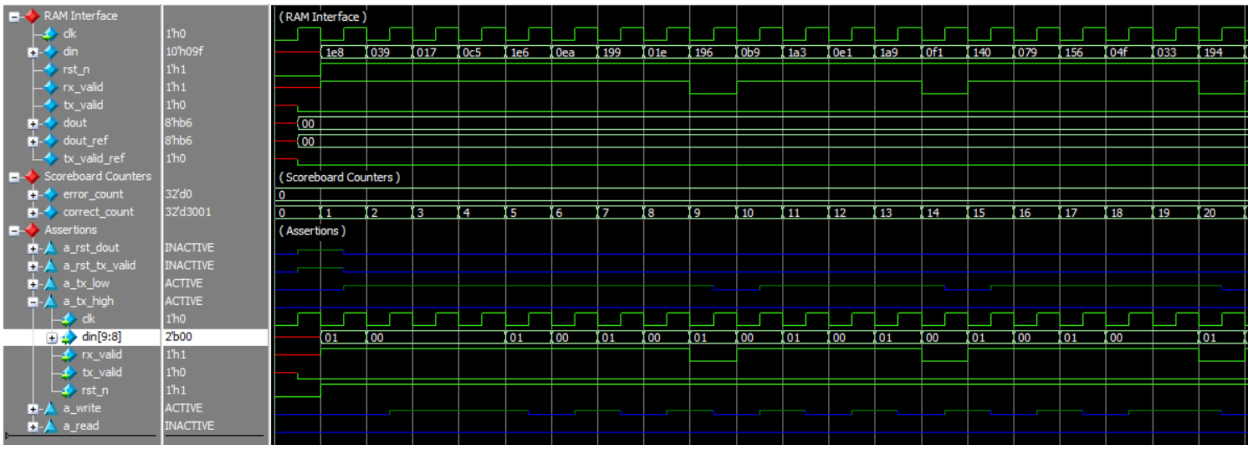
```
=== Instance: /RAM_top/DUT/ram
=== Design Unit: work.RAM_sva
================================================================================

Assertion Coverage:
    Assertions                       6         6         0   100.00%
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/RAM_top/DUT/ram/a_rst_dout
                     RAM_sva.sv(12)                     0          1
/RAM_top/DUT/ram/a_rst_tx_valid
                     RAM_sva.sv(14)                     0          1
/RAM_top/DUT/ram/a_tx_low
                     RAM_sva.sv(20)                     0          1
/RAM_top/DUT/ram/a_tx_high
                     RAM_sva.sv(26)                     0          1
/RAM_top/DUT/ram/a_write
                     RAM_sva.sv(32)                     0          1
/RAM_top/DUT/ram/a_read
                     RAM_sva.sv(37)                     0          1

Directive Coverage:
    Directives                       6         6         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                   Design Design   Lang File(Line)      Hits Status
                                       Unit   UnitType
--------------------------------------------------------------------------------------------
/RAM_top/DUT/ram/c_rst_dout            RAM_sva Verilog  SVA  RAM_sva.sv(13)    44 Covered
/RAM_top/DUT/ram/c_rst_tx_valid        RAM_sva Verilog  SVA  RAM_sva.sv(15)    44 Covered
/RAM_top/DUT/ram/c_tx_low              RAM_sva Verilog  SVA  RAM_sva.sv(23)  2000 Covered
/RAM_top/DUT/ram/c_tx_high             RAM_sva Verilog  SVA  RAM_sva.sv(28)   612 Covered
/RAM_top/DUT/ram/c_write               RAM_sva Verilog  SVA  RAM_sva.sv(34)  1003 Covered
/RAM_top/DUT/ram/c_read                RAM_sva Verilog  SVA  RAM_sva.sv(39)   691 Covered
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         44        44         0   100.00%
```

BUG REPORT

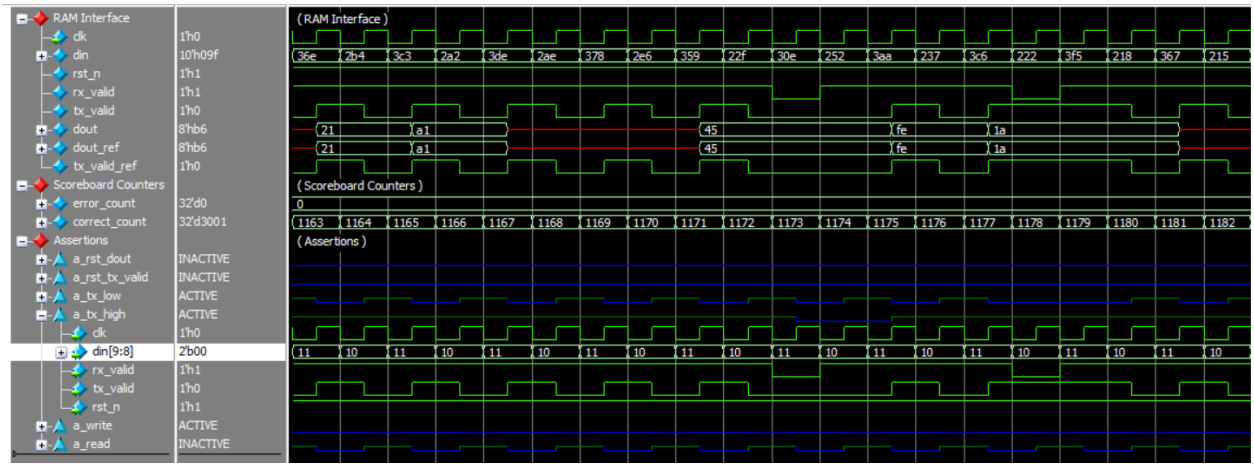
Expected Behavior	Actual Behavior	Edit
In `READD_DATA` case, data is read from the previous read address sent.	Data is read from the previous write address sent.	`2'b11 : dout <= MEM[Rd_Addr];`
`tx_valid` expression is evaluated only if reset is deasserted and `rx_valid` is asserted.	`tx_valid` is updated each cycle regardless `rst_n` and `rx_valid` values.	Moved the `tx_valid` expression inside the `else if(tx_valid)` branch to update only when `rst_n && rx_valid`.

