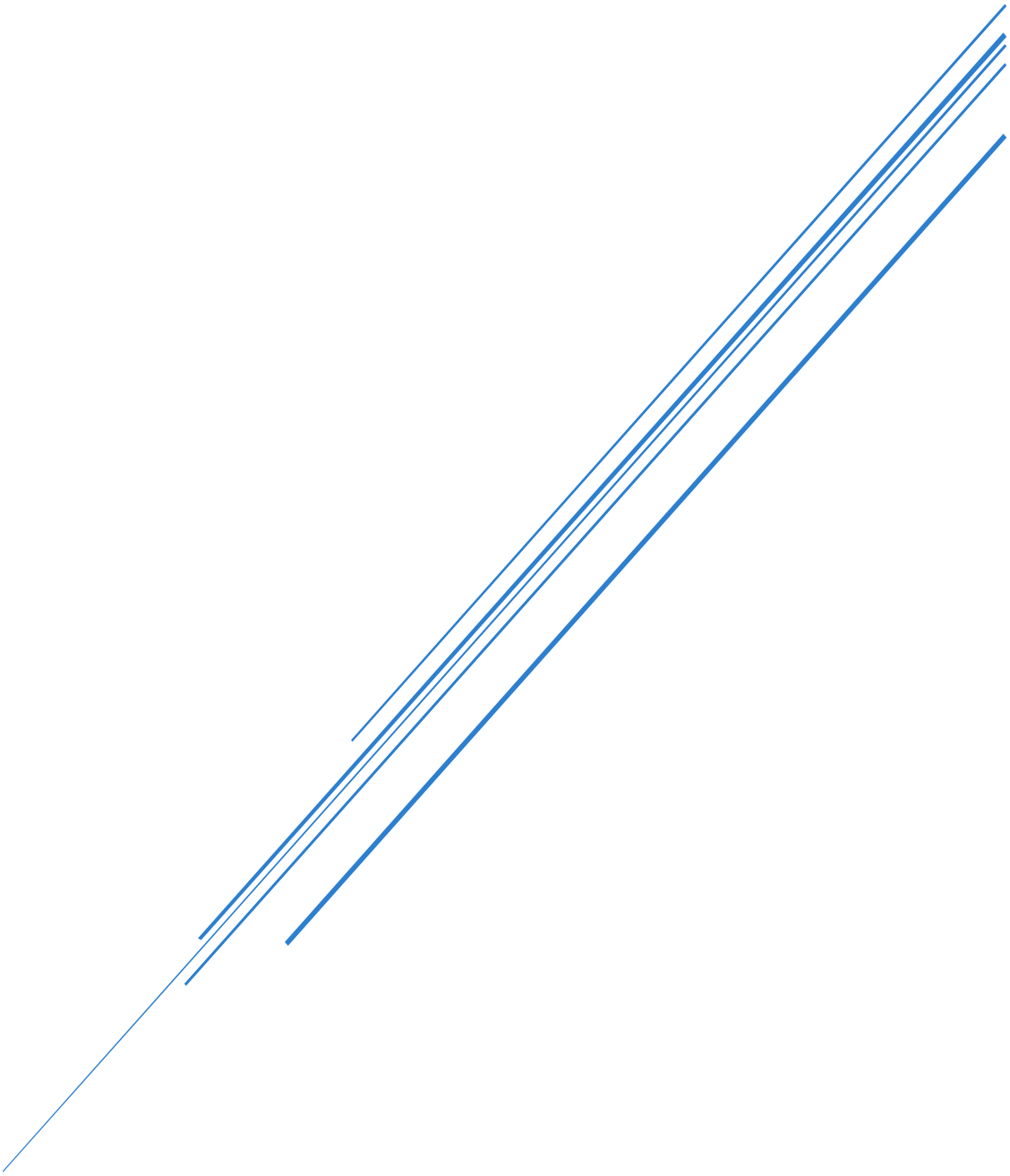
System Verilog project

SPI (Serial Peripheral Interface) slave with single-port RAM

Digital UVM Verification

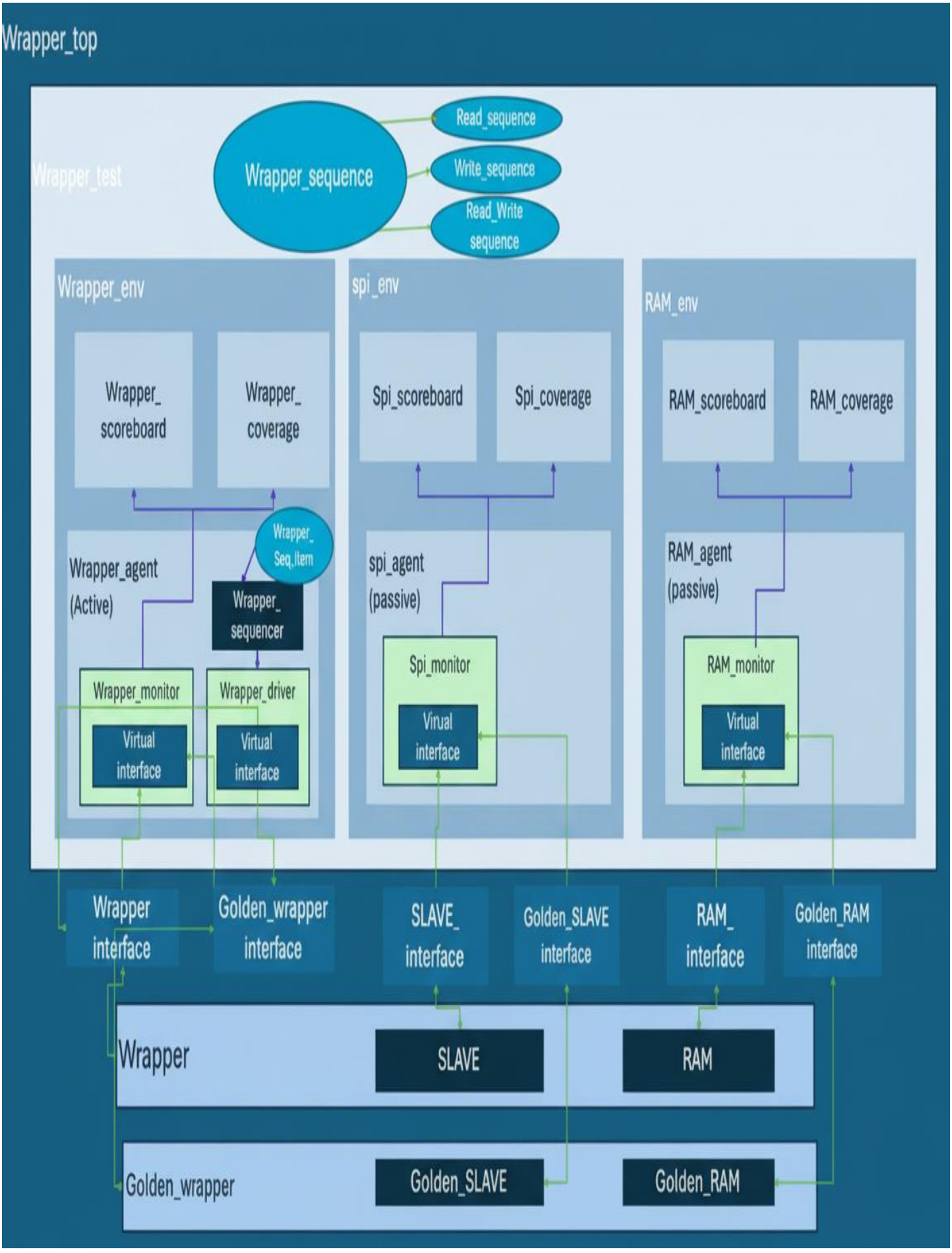


YOUSSEF MOHAMED ABDELAAL BAYOUM	G3
Nabil Ebrahim Abd ElAty Abd ElRazeq	G2
Abdelrahman Mahmoud	G2

Table of Contents

Project Title: SPI (Serial Peripheral Interface) slave with single-port RAM

UVM test structure



Detailed Description of UVM Testbench Operation

The verification environment for the **Wrapper system** is developed using the **Universal Verification Methodology (UVM)** to ensure **modularity, reusability, and coverage-driven verification**.
The complete UVM testbench architecture is illustrated in the **Wrapper_top** block diagram and consists of three primary environments:
Wrapper_env, **spi_env**, and **RAM_env**, each dedicated to monitoring and verifying a specific part of the Design Under Test (DUT).

---

1. Overview

At the top level, the testbench (**Wrapper_test**) instantiates the three environments and coordinates the overall stimulus generation through the **Wrapper_sequence**.
The DUT (**Wrapper**) communicates with two submodules, **SLAVE** and **RAM**, via their respective interfaces.
For functional verification, corresponding **Golden Models** (**Golden_wrapper**, **Golden_SLAVE**, and **Golden_RAM**) are instantiated to serve as reference models for result comparison.