WAVEFORM

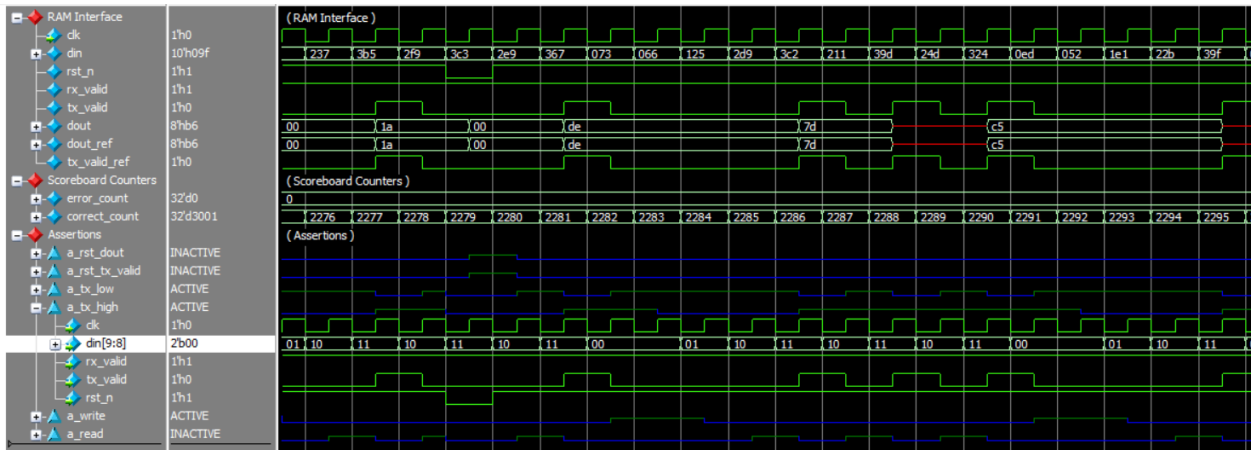
RESET AND WRITE ONLY SEQUENCES



READ ONLY SEQUENCE



WRITE AND READ SEQUENCE



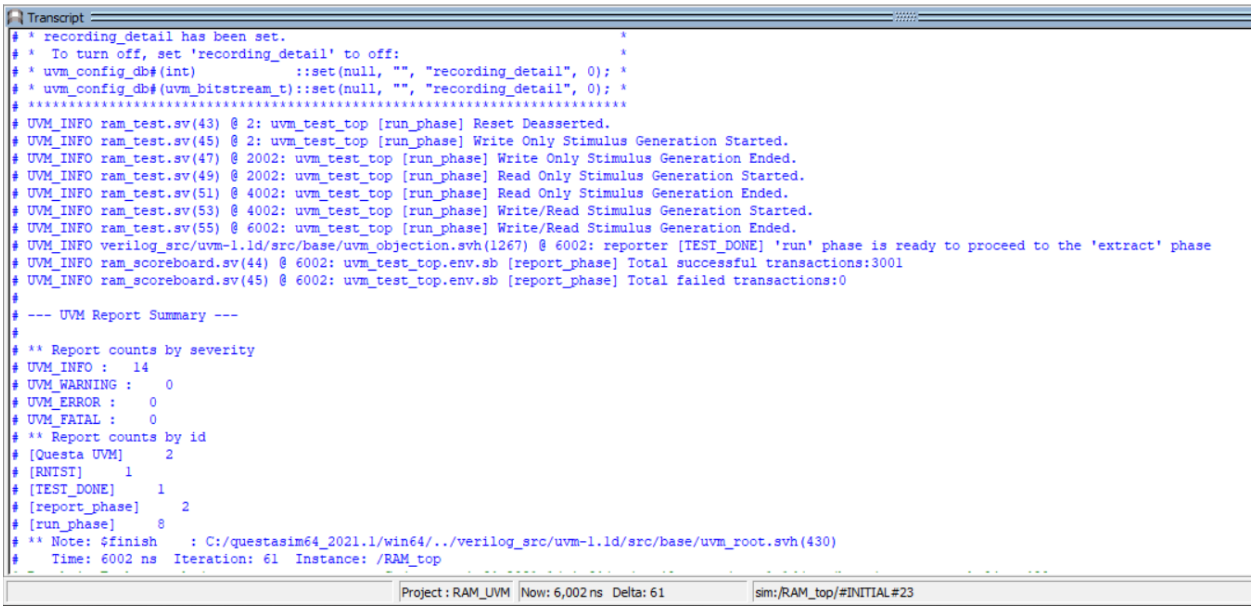
ASSERTIONS

Feature	Assertion

Whenever reset is asserted, (`tx_valid` and `dout`) are low.	`a_rst_dout: assert property(@(posedge clk) ~rst_n |=> dout == 0);`
	
	`a_rst_tx_valid: assert property(@(posedge clk) ~rst_n |=> tx_valid == 0);`

During address or data input phases (`write_add_seq`, `write_data_seq`, `read_add_seq`), the `tx_valid` signal must remain deasserted.	`a_tx_low: assert property(@(posedge clk) disable iff(~rst_n) din[9:8] inside {WRITE_ADDRESS, WRITE_DATA, READ_ADDRESS} && rx_valid |=> ~tx_valid || $fell(tx_valid));`
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.	`a_tx_high: assert property(@(posedge clk) disable iff(~rst_n) (din[9:8] == READ_DATA) && rx_valid |=> tx_valid ##1 $fell(tx_valid) [->1]);`
Every `Write Address` operation must be eventually followed by a `Write Data` operation.	`a_write: assert property(@(posedge clk) disable iff(~rst_n) din[9:8] == WRITE_ADDRESS |=> (din[9:8] == WRITE_DATA) [->1]);`
Every `Read Address` operation must be eventually followed by a `Read Data` operation.	`a_read: assert property(@(posedge clk) disable iff(~rst_n) din[9:8] == READ_ADDRESS |=> (din[9:8] == READ_DATA) [->1]);`

TRANSCRIPT


Transcript
* recording_detail has been set. *
* To turn off, set 'recording_detail' to off: *
* uvm_config_db#(int) ::set(null, "", "recording_detail", 0); *
* uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
UVM_INFO ram_test.sv(43) @ 2: uvm_test_top [run_phase] Reset Deasserted.
UVM_INFO ram_test.sv(45) @ 2: uvm_test_top [run_phase] Write Only Stimulus Generation Started.
UVM_INFO ram_test.sv(47) @ 2002: uvm_test_top [run_phase] Write Only Stimulus Generation Ended.
UVM_INFO ram_test.sv(49) @ 2002: uvm_test_top [run_phase] Read Only Stimulus Generation Started.
UVM_INFO ram_test.sv(51) @ 4002: uvm_test_top [run_phase] Read Only Stimulus Generation Ended.
UVM_INFO ram_test.sv(53) @ 4002: uvm_test_top [run_phase] Write/Read Stimulus Generation Started.
UVM_INFO ram_test.sv(55) @ 6002: uvm_test_top [run_phase] Write/Read Stimulus Generation Ended.
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO ram_scoreboard.sv(44) @ 6002: uvm_test_top.env.sb [report_phase] Total successful transactions:3001
UVM_INFO ram_scoreboard.sv(45) @ 6002: uvm_test_top.env.sb [report_phase] Total failed transactions:0
--- UVM Report Summary ---
** Report counts by severity
UVM_INFO : 14
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
[Questa UVM] 2
[RNTST] 1
[TEST_DONE] 1
[report_phase] 2
[run_phase] 8
** Note: $finish : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
Time: 6002 ns Iteration: 61 Instance: /RAM_top
Project : RAM_UVM
Now: 6,002 ns Delta: 61
sim:/RAM_top/#INITIAL#23

Part Three – SPI Wrapper

Code Snippets

Design

```
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
```

Golden Model

```
module SPI_wrapper_ref(
    input  MOSI,
    input  SS_n,
    input  clk,
    input  rst_n,
    output MISO
);
    wire [9:0] rx_data;
    wire [7:0] tx_data;
    wire rx_valid, tx_valid;
    // SPI_slave module
    spi_slave_ref SPI_Slave(
        .MOSI(MOSI),
        .SS_n(SS_n),
        .clk(clk),
        .rst_n(rst_n),
```

```
        .tx_data(tx_data),
        .tx_valid(tx_valid),
        .MISO(MISO),
        .rx_data(rx_data),
        .rx_valid(rx_valid)
    );
    // MEMORY module
    RAM_ref RAM_ref(
        .din(rx_data),
        .clk(clk),
        .rst_n(rst_n),
        .rx_valid(rx_valid),
        .dout(tx_data),
        .tx_valid(tx_valid)
    );
endmodule
```

INTERFACE

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

SVA

```
module wrapper_sva(
    input  MOSI,
    input  SS_n,
    input  clk,
    input  rst_n,
    input  MISO
);

property property_1;
    @(posedge clk) (~rst_n |-> ##2 (~MISO));
endproperty

// description: checking the that the MISO is stable always except in the case of
READ_DATA == 111
property property_2;
```

```
    @(posedge clk) disable iff(~rst_n)
    ( SS_n ##1 ~SS_n ##1 ~MOSI[*3]  |=> $stable(MISO)[->8]);
endproperty

property property_3;
    @(posedge clk) disable iff(~rst_n)
    ( SS_n ##1 ~SS_n ##1 ~MOSI[*2] ##1 MOSI  |=> $stable(MISO)[->8]);
endproperty

property property_4;
    @(posedge clk) disable iff(~rst_n)
    ( SS_n ##1 ~SS_n ##1 MOSI[*2] ##1 ~MOSI |=> $stable(MISO)[->8]);
endproperty

rst_prop                :assert property(property_1);
MISO_WRITE_ADD_prop     :assert property(property_2);
MISO_WRITE_DATA_prop    :assert property(property_3);
MISO_READ_DATA_prop     :assert property(property_4);

cov_rst_prop            :cover property (property_1);
cov_MISO_WRITE_ADD_prop :cover property (property_2);
cov_MISO_WRITE_DATA_prop :cover property (property_3);
cov_MISO_READ_DATA_prop :cover property (property_4);
endmodule
```

TOP MODULE

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

module SPI_wrapper_top();
  // Clock generation
  bit clk;
  initial begin
    forever #1 clk = ~clk;
  end

  // Instantiation the interfaces
  spi_slave_if SLAVEif (clk);
  RAM_if ram_if (clk);
  SPI_wrapper_if wrapper_if(clk);

  // connect the common wires with the slave
```

```
  //// connecting inputs of the slave
  assign SLAVEif.SS_n = wrapper_if.SS_n;

  //// connecting outputs of the slave to the wrapper
  assign SLAVEif.MOSI = wrapper_if.MOSI;

  // connect the commen wires between the slave and the ram
  //// connecting inputs of the slave
  assign ram_if.din = SLAVEif.rx_data;
  assign ram_if.rx_valid = SLAVEif.rx_valid;

  //// connecting outputs of the ram to the slave
  assign SLAVEif.tx_data  = ram_if.dout;
  assign SLAVEif.tx_valid = ram_if.tx_valid;

  // connecting the rst
  assign SLAVEif.rst_n = wrapper_if.rst_n;
  assign ram_if.rst_n  = wrapper_if.rst_n;

  // Instantiation of the DUTS
  SLAVE SPI_slave (
    SLAVEif.MOSI, SLAVEif.MISO, SLAVEif.SS_n, SLAVEif.clk, SLAVEif.rst_n,
    SLAVEif.rx_data, SLAVEif.rx_valid, SLAVEif.tx_data, SLAVEif.tx_valid
  );


  spi_slave_ref GM (SLAVEif.MOSI, SLAVEif.SS_n, SLAVEif.clk, SLAVEif.rst_n,
SLAVEif.tx_data,
  SLAVEif.tx_valid, SLAVEif.MISO_exp, SLAVEif.rx_data_exp, SLAVEif.rx_valid_exp);

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

  RAM_ref ram_ref(.clk(ram_if.clk),.rst_n(ram_if.rst_n),.din(ram_if.din),
        .rx_valid(ram_if.rx_valid),.tx_valid(ram_if.tx_valid_ref),
        .dout(ram_if.dout_ref));

  WRAPPER SPI_wrapper (
    .MOSI(wrapper_if.MOSI),.MISO(wrapper_if.MISO),
    .SS_n(wrapper_if.SS_n),.clk(wrapper_if.clk),.rst_n(wrapper_if.rst_n)
  );
```

```
  SPI_wrapper_ref REF
  (
    .MOSI(wrapper_if.MOSI),
    .SS_n(wrapper_if.SS_n),
    .clk(wrapper_if.clk),
    .rst_n(wrapper_if.rst_n),
    .MISO(wrapper_if.MISO_ref)
  );

  // binding the interfaces with the duts
  bind RAM RAM_sva RAM_sva_inst(
    .clk(ram_if.clk),.rst_n(ram_if.rst_n),.din(ram_if.din),
    .rx_valid(ram_if.rx_valid),.tx_valid(ram_if.tx_valid),
    .dout(ram_if.dout)
  );

  bind SLAVE spi_slave_sva SLAVE_sva_inst(
    .MOSI(SLAVEif.MOSI), .clk(SLAVEif.clk), .rst_n(SLAVEif.rst_n),
.SS_n(SLAVEif.SS_n),
    .tx_valid(SLAVEif.tx_valid), .tx_data(SLAVEif.tx_data),
.rx_data(SLAVEif.rx_data),
    .rx_valid(SLAVEif.rx_valid), .MISO(SLAVEif.MISO)
  );

  bind WRAPPER wrapper_sva WRAPPER_sva_inst(
    .MOSI(wrapper_if.MOSI),
    .SS_n(wrapper_if.SS_n),
    .clk(wrapper_if.clk),
    .rst_n(wrapper_if.rst_n),
    .MISO(wrapper_if.MISO_ref)
  );

  initial begin
    // set the virtual interfaces in the configuration database
    uvm_config_db #(virtual spi_slave_if)::set(null, "uvm_test_top", "SLAVE_IF",
SLAVEif);
    uvm_config_db #(virtual RAM_if)      ::set(null, "uvm_test_top", "RAM_IF",
ram_if);
    uvm_config_db #(virtual SPI_wrapper_if)  ::set(null, "uvm_test_top",
"WRAPPER_IF", wrapper_if);

    // run test using run_test task
    run_test("SPI_wrapper_test");
  end
endmodule
```

TEST

```
package SPI_wrapper_test_pkg;
import SPI_wrapper_env_pkg::*;
import SPI_wrapper_config_pkg::*;

import spi_slave_config_obj_pkg::*;
import spi_slave_environment_pkg::*;

import ram_enviroment_pkg::*;
import ram_config_obj_pkg::*;

import SPI_wrapper_rst_seq_pkg::*;
import SPI_wrapper_read_only_seq_pkg::*;
import SPI_wrapper_write_only_seq_pkg::*;
import SPI_wrapper_read_write_seq_pkg::*;

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


class SPI_wrapper_test extends uvm_test;
  `uvm_component_utils(SPI_wrapper_test)

  SPI_wrapper_env wrapper_env;
  SPI_wrapper_config SPI_wrapper_cfg_;

  ram_enviroment ram_env;
  ram_config_obj cfg_ram;

  spi_slave_config_obj cfg_obj_test;
  spi_slave_environment slave_env;

  SPI_wrapper_rst_seq reset_seq;
  SPI_wrapper_read_only_seq read_seq;
  SPI_wrapper_write_only_seq write_seq;
  SPI_wrapper_read_write_seq read_write_seq;

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

  // Build the enviornment in the build phase
  function void build_phase(uvm_phase phase);
```

```
    super.build_phase(phase);

    wrapper_env = SPI_wrapper_env::type_id::create("wrapper_env",this);
    SPI_wrapper_cfg_ =
SPI_wrapper_config::type_id::create("SPI_wrapper_cfg_",this);

    ram_env = ram_enviroment::type_id::create("ram_env",this);
    cfg_ram = ram_config_obj::type_id::create("cfg_ram");

    slave_env  =spi_slave_environment::type_id::create("slave_env",this);
    cfg_obj_test=spi_slave_config_obj::type_id::create("cfg_obj_test");

    reset_seq             =
SPI_wrapper_rst_seq::type_id::create("reset_seq",this);
    read_seq        =
SPI_wrapper_read_only_seq::type_id::create("read_seq",this);
    write_seq      =
SPI_wrapper_write_only_seq::type_id::create("write_seq",this);
    read_write_seq =
SPI_wrapper_read_write_seq::type_id::create("read_write_seq",this);

    if(!uvm_config_db #(virtual
SPI_wrapper_if)::get(this,"","WRAPPER_IF",SPI_wrapper_cfg_.SPI_wrapper_vif))
      `uvm_fatal("build_phase","unable to get the interface");
    uvm_config_db
#(SPI_wrapper_config)::set(this,"wrapper_env.*","WRAPPER_CFG",SPI_wrapper_cfg_);
    SPI_wrapper_cfg_.is_active = UVM_ACTIVE;

    if(!uvm_config_db#(virtual RAM_if)::get(this, "", "RAM_IF",
cfg_ram.ram_cfg_vif))
      `uvm_fatal("build_phase","Unable to get the ram virtual interface - test");
    uvm_config_db#(ram_config_obj)::set(this, "ram_env.*", "RAM_CFG", cfg_ram);
    cfg_ram.is_active = UVM_PASSIVE;

    if(!uvm_config_db#(virtual
spi_slave_if)::get(this,"","SLAVE_IF",cfg_obj_test.config_vif))
      `uvm_fatal("build_phase","Test-Unable to get the virtual interface");
    uvm_config_db
#(spi_slave_config_obj)::set(this,"slave_env.*","SLAVE_CFG",cfg_obj_test);
    cfg_obj_test.is_active = UVM_PASSIVE;
  endfunction



  task run_phase(uvm_phase phase);
```

```
    super.run_phase(phase);
    phase.raise_objection(this);

    `uvm_info("run_phase", "reset_seq_on", UVM_LOW);
    reset_seq.start(wrapper_env.agt.sqr);
    `uvm_info("run_phase", "reset_seq_off", UVM_LOW);

    `uvm_info("run_phase", "write_seq_on", UVM_LOW);
    write_seq.start(wrapper_env.agt.sqr);
    `uvm_info("run_phase", "write_seq_off", UVM_LOW);

    `uvm_info("run_phase", "read_seq_on", UVM_LOW);
    read_seq.start(wrapper_env.agt.sqr);
    `uvm_info("run_phase", "read_seq_off", UVM_LOW);


    `uvm_info("run_phase", "read_write_seq_on", UVM_LOW);
    read_write_seq.start(wrapper_env.agt.sqr);
    `uvm_info("run_phase", "read_write_seq_off", UVM_LOW);

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

ENVIRONMENT

```
package SPI_wrapper_env_pkg;
import uvm_pkg::*;
import SPI_wrapper_driver_pkg::*;
import SPI_wrapper_agent_pkg::*;
import SPI_wrapper_score_pkg::*;
`include "uvm_macros.svh"

class SPI_wrapper_env extends uvm_env;
  `uvm_component_utils(SPI_wrapper_env)

  SPI_wrapper_agent agt;
  SPI_wrapper_score sb;

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

```
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agt = SPI_wrapper_agent::type_id::create("agt", this);
    sb  = SPI_wrapper_score::type_id::create("sb", this);
  endfunction

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

AGENT

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

    class SPI_wrapper_agent extends uvm_agent;
        `uvm_component_utils(SPI_wrapper_agent)
        SPI_wrapper_sqr sqr;
        SPI_wrapper_driver drv;
        SPI_wrapper_monitor mon;
        SPI_wrapper_config SPI_wrapper_cfg;
        uvm_analysis_port #(SPI_wrapper_seq_item) agent_aport;

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

        function void  build_phase(uvm_phase phase);
            super.build_phase(phase);
            uvm_config_db #(SPI_wrapper_config) :: get(this, "", "WRAPPER_CFG",
SPI_wrapper_cfg);

            if(SPI_wrapper_cfg.is_active == UVM_ACTIVE) begin
```

```
                drv = SPI_wrapper_driver::type_id::create("drv", this);
                sqr = SPI_wrapper_sqr::type_id::create("sqr", this);
            end
            mon = SPI_wrapper_monitor::type_id::create("mon", this);
            agent_aport = new("agent_port", this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(SPI_wrapper_cfg.is_active == UVM_ACTIVE) begin
                drv.wrapper_vif = SPI_wrapper_cfg.SPI_wrapper_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end

            mon.SPI_wrapper_vif = SPI_wrapper_cfg.SPI_wrapper_vif;
            mon.mon_aport.connect(agent_aport);
        endfunction
    endclass
endpackage
```

DRIVER

```
package SPI_wrapper_driver_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import SPI_wrapper_config_pkg::*;
import SPI_wrapper_seq_item_pkg::*;
    class SPI_wrapper_driver extends uvm_driver #(SPI_wrapper_seq_item);
        `uvm_component_utils(SPI_wrapper_driver)
        virtual SPI_wrapper_if wrapper_vif;
        SPI_wrapper_seq_item stim_seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item =
SPI_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.SS_n     = stim_seq_item.SS_n;
```

```
                wrapper_vif.rst_n = stim_seq_item.rst_n;
                @(negedge wrapper_vif.clk);
                seq_item_port.item_done();
            end
        endtask;
    endclass
endpackage
```

MONITOR