All UVM components are interconnected using **virtual interfaces**, enabling seamless transaction-level communication between the UVM class-based environment and the DUT's signal-level interfaces.

---

2. Wrapper Environment (Active Agent)

The **Wrapper_env** is the main verification environment and serves as the **active component** of the testbench.
It comprises the following elements:

- **Wrapper_agent (Active)** – Generates and drives stimulus to the DUT.
- **Wrapper_driver** – Converts high-level sequence items into signal-level transactions via the **Wrapper_interface**.
- **Wrapper_monitor** – Observes DUT responses, reconstructs transactions, and forwards them to analysis components.
- **Wrapper_scoreboard** – Compares DUT outputs with the **Golden_wrapper** reference model.
- **Wrapper_coverage** – Collects functional coverage to evaluate verification progress.

The **Wrapper_agent** operates in **active mode**, meaning it both **drives stimulus** (through the driver) and **monitors responses**.
The **Wrapper_sequencer** coordinates between the test sequences and the driver, ensuring proper flow of transaction data.

3. SPI and RAM Environments (Passive Agents)

The **spi_env** and **RAM_env** are configured as **passive environments**, meaning they do not generate stimulus but only observe and analyze activity on their respective interfaces.

SPI ENVIRONMENT

- **spi_agent (Passive)**
  - **spi_monitor**
  - **spi_scoreboard**
  - **spi_coverage**

The **spi_monitor** samples the SPI interface signals, converts them into SPI transaction objects, and forwards these objects to the scoreboard and coverage collectors.

RAM ENVIRONMENT

- **RAM_agent (Passive)**
  - **RAM_monitor**
  - **RAM_scoreboard**
  - **RAM_coverage**

The **RAM_monitor** captures all memory interface transactions and forwards them to the **RAM_scoreboard** for checking against the **Golden_RAM** model.

Both passive environments validate the correctness of data exchanged between the Wrapper and its connected modules while collecting coverage metrics to ensure that all protocol features are exercised.

---

4. Sequences and Transaction Flow

Test stimulus is generated by the **Wrapper_sequence**, which controls different operational scenarios:

- **Read_sequence**
- **Write_sequence**
- **Read_Write_sequence**

Each sequence creates **Wrapper_seq_item** objects representing read or write operations, including parameters such as address, data, and operation type.
These sequence items are passed to the **Wrapper_sequencer**, which forwards them to the **Wrapper_driver**.

The **driver** translates these transactions into signal-level activities on the DUT interface.
Meanwhile, the **Wrapper_monitor**, **spi_monitor**, and **RAM_monitor** continuously observe the DUT behavior and capture corresponding responses.

5. Scoreboards and Golden Models

Each environment incorporates a **scoreboard** that performs transaction-level comparisons between the DUT outputs and the expected results generated by the **Golden Models**.

- The **Golden Models** (**Golden_wrapper**, **Golden_SLAVE**, and **Golden_RAM**) act as behavioral reference models that represent the expected functionality of each DUT component.
- When a transaction is driven to the DUT, the same input is also applied to the corresponding golden model.
- The **scoreboard** compares the DUT's actual response with the golden model's predicted response.
- Any mismatch is reported as a **UVM_ERROR**, ensuring accurate functional validation.

This mechanism guarantees that the Wrapper and its submodules operate correctly under all tested conditions.

---

6. Coverage and Assertions

To ensure complete verification:

- **Functional Coverage** components within each environment track:
  - Transaction types
  - Address ranges
  - Data patterns
  - Operation combinations
- **SystemVerilog Assertions (SVA)** are embedded throughout the environment to verify:
  - Protocol compliance
  - Reset behavior
  - Signal stability
  - Timing correctness

Examples include checking that all outputs remain inactive during reset and validating that handshake signals (tx_valid, rx_valid) transition correctly.

The integration of **functional coverage** and **assertions** provides both **quantitative** and **qualitative** confidence in the correctness of the DUT.

---

7. Test Execution Flow

1. **Build Phase:**
   All UVM components (environments, agents, monitors, scoreboards) are constructed and connected. Virtual interfaces are configured.
2. **Connect Phase:**
   Analysis ports are connected between monitors, scoreboards, and coverage components.
3. **Run Phase:**
   - The **Wrapper_sequence** is started on the **Wrapper_sequencer**.
   - **Drivers** translate sequence items into DUT-level signal activity.
   - **Monitors** capture DUT responses and send transactions to scoreboards and coverage modules.
   - **Scoreboards** perform comparisons between DUT outputs and golden model predictions.