```
package SPI_wrapper_monitor_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SPI_wrapper_config_pkg::*;
    import SPI_wrapper_seq_item_pkg::*;
    //import SPI_wrapper_shared_pkg::*;
    class SPI_wrapper_monitor extends uvm_monitor;
        `uvm_component_utils(SPI_wrapper_monitor)
        virtual SPI_wrapper_if SPI_wrapper_vif;
        SPI_wrapper_seq_item rsp_seq_item;
        uvm_analysis_port #(SPI_wrapper_seq_item) mon_aport;

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

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

        endfunction

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

```
                mon_aport.write(rsp_seq_item);
            end
        endtask
    endclass //className extends superClassendpackage
endpackage
```

SCOREBOARD

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

    class SPI_wrapper_score extends uvm_scoreboard;
        `uvm_component_utils(SPI_wrapper_score)
        uvm_analysis_export   #(SPI_wrapper_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(SPI_wrapper_seq_item) sb_fifo;
        SPI_wrapper_seq_item sb_seq_item;

        int error_count   = 0;
        int correct_count = 0;

        function new(string name = "SPI_wrapper_score", 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(sb_seq_item);
```

```
                if( sb_seq_item.MISO != sb_seq_item.MISO_ref) begin
                    `uvm_error("run_phase", $sformatf("Comparison failed,
MISO_ref = %b | MISO = %b", sb_seq_item.MISO_ref, sb_seq_item.MISO));
                    error_count ++;
                end
                else begin
                    correct_count++;
                end
            end
        endtask

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

SEQUENCE ITEM

```
package SPI_wrapper_seq_item_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    typedef enum int {
        WRITE_ONLY = 0,
        READ_ONLY = 1,
        WRITE_READ = 2
    } spi_mode_e;


    class SPI_wrapper_seq_item extends uvm_sequence_item;
        `uvm_object_utils(SPI_wrapper_seq_item)

        bit  MOSI;
        rand logic  SS_n;
        rand bit  rst_n;
        logic MISO_ref;

        bit [2:0] prev_op;
        bit transaction_ended = 0;

        logic  MISO;
```

```
        int    counter=0;
        int    max_cnt;
        int mosi_index=10;
        rand bit [10:0] mosi_arr;
        spi_mode_e mode;

        constraint rst_n_const
        {
            rst_n dist {1:/98 ,0:/2};
        }

        constraint SS_n_const
        {
            if(transaction_ended)
                SS_n == 1;
            else
                SS_n == 0;
        }

        // WRAPPER_1
        constraint wr_add { mosi_arr[10:8]==3'b000; }
        constraint wr_add_or_data { mosi_arr[10:8] inside {3'b000,3'b001 }; }

        // WRAPPER_2
        constraint rd_add { mosi_arr[10:8]==3'b110; }
        constraint rd_data { mosi_arr[10:8]==3'b111; }

        // WRAPPER_3
        constraint wr_data_add {mosi_arr[10:8] dist {3'b110:/60, 3'b000:/40}; }
        constraint rd_data_add {mosi_arr[10:8] dist {3'b110:/40, 3'b000:/60}; }

    static function SPI_wrapper_seq_item create_item(string name=
"SPI_wrapper_seq_item", spi_mode_e mode);
        SPI_wrapper_seq_item item;
        item = SPI_wrapper_seq_item::type_id::create(name);
        item.mode = mode;

        if (mode == WRITE_ONLY) begin
            item.rd_data.constraint_mode(0);
            item.rd_add.constraint_mode(0);
            item.wr_add_or_data.constraint_mode(0);
            item.wr_add.constraint_mode(1);
            item.wr_data_add.constraint_mode(0);
            item.rd_data_add.constraint_mode(0);
```

```
        end else if (mode == READ_ONLY) begin
            item.wr_add_or_data.constraint_mode(0);
            item.wr_add.constraint_mode(0);
            item.rd_data.constraint_mode(0);
            item.rd_add.constraint_mode(1);
            item.wr_data_add.constraint_mode(0);
            item.rd_data_add.constraint_mode(0);

        end else if (mode==WRITE_READ) begin
            item.rd_data.constraint_mode(0);
            item.rd_add.constraint_mode(0);
            item.wr_add_or_data.constraint_mode(0);
            item.wr_add.constraint_mode(1);
            item.wr_data_add.constraint_mode(0);
            item.rd_data_add.constraint_mode(0);

        end

        return item;
    endfunction

        function void post_randomize();
            if (~rst_n) begin
                transaction_ended=1;
                counter=0;
                mosi_index=10;
                if(mode==WRITE_ONLY)begin
                    mosi_arr[10:8]=0;

                    rd_data.constraint_mode(0);
                    rd_add.constraint_mode(0);

                    wr_add_or_data.constraint_mode(0);
                    wr_add.constraint_mode(1);
                    wr_data_add.constraint_mode(0);
                    rd_data_add.constraint_mode(0);

                end else if (mode==READ_ONLY) begin
                    mosi_arr[10:8]=3'b110;

                    rd_data.constraint_mode(0);
                    rd_add.constraint_mode(1);

                    wr_add_or_data.constraint_mode(0);
                    wr_add.constraint_mode(0);
```

```
                wr_data_add.constraint_mode(0);
                rd_data_add.constraint_mode(0);


            end else if (mode==WRITE_READ) begin
               mosi_arr[10:8]=0;

                rd_data.constraint_mode(0);
                rd_add.constraint_mode(0);

                wr_add_or_data.constraint_mode(0);
                wr_add.constraint_mode(1);

                wr_data_add.constraint_mode(0);
                rd_data_add.constraint_mode(0);

            end

            mosi_arr.rand_mode(1);
        end else begin
            if(mosi_arr[10:8]==3'b111) begin
                max_cnt=23;
            end else begin
                max_cnt=13;
            end

          if(~SS_n)
            counter++;

            if (counter==max_cnt) begin
                transaction_ended=1;
                counter=0;
                mosi_arr.rand_mode(1);
                if(mode==WRITE_ONLY || mode==READ_ONLY)begin
                    if (prev_op==3'b000) begin //write_add
                        wr_add_or_data.constraint_mode(1);
                        wr_add.constraint_mode(0);

                    end else if (prev_op==3'b001)begin //write_data
                        wr_add_or_data.constraint_mode(0);
                        wr_add.constraint_mode(1);
                    end else if (prev_op==3'b110) begin //read_add
                        rd_data.constraint_mode(1);
                        rd_add.constraint_mode(0);
```

```
                    end else if (prev_op==3'b111)begin //read_data
                        rd_data.constraint_mode(0);
                        rd_add.constraint_mode(1);
                    end

                end else if (mode==WRITE_READ) begin
                    if (prev_op==3'b000) begin //write_add
                        rd_data.constraint_mode(0);

                        wr_add_or_data.constraint_mode(1);
//write_add_or_write_data

                        wr_data_add.constraint_mode(0);
                        rd_data_add.constraint_mode(0);
                        wr_add.constraint_mode(0);
                        rd_add.constraint_mode(0);

                    end else if(prev_op==3'b001) begin //write_data
                        rd_data.constraint_mode(0);

                        wr_add_or_data.constraint_mode(0);

                        rd_data_add.constraint_mode(0);
                        wr_add.constraint_mode(0);
                        rd_add.constraint_mode(0);
                        wr_data_add.constraint_mode(1);



                    end else if (prev_op==3'b110) begin

                        wr_add_or_data.constraint_mode(0);

                        wr_data_add.constraint_mode(0);
                        rd_data_add.constraint_mode(0);
                        wr_add.constraint_mode(0);
                        rd_add.constraint_mode(0);
                        rd_data.constraint_mode(1);



                    end else if (prev_op==3'b111) begin
                        rd_data.constraint_mode(0);
                        wr_add.constraint_mode(0);
```

```
                        wr_add_or_data.constraint_mode(0);

                        wr_data_add.constraint_mode(0);
                        rd_add.constraint_mode(0);
                        rd_data_add.constraint_mode(1);
                    end
                end

                mosi_index=10;


            end
            else begin
                transaction_ended=0;
                prev_op=mosi_arr[10:8];
                mosi_arr.rand_mode(0);
            end

            if(mosi_index>=0 && counter>=2 )begin
                   MOSI = mosi_arr[mosi_index];
                   mosi_index--;
            end
        end
    endfunction


    function string convert2string();
        return $sformatf(
            "%s mode = %s | MOSI = 0b%b | SS_n = 0b%b | rst_n = 0b%b | MISO =
0b%b",
            super.convert2string(), mosi_arr[10:8], MOSI, SS_n, rst_n, MISO
        );
    endfunction

    function string convert2string_stimulus();
        return $sformatf("mode = %s | MOSI = 0b%b | SS_n = 0b%b | rst_n =
0b%b | MISO = 0b%b",
            mosi_arr[10:8], MOSI, SS_n, rst_n, MISO
        );
    endfunction
endclass
endpackage
```

Reset Sequence

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

    import SPI_wrapper_seq_item_pkg::*;
    //import SPI_wrapper_shared_pkg::*;
    class SPI_wrapper_rst_seq extends uvm_sequence #(SPI_wrapper_seq_item);
        `uvm_object_utils(SPI_wrapper_rst_seq);
        SPI_wrapper_seq_item seq_item;

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

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

Read Only Sequence

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

    class SPI_wrapper_read_only_seq extends uvm_sequence #(SPI_wrapper_seq_item);
        `uvm_object_utils(SPI_wrapper_read_only_seq);
        SPI_wrapper_seq_item seq_item;

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

        task body;
        seq_item = SPI_wrapper_seq_item::create_item("seq_item", READ_ONLY);
            repeat(1000) begin
```

```
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
endpackage
```

WRITE ONLY SEQUENCE

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

    class SPI_wrapper_write_only_seq extends uvm_sequence
#(SPI_wrapper_seq_item);
        `uvm_object_utils(SPI_wrapper_write_only_seq);
        SPI_wrapper_seq_item seq_item;

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

        task body;
            seq_item = SPI_wrapper_seq_item::create_item("seq_item", WRITE_ONLY);
            repeat(2000) begin
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
endpackage
```

READ WRITE SEQUENCE

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

    class SPI_wrapper_read_write_seq extends uvm_sequence
#(SPI_wrapper_seq_item);
```

```
        `uvm_object_utils(SPI_wrapper_read_write_seq);
        SPI_wrapper_seq_item seq_item;

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

        task body;
            seq_item = SPI_wrapper_seq_item::create_item("seq_item", WRITE_READ);
            repeat(2000) begin
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
endpackage
```

SEQUENCER

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

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

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

CONFIGURATION OBJECT

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

        virtual SPI_wrapper_if SPI_wrapper_vif;
        uvm_active_passive_enum is_active;

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

RAM CONFIGURATION OBJECT

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

    class ram_config_obj extends uvm_object;
        `uvm_object_utils(ram_config_obj);

        virtual RAM_if ram_cfg_vif;
        uvm_active_passive_enum is_active;

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

SLAVE CONFIGURATION OBJECT

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

        virtual spi_slave_if config_vif;
        uvm_active_passive_enum is_active;

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

DO FILE

```
vlib work
vlog -f src_file.list +cover -covercells
vsim -voptargs=+acc work.SPI_wrapper_top -classdebug -uvmcontrol=all -cover
run 0
#add wave -position insertpoint  \
#sim:/uvm_root/uvm_test_top/read_seq.seq_item
add wave -position insertpoint  \
sim:/uvm_root/uvm_test_top/read_write_seq.seq_item
do wave.do
run -all
```

WAVE DO FILE

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider {Wrapper inteface}
add wave -noupdate /SPI_wrapper_top/wrapper_if/clk
add wave -noupdate -color Sienna /SPI_wrapper_top/wrapper_if/rst_n
add wave -noupdate -color Yellow /SPI_wrapper_top/wrapper_if/SS_n
add wave -noupdate /SPI_wrapper_top/wrapper_if/MOSI
add wave -noupdate /SPI_wrapper_top/wrapper_if/MISO
add wave -noupdate /SPI_wrapper_top/wrapper_if/MISO_ref
add wave -noupdate -divider {Slave interface}
add wave -noupdate /SPI_wrapper_top/SLAVEif/clk
add wave -noupdate -color Sienna /SPI_wrapper_top/SLAVEif/rst_n
add wave -noupdate -color Yellow /SPI_wrapper_top/SLAVEif/SS_n
add wave -noupdate /SPI_wrapper_top/SLAVEif/MOSI
```

```
add wave -noupdate /SPI_wrapper_top/SLAVEif/rx_data
add wave -noupdate /SPI_wrapper_top/SLAVEif/rx_data_exp
add wave -noupdate /SPI_wrapper_top/SLAVEif/rx_valid
add wave -noupdate /SPI_wrapper_top/SLAVEif/rx_valid_exp
add wave -noupdate /SPI_wrapper_top/SLAVEif/MISO
add wave -noupdate /SPI_wrapper_top/SLAVEif/MISO_exp
add wave -noupdate /SPI_wrapper_top/SLAVEif/tx_valid
add wave -noupdate /SPI_wrapper_top/SLAVEif/tx_data
add wave -noupdate /SPI_wrapper_top/SPI_slave/cs
add wave -noupdate /SPI_wrapper_top/GM/cs
add wave -noupdate -divider {RAM interface}
add wave -noupdate /SPI_wrapper_top/ram_if/clk
add wave -noupdate -color Sienna /SPI_wrapper_top/ram_if/rst_n
add wave -noupdate /SPI_wrapper_top/ram_if/din
add wave -noupdate /SPI_wrapper_top/ram_if/rx_valid
add wave -noupdate /SPI_wrapper_top/ram_if/tx_valid
add wave -noupdate /SPI_wrapper_top/ram_if/tx_valid_ref
add wave -noupdate /SPI_wrapper_top/ram_if/dout
add wave -noupdate /SPI_wrapper_top/ram_if/dout_ref
add wave -noupdate
/SPI_wrapper_read_write_seq_pkg::SPI_wrapper_read_write_seq::body/#ublk#215952055
#16/immed__18
add wave -noupdate
/SPI_wrapper_write_only_seq_pkg::SPI_wrapper_write_only_seq::body/#ublk#101763687
#16/immed__18
add wave -noupdate
/SPI_wrapper_read_only_seq_pkg::SPI_wrapper_read_only_seq::body/#ublk#154982439#1
6/immed__18
add wave -noupdate /SPI_wrapper_top/SPI_slave/assert__p_idle_chk_trans
add wave -noupdate /SPI_wrapper_top/SPI_slave/assert__chk_to_states_trans
add wave -noupdate /SPI_wrapper_top/SPI_slave/assert__p_to_idle_trans
add wave -noupdate
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_wr_add
add wave -noupdate
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_wr_data
add wave -noupdate
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_rd_add
add wave -noupdate
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_rd_data
add wave -noupdate /SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_reset
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_rst_dout
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_rst_tx_valid
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_tx_low
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_tx_high
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_write
```

```
add wave -noupdate /SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_read
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_rst_dout
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_rst_tx_valid
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_tx_low
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_tx_high
add wave -noupdate /SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_write
add wave -noupdate /SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_read
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__p_idle_chk_trans
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__chk_to_states_trans
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__p_to_idle_trans
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_wr_
add
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_wr_
data
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_rd_
add
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_rd_
data
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_reset
add wave -noupdate /SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/rst_prop
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_WRITE_ADD_prop
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_WRITE_DATA_prop
add wave -noupdate
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_READ_DATA_prop
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {34 ns} 0} {{Cursor 3} {6940 ns} 0}
quietly wave cursor active 1
configure wave -namecolwidth 281
configure wave -valuecolwidth 56
configure wave -justifyvalue left
configure wave -signalnamewidth 1
```

```
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {0 ns} {94 ns}
```

SOURCE FILES LIST

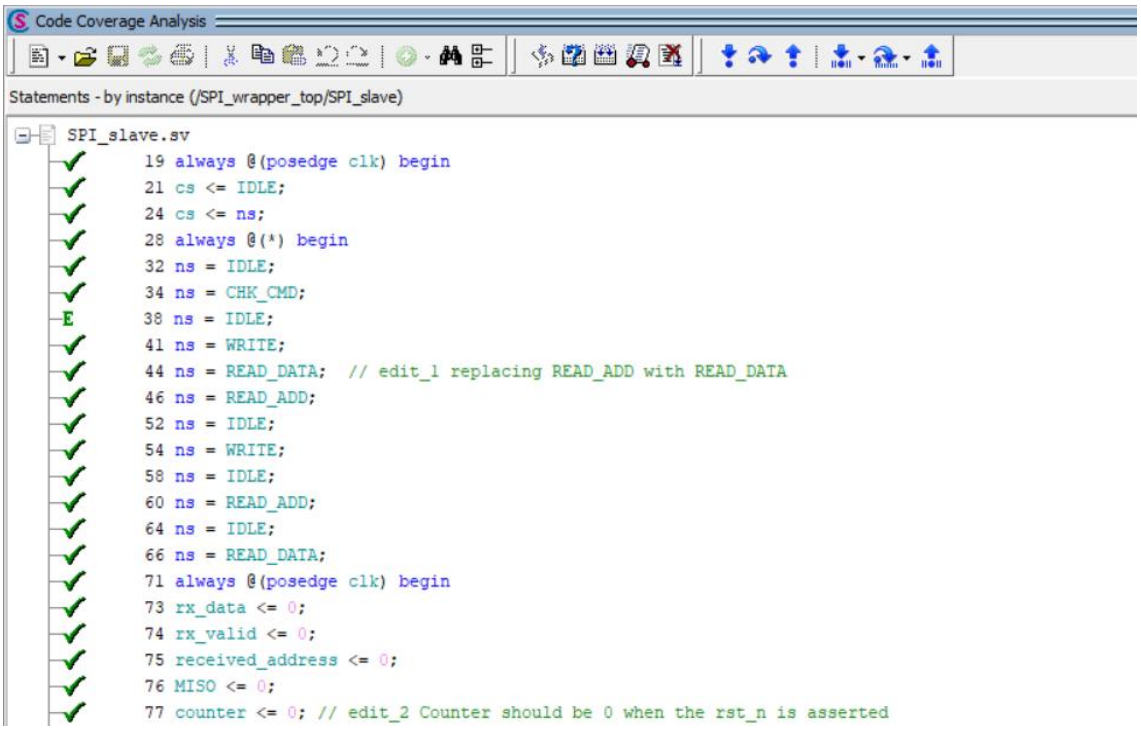
```
SPI_slave.sv
SPI_slave_ref.v
SPI_slave_agent.sv
SPI_slave_cfg.sv
SPI_slave_cov.sv
SPI_slave_driver.sv
SPI_slave_if.sv
SPI_slave_sva.sv
SPI_slave_main_seq.sv
SPI_slave_monitor.sv
SPI_slave_rst_seq.sv
SPI_slave_scoreboard.sv
SPI_slave_seq_item.sv
SPI_slave_sequencer.sv
SPI_slave_env.sv

RAM.v
RAM_ref.v
ram_pkg.sv
RAM_if.sv
RAM_sva.sv
ram_config_obj.sv
ram_sequence_item.sv
ram_sequence.sv
ram_sequencer.sv
ram_driver.sv
ram_monitor.sv
ram_agent.sv
ram_scoreboard.sv
ram_coverage.sv
ram_enviroment.sv
```

```
SPI_wrapper.v
SPI_wrapper_ref.v
SPI_wrapper_interface.sv
SPI_wrapper_sva.sv
SPI_wrapper_config.sv
SPI_wrapper_driver.sv
SPI_wrapper_monitor.sv
SPI_wrapper_seq_item.sv
SPI_wrapper_sqr.sv
SPI_wrapper_agent.sv
SPI_wrapper_score.sv
SPI_wrapper_env.sv
SPI_wrapper_rst_seq.sv
SPI_wrapper_read_only_seq.sv
SPI_wrapper_write_only_seq.sv
SPI_wrapper_read_write_seq.sv
SPI_wrapper_test.sv
SPI_wrapper_top.sv
```