4. **Check and Report Phases:**
    - Scoreboards summarize any mismatches detected during simulation.
    - Coverage reports are generated to assess verification completeness.
    - Assertion results are reviewed to validate protocol compliance.

---

8. Summary

The developed **UVM testbench** provides a **modular, scalable, and coverage-driven** verification framework for validating the **Wrapper system**.
By integrating **active and passive agents**, **golden reference models**, **functional coverage**, and **assertions**, the testbench ensures comprehensive verification of:

- **Data correctness**
- **Protocol adherence**
- **Timing integrity**
- **Functional coverage closure**

This structured approach guarantees a high level of verification confidence and maintains compliance with UVM best practices.

SLAVE part

First: Snippets from codes:

Design and bugs found:

- Bug 1 :

```
if (~rst_n) begin
    rx_data <= 0;
    rx_valid <= 0;
    received_address <= 0;
    MISO <= 0;
end
```

```
if (~rst_n) begin
    rx_data <= 0;
    rx_valid <= 0;
    received_address <= 0;
    counter <= 0;
    MISO <= 0;
end
```

• Counter Reset Behavior:
When the reset signal is asserted, the internal counter is synchronously cleared to zero. This guarantees that the counter always starts from a known state after reset, preventing any unintended accumulation or timing inconsistencies from previous operations.

- Bug 2 :

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

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

• Command Validation Handling:
In the `CHK_CMD` state, if the received address flag is not asserted, the design transitions to the `READ_ADDR` state instead of `READ_DATA`. This ensures that data is only read after a valid address has been received, maintaining proper command flow and preventing unintended memory access.

- Bug 3 :

```
READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 8;
                    end
                end
            end
```

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

• Read Data Counter Adjustment:
In the `READ_DATA` state, if `tx_valid` is not asserted and the counter loop completes with the counter value equal to zero, the counter is reassigned to `9` instead of `8`. This modification ensures proper alignment of data transmission timing and prevents premature counter rollover during read operations.

- Complete design without bugs & assertions:

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

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

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

reg [3:0] counter;
reg       received_address;
reg [2:0] cs, ns;
```

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

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

```
always @(posedge clk) begin
    if (~rst_n) begin
        rx_data <= 0;
        rx_valid <= 0;
        received_address <= 0;
        counter <= 0;
        MISO <= 0;
    end
    else begin
        case (cs)
            IDLE : begin
                rx_valid <= 0;
            end
            CHK_CMD : begin
                counter <= 10;
            end
            WRITE : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end
            READ_ADD : begin
                if (counter > 0) begin
                    rx_data[counter-1] <= MOSI;
                    counter <= counter - 1;
                end
                else begin
                    rx_valid <= 1;
                    received_address <= 1;
                end
            end
            READ_DATA : begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (counter > 0) begin
                        MISO <= tx_data[counter-1];
                        counter <= counter - 1;
                    end
                    else begin
                        received_address <= 0;
                    end
                end
                else begin
                    if (counter > 0) begin
                        rx_data[counter-1] <= MOSI;
                        counter <= counter - 1;
                    end
                    else begin
                        rx_valid <= 1;
                        counter <= 9;
                    end
                end
            end
        endcase
    end
end
```

```
`ifdef SIM
    // property check reset
    property check_reset;
        @(posedge clk)
            (!rst_n) |=> (rx_valid == 0 && rx_data == 0 && MISO == 0);
    endproperty

    // Property: IDLE → CHK_CMD transition
    property idle_to_chk_cmd;
        @(posedge clk) disable iff (!rst_n)
        (cs == IDLE && !SS_n) |=> (cs == CHK_CMD);
    endproperty

    // Property: IDLE → IDLE when SS_n is high
    property idle_to_idle;
        @(posedge clk) disable iff (!rst_n)
        (cs == IDLE && SS_n) |=> (cs == IDLE);
    endproperty

    // Property: CHK_CMD → WRITE (when MOSI=0 and SS_n=0)
    property chk_cmd_to_write;
        @(posedge clk) disable iff (!rst_n)
        (cs == CHK_CMD && !SS_n && !MOSI) |=> (cs == WRITE);
    endproperty

    // Property: CHK_CMD → READ_ADD (when MOSI=1, received_address=0, SS_n=0)
    property chk_cmd_to_read_add;
        @(posedge clk) disable iff (!rst_n)
        (cs == CHK_CMD && !SS_n && MOSI && !received_address) |=> (cs == READ_ADD);
    endproperty