COVERAGE REPORTS

CODE COVERAGE REPORT

- SLAVE



```
 82 rx_valid <= 0;
 85 counter <= 10;
 88 counter <= counter - 1;
 90 rx_data[counter-1] <= MOSI;
 92 rx_valid <= (counter ==0 )? 1 : 0; //EDIT
 95 counter <= counter - 1;
 97 rx_data[counter-1] <= MOSI;
100 received_address <= 1;
102 rx_valid <= (counter ==0 )? 1 : 0; //EDIT
106 rx_valid <= 0;
108 MISO <= tx_data[counter-1];
109 counter <= counter - 1;
112 received_address <= 0;
121 rx_data[counter-1] <= MOSI;
122 counter <= counter - 1;
125 rx_valid <= 1;
126 counter <= 8;
```

Branches - by instance (/SPI_wrapper_top/SPI_slave)

SPI_slave.sv

```
 20 if (~rst_n) begin
 23 else begin
 29 case (cs)
 30 IDLE : begin
 31 if (SS_n)
 33 else
 36 CHK_CMD : begin
 37 if (SS_n)
 39 else begin
 40 if (~MOSI)
 42 else begin
 43 if (received_address)
 45 else
 50 WRITE : begin
 51 if (SS_n)
 53 else
 56 READ_ADD : begin
 57 if (SS_n)
 59 else
 62 READ_DATA : begin
 63 if (SS_n)
 65 else
 72 if (~rst_n) begin
 79 else begin
```


80 case (cs)
81 IDLE : begin
84 CHK_CMD : begin
87 WRITE : begin
89 if (counter > 0) begin
92 rx_valid <= (counter ==0)? 1 : 0; //EDIT
94 READ_ADD : begin
96 if (counter > 0) begin
99 else begin
102 rx_valid <= (counter ==0)? 1 : 0; //EDIT
104 READ_DATA : begin
105 if (tx_valid) begin
107 if (counter > 0) begin
111 else begin
115 else begin
120 if (counter > 0 && ~rx_valid) begin
124 else begin


FSMs - by instance (/SPI_wrapper_top/SPI_slave)
cs
IDLE (0)
CHK_CMD (2)
READ_ADD (3)
READ_DATA (4)
WRITE (1)


Toggles - by instance (/SPI_wrapper_top/SPI_slave)
sim:/SPI_wrapper_top/SPI_slave
clk
counter
cs
MISO
MOSI
ns
received_address
rst_n
rx_data
rx_valid
SS_n
tx_data
tx_valid

- RAM

Statements - by instance (/SPI_wrapper_top/SPI_RAM)

RAM.v

```
13 always @(posedge clk) begin
15 dout <= 0;
16 tx_valid <= 0;
17 Rd_Addr <= 0;
18 Wr_Addr <= 0;
23 2'b00 : Wr_Addr <= din[7:0];
24 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 2'b10 : Rd_Addr <= din[7:0];
26 2'b11 : dout <= MEM[Rd_Addr];   // edit_1 Rd_Addr not Wr_Addr
27 default : dout <= 0;
29 tx_valid <= (din[9] && din[8])? 1'b1 : 1'b0; // edit_2 -> only if reset signal is deasserted
```

Branches - by instance (/SPI_wrapper_top/SPI_RAM)

RAM.v

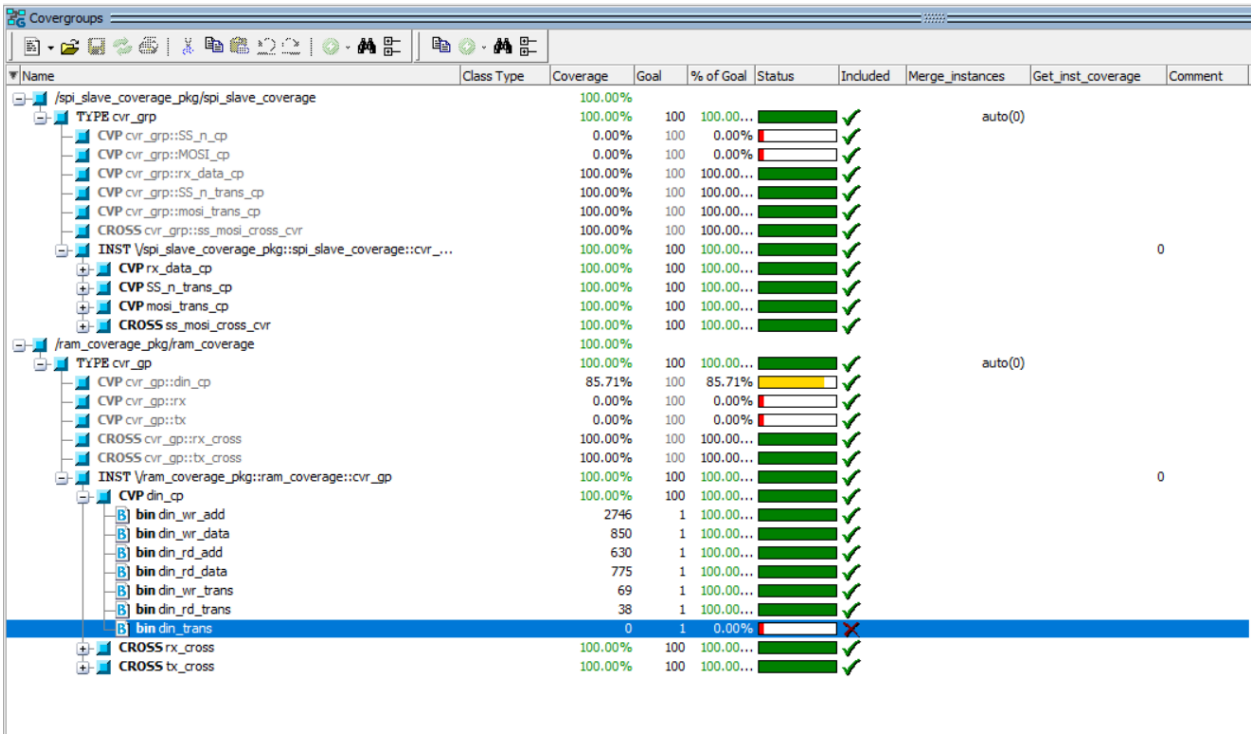
```
14 if (~rst_n) begin
21 if (rx_valid) begin
23 2'b00 : Wr_Addr <= din[7:0];
24 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 2'b10 : Rd_Addr <= din[7:0];
26 2'b11 : dout <= MEM[Rd_Addr];   // edit_1 Rd_Addr not Wr_Addr
27 default : dout <= 0;
```

Toggles - by instance (/SPI_wrapper_top/SPI_RAM)

sim:/SPI_wrapper_top/SPI_RAM

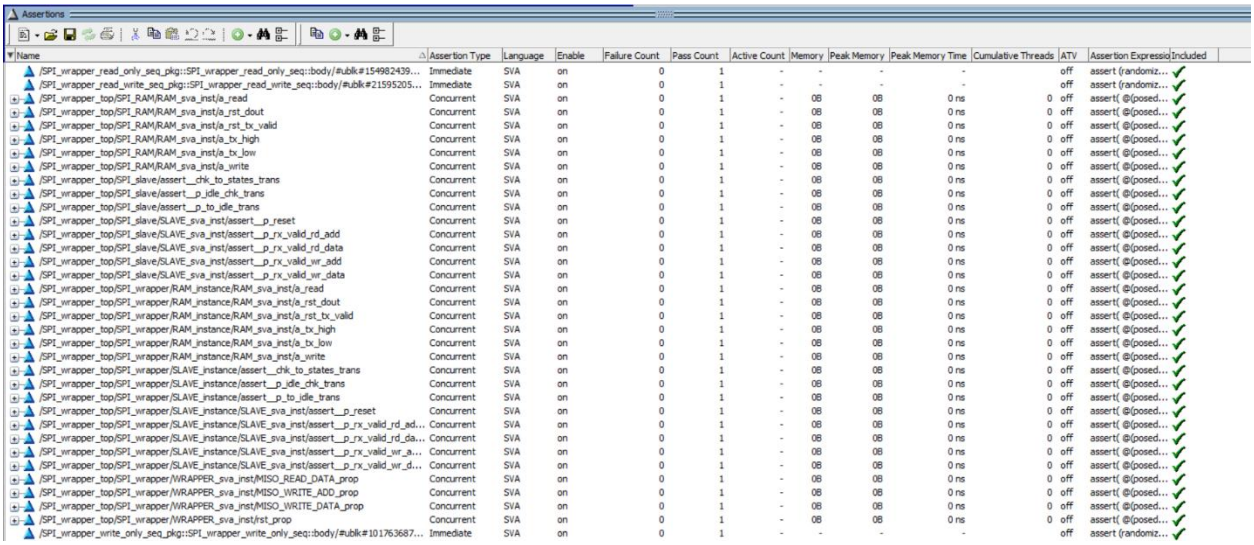
- clk
- din
- dout
- Rd_Addr
- rst_n
- rx_valid
- tx_valid
- Wr_Addr

FUNCTIONAL COVERAGE REPORT



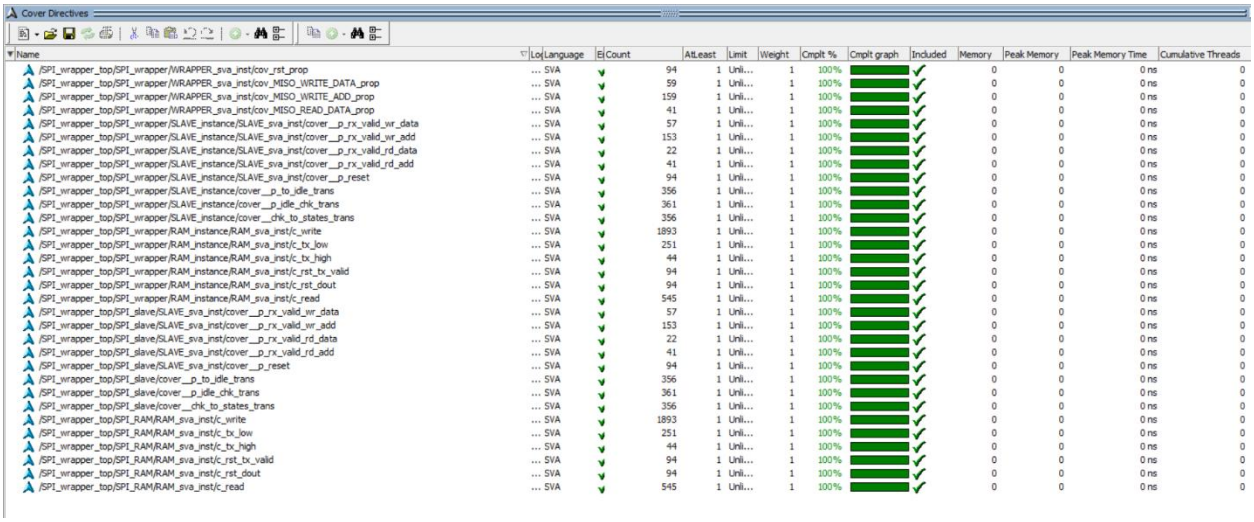
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/spi_slave_coverage_pkg/spi_slave_coverage		100.00%							
TYPE cvr_grp		100.00%	100	100.00...			auto(0)		
CVP cvr_grp::SS_n_cp		0.00%	100	0.00%					
CVP cvr_grp::MOSI_cp		0.00%	100	0.00%					
CVP cvr_grp::rx_data_cp		100.00%	100	100.00...					
CVP cvr_grp::SS_n_trans_cp		100.00%	100	100.00...					
CVP cvr_grp::mosi_trans_cp		100.00%	100	100.00...					
CROSS cvr_grp::ss_mosi_cross_cvr		100.00%	100	100.00...					
INST \/spi_slave_coverage_pkg::spi_slave_coverage::cvr_...		100.00%	100	100.00...					0
CVP rx_data_cp		100.00%	100	100.00...					
CVP SS_n_trans_cp		100.00%	100	100.00...					
CVP mosi_trans_cp		100.00%	100	100.00...					
CROSS ss_mosi_cross_cvr		100.00%	100	100.00...					
/ram_coverage_pkg/ram_coverage		100.00%							
TYPE cvr_gp		100.00%	100	100.00...			auto(0)		
CVP cvr_gp::din_cp		85.71%	100	85.71%					
CVP cvr_gp::rx		0.00%	100	0.00%					
CVP cvr_gp::tx		0.00%	100	0.00%					
CROSS cvr_gp::rx_cross		100.00%	100	100.00...					
CROSS cvr_gp::tx_cross		100.00%	100	100.00...					
INST \/ram_coverage_pkg::ram_coverage::cvr_gp		100.00%	100	100.00...					0
CVP din_cp		100.00%	100	100.00...					
bin din_wr_add		2746	1	100.00...					
bin din_wr_data		850	1	100.00...					
bin din_rd_add		630	1	100.00...					
bin din_rd_data		775	1	100.00...					
bin din_wr_trans		69	1	100.00...					
bin din_rd_trans		38	1	100.00...					
bin din_trans		0	1	0.00%					
CROSS rx_cross		100.00%	100	100.00...					
CROSS tx_cross		100.00%	100	100.00...					

SEQUENTIAL DOMAIN COVERAGE REPORT



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expressio	Included
/SPI_wrapper_read_only_seq_pkg::SPI_wrapper_read_only_seq::body/#ublk#154982439...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomiz...	✓
/SPI_wrapper_read_write_seq_pkg::SPI_wrapper_read_write_seq::body/#ublk#21595205...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomiz...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_read	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_rst_dout	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_rst_tx_valid	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_tx_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_tx_low	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/a_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/assert__chk_to_states_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/assert__p_idle_chk_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/assert__p_to_idle_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_reset	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_rd_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_rd_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_wr_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/assert__p_rx_valid_wr_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_read	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_rst_dout	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_rst_tx_valid	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_tx_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_tx_low	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/a_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__chk_to_states_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__p_idle_chk_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/assert__p_to_idle_trans	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_reset	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_rd_ad...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_rd_da...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_wr_a...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/assert__p_rx_valid_wr_d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_READ_DATA_prop	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_WRITE_ADD_prop	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/MISO_WRITE_DATA_prop	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/rst_prop	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posed...	✓
/SPI_wrapper_write_only_seq_pkg::SPI_wrapper_write_only_seq::body/#ublk#101763687...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomiz...	✓