    // Property: CHK_CMD → READ_DATA (when MOSI=1, received_address=1, SS_n=0)
    property chk_cmd_to_read_data;
        @(posedge clk) disable iff (!rst_n)
        (cs == CHK_CMD && !SS_n && MOSI && received_address) |=> (cs == READ_DATA);
    endproperty

    // Property: CHK_CMD → IDLE (when SS_n goes high)
    property chk_cmd_to_idle;
        @(posedge clk) disable iff (!rst_n)
        (cs == CHK_CMD && SS_n) |=> (cs == IDLE);
    endproperty

    // Property: WRITE → IDLE (when SS_n goes high)
    property write_to_idle;
        @(posedge clk) disable iff (!rst_n)
        (cs == WRITE && SS_n) |=> (cs == IDLE);
    endproperty

    // Property: WRITE → WRITE (when SS_n remains low)
    property write_to_write;
        @(posedge clk) disable iff (!rst_n)
        (cs == WRITE && !SS_n) |=> (cs == WRITE);
    endproperty

    // Property: READ_ADD → IDLE (when SS_n goes high)
    property read_add_to_idle;
        @(posedge clk) disable iff (!rst_n)
        (cs == READ_ADD && SS_n) |=> (cs == IDLE);
    endproperty

    // Property: READ_ADD → READ_ADD (when SS_n remains low)
    property read_add_to_read_add;
        @(posedge clk) disable iff (!rst_n)
        (cs == READ_ADD && !SS_n) |=> (cs == READ_ADD);
    endproperty

    // Property: READ_DATA → IDLE (when SS_n goes high)
    property read_data_to_idle;
        @(posedge clk) disable iff (!rst_n)
        (cs == READ_DATA && SS_n) |=> (cs == IDLE);
    endproperty

    // Property: READ_DATA → READ_DATA (when SS_n remains low)
    property read_data_to_read_data;
        @(posedge clk) disable iff (!rst_n)
        (cs == READ_DATA && !SS_n) |=> (cs == READ_DATA);
    endproperty

    // Property: Valid states only
    property valid_states_only;
        @(posedge clk) disable iff (!rst_n)
        cs inside {IDLE, WRITE, CHK_CMD, READ_ADD, READ_DATA};
    endproperty
```

```
    //=========================================================================
     // Assertions
     //=========================================================================
     assert_idle_to_chk_cmd: assert property (idle_to_chk_cmd)
         else $error("FSM Error @ %0t: IDLE did not transition to CHK_CMD when SS_n=0", $time);

     assert_idle_to_idle: assert property (idle_to_idle)
         else $error("FSM Error @ %0t: IDLE did not stay in IDLE when SS_n=1", $time);

     assert_chk_cmd_to_write: assert property (chk_cmd_to_write)
         else $error("FSM Error @ %0t: CHK_CMD did not transition to WRITE when MOSI=0", $time);

     assert_chk_cmd_to_read_add: assert property (chk_cmd_to_read_add)
         else $error("FSM Error @ %0t: CHK_CMD did not transition to READ_ADD", $time);

     assert_chk_cmd_to_read_data: assert property (chk_cmd_to_read_data)
         else $error("FSM Error @ %0t: CHK_CMD did not transition to READ_DATA", $time);

     assert_chk_cmd_to_idle: assert property (chk_cmd_to_idle)
         else $error("FSM Error @ %0t: CHK_CMD did not transition to IDLE when SS_n=1", $time);

     assert_write_to_idle: assert property (write_to_idle)
         else $error("FSM Error @ %0t: WRITE did not transition to IDLE when SS_n=1", $time);

     assert_write_to_write: assert property (write_to_write)
         else $error("FSM Error @ %0t: WRITE did not stay in WRITE when SS_n=0", $time);

     assert_read_add_to_idle: assert property (read_add_to_idle)
         else $error("FSM Error @ %0t: READ_ADD did not transition to IDLE when SS_n=1", $time);

     assert_read_add_to_read_add: assert property (read_add_to_read_add)
         else $error("FSM Error @ %0t: READ_ADD did not stay in READ_ADD when SS_n=0", $time);

     assert_read_data_to_idle: assert property (read_data_to_idle)
         else $error("FSM Error @ %0t: READ_DATA did not transition to IDLE when SS_n=1", $time);

     assert_read_data_to_read_data: assert property (read_data_to_read_data)
         else $error("FSM Error @ %0t: READ_DATA did not stay in READ_DATA when SS_n=0", $time);

     assert_valid_states: assert property (valid_states_only)
         else $error("FSM Error @ %0t: Invalid state detected: cs=%b", $time, cs);

     a_check_reset: assert property (check_reset)
         else $error("Assertion FAILED: Reset condition not satisfied — outputs not cleared properly.");

     //=========================================================================
     // Coverage for FSM transitions
     //=========================================================================
     cover_idle_to_chk_cmd: cover property (idle_to_chk_cmd);
     cover_chk_cmd_to_write: cover property (chk_cmd_to_write);
     cover_chk_cmd_to_read_add: cover property (chk_cmd_to_read_add);
     cover_chk_cmd_to_read_data: cover property (chk_cmd_to_read_data);
     cover_write_to_idle: cover property (write_to_idle);
     cover_read_add_to_idle: cover property (read_add_to_idle);
     cover_read_data_to_idle: cover property (read_data_to_idle);
     c_check_reset: cover property (check_reset);

 `endif

 endmodule
```

Interface :

```
interface SLAVE_if (clk);
    input clk ;
    logic rst_n ;
    logic MOSI ;
    logic SS_n ;
    logic tx_valid;
    logic [7:0] tx_data;
    logic [9:0] rx_data;
    logic rx_valid ;
    bit MISO;
endinterface
```

Top module:

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

module SLAVE_top();
  bit clk ,reset ;
  initial begin
    forever
      #1 clk =~clk ;
  end
  SLAVE_if inst_if (clk) ;
  SLAVE inst_DUT (inst_if.MOSI,inst_if.MISO,inst_if.SS_n,inst_if.clk,
                  .rst_n,inst_if.rx_data,inst_if.rx_valid,inst_if.tx_data,
                  inst_if.tx_valid);

  initial begin
    uvm_config_db #(virtual SLAVE_if)::set(null, "uvm_test_top" , "SLAVE_if" ,inst_if) ;
    run_test("SLAVE_test");
  end
endmodule
```

SLAVE test file:

```
package SLAVE_test_pkg;
  import SLAVE_env_pkg::*;
  import SLAVE_config_pkg::*;
  import SLAVE_reset_seq_pkg::*;
  import SLAVE_seq_pkg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class SLAVE_test extends uvm_test;
    `uvm_component_utils(SLAVE_test)
    SLAVE_env env ;
    SLAVE_config SLAVE_cfg ;
    SLAVE_seq seq;
    SLAVE_seq_reset rst_seq;

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

    function void build_phase (uvm_phase phase);
      super.build_phase(phase) ;
      env = SLAVE_env::type_id::create("env",this);
      SLAVE_cfg = SLAVE_config::type_id::create("SLAVE_cfg");
      seq = SLAVE_seq::type_id::create("seq");
      rst_seq = SLAVE_seq_reset::type_id::create("rst_seq");

      if (!uvm_config_db #(virtual SLAVE_if)::get(this, "" , "SLAVE_if" , SLAVE_cfg.SLAVE_vif))
        `uvm_fatal("build_phase" , "TEST - unable to get the virtual intrface");

      uvm_config_db #(SLAVE_config)::set(this , "*" , "CFG" , SLAVE_cfg);

    endfunction

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

      `uvm_info("run_phase", "seq stimulis generated started " , UVM_MEDIUM)
      seq.start(env.agt.sqr);
      `uvm_info("run_phase", "seq stimulis generated ended" , UVM_MEDIUM)

      phase.drop_objection(this);
    endtask: run_phase
  endclass

endpackage
```

SLAVE seq_item file:

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

    class SLAVE_item extends uvm_sequence_item;
        `uvm_object_utils(SLAVE_item)

        // Randomized inputs
        rand logic rst_n;
        rand logic SS_n;
        rand bit [10:0] MOSI_array;

        // Non-randomized inputs (driven from MOSI_array or constraints)
        logic MOSI;
        rand logic tx_valid;
        rand logic [7:0] tx_data;

        // DUT outputs (monitored)
        logic [9:0] rx_data;
        logic rx_valid;
        logic MISO;

        // State tracking variables
        logic [4:0] count;              // Counter for SS_n timing (0-22)
        logic [3:0] bit_index;          // Track which bit of MOSI_array to send (0-10)
        logic prev_SS_n;                // Previous value of SS_n
        bit [10:0] stored_MOSI_array;   // Store the randomized array
        bit