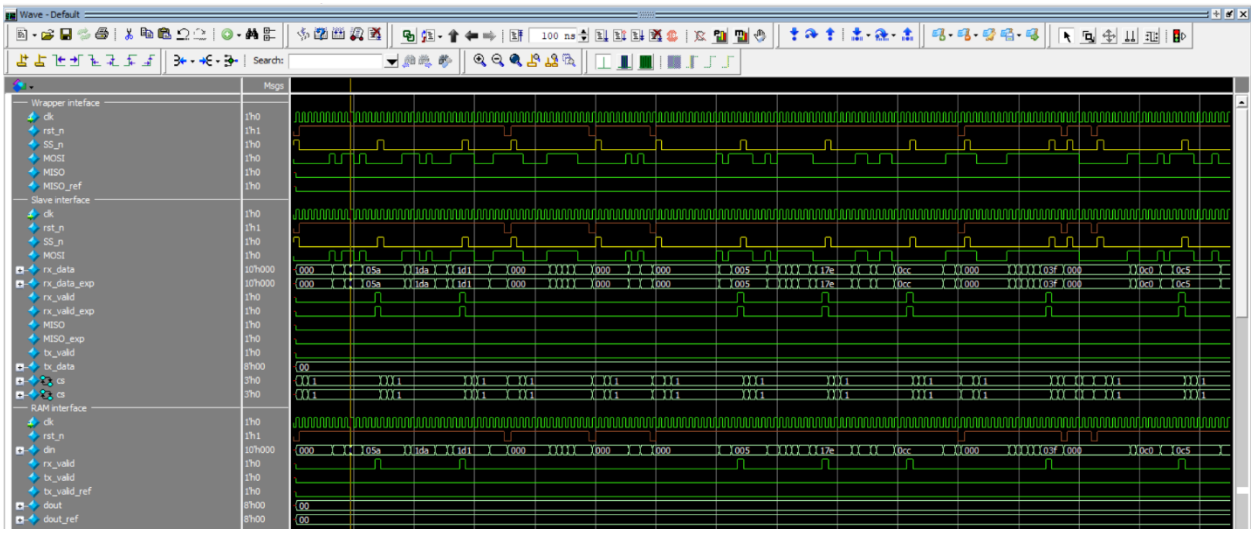


Cover Directives

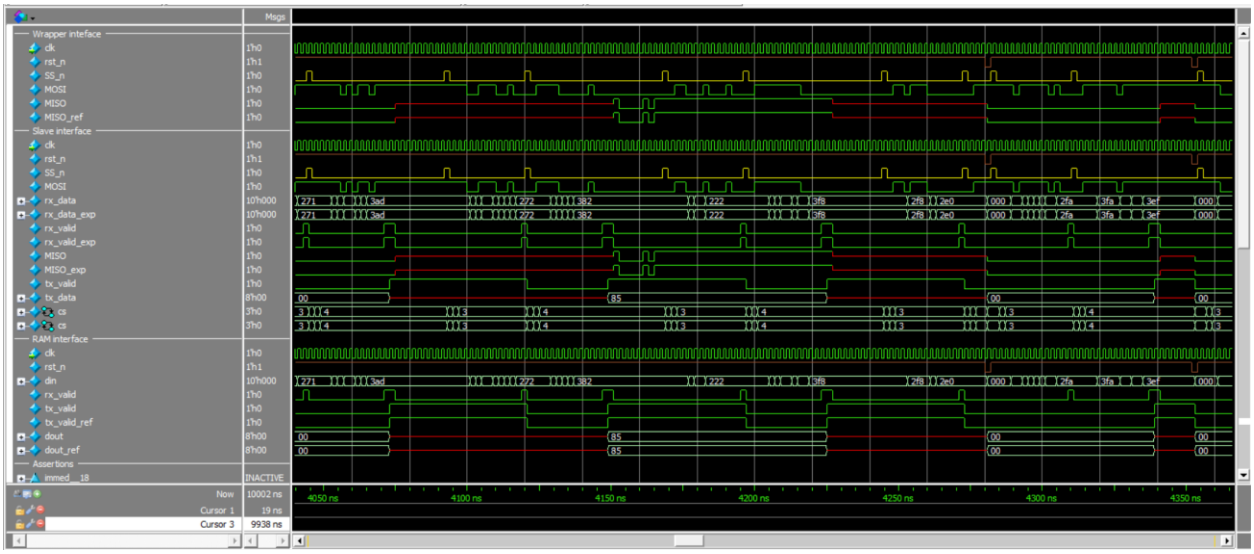
Name	Lo	Language	E	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/cov_rst_prop	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/cov_MISO_WRITE_DATA_prop	...	SVA	✓	59	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/cov_MISO_WRITE_ADD_prop	...	SVA	✓	159	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/WRAPPER_sva_inst/cov_MISO_READ_DATA_prop	...	SVA	✓	41	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/cover__p_rx_valid_wr_data	...	SVA	✓	57	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/cover__p_rx_valid_wr_add	...	SVA	✓	153	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/cover__p_rx_valid_rd_data	...	SVA	✓	22	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/cover__p_rx_valid_rd_add	...	SVA	✓	41	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/SLAVE_sva_inst/cover__p_reset	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/cover__p_to_idle_trans	...	SVA	✓	356	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/cover__p_idle_chk_trans	...	SVA	✓	361	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/SLAVE_instance/cover__chk_to_states_trans	...	SVA	✓	356	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_write	...	SVA	✓	1893	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_tx_low	...	SVA	✓	251	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_tx_high	...	SVA	✓	44	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_rst_tx_valid	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_rst_dout	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_wrapper/RAM_instance/RAM_sva_inst/c_read	...	SVA	✓	545	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/cover__p_rx_valid_wr_data	...	SVA	✓	57	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/cover__p_rx_valid_wr_add	...	SVA	✓	153	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/cover__p_rx_valid_rd_data	...	SVA	✓	22	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/cover__p_rx_valid_rd_add	...	SVA	✓	41	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/SLAVE_sva_inst/cover__p_reset	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/cover__p_to_idle_trans	...	SVA	✓	356	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/cover__p_idle_chk_trans	...	SVA	✓	361	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_slave/cover__chk_to_states_trans	...	SVA	✓	356	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_write	...	SVA	✓	1893	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_tx_low	...	SVA	✓	251	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_tx_high	...	SVA	✓	44	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_rst_tx_valid	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_rst_dout	...	SVA	✓	94	1	Uni...	1	100%		✓	0	0	0 ns	0
/SPI_wrapper_top/SPI_RAM/RAM_sva_inst/c_read	...	SVA	✓	545	1	Uni...	1	100%		✓	0	0	0 ns	0

WAVEFORM

- WRITE ONLY SEQ



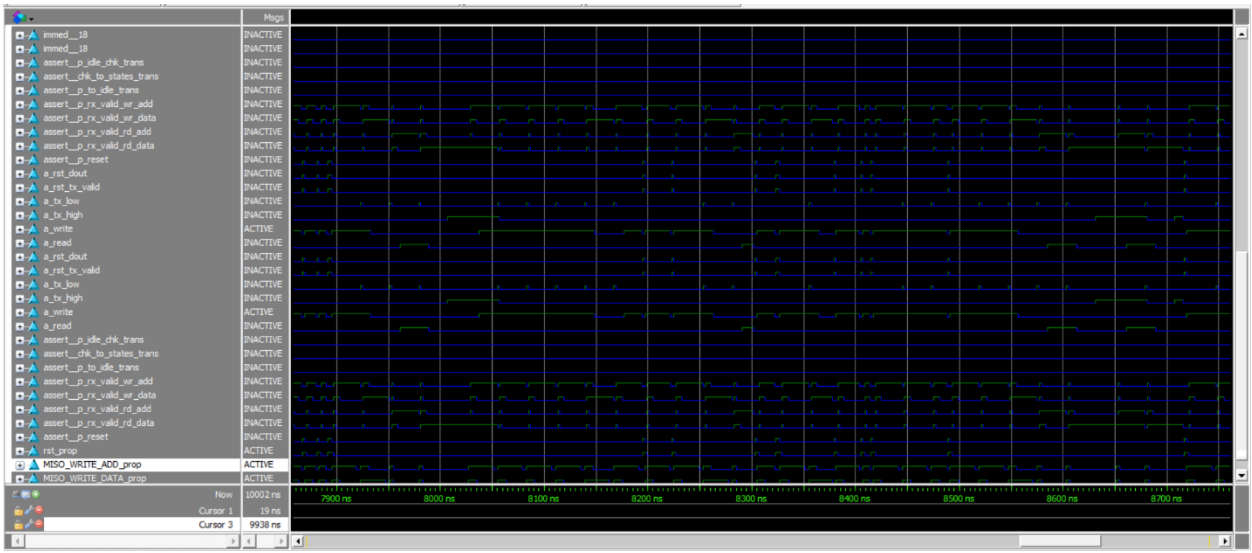
- READ ONLY SEQ



- READ WRITE SEQ

ASSERTIONS

Feature	Assertion
If the rst_n is asserted low, the MOSI will go low the next rising edge	`@(posedge clk) (~rst_n |-> ##2 (~MISO));`
If the rst_n is high and the current state is READ_ADD, the MOSI will be stable.	`@(posedge clk) disable iff(~rst_n)` `( SS_n ##1 ~SS_n ##1 MOSI[*2] ##1 ~MOSI |=> $stable(MISO)[->8]);`
If the rst_n is high and the current state is WRITE_ADD, the MOSI will be stable.	`@(posedge clk) disable iff(~rst_n)` `( SS_n ##1 ~SS_n ##1 ~MOSI[*3] |=> $stable(MISO)[->8]);`
If the rst_n is high and the current state is WRITE_DATA, the MOSI will be stable.	`@(posedge clk) disable iff(~rst_n)` `( SS_n ##1 ~SS_n ##1 ~MOSI[*2] ##1 MOSI |=> $stable(MISO)[->8]);`



TRANSCRIPT

Transcript

```
# UVM_INFO SPI_wrapper_test.sv(83) @ 2: uvm_test_top [run_phase] reset_seq_off
# UVM_INFO SPI_wrapper_test.sv(85) @ 2: uvm_test_top [run_phase] write_seq_on
# UVM_INFO SPI_wrapper_test.sv(87) @ 4002: uvm_test_top [run_phase] write_seq_off
# UVM_INFO SPI_wrapper_test.sv(89) @ 4002: uvm_test_top [run_phase] read_seq_on
# UVM_INFO SPI_wrapper_test.sv(91) @ 6002: uvm_test_top [run_phase] read_seq_off
# UVM_INFO SPI_wrapper_test.sv(94) @ 6002: uvm_test_top [run_phase] read_write_seq_on
# UVM_INFO SPI_wrapper_test.sv(96) @ 10002: uvm_test_top [run_phase] read_write_seq_off
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 10002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ram_scoreboard.sv(44) @ 10002: uvm_test_top.ram_env.sb [report_phase] Total successful transactions:5001
# UVM_INFO ram_scoreboard.sv(45) @ 10002: uvm_test_top.ram_env.sb [report_phase] Total failed transactions:0
# UVM_INFO SPI_slave_scoreboard.sv(48) @ 10002: uvm_test_top.slave_env.sb [report_phase] Total successful transactions:5001
# UVM_INFO SPI_slave_scoreboard.sv(49) @ 10002: uvm_test_top.slave_env.sb [report_phase] Total failed transactions:0
# UVM_INFO SPI_wrapper_score.sv(47) @ 10002: uvm_test_top.wrapper_env.sb [report_phase] Total successful transactions:5001
# UVM_INFO SPI_wrapper_score.sv(48) @ 10002: uvm_test_top.wrapper_env.sb [report_phase] Total failed transactions:0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   18
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     6
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10002 ns  Iteration: 61  Instance: /SPI_wrapper_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 18>
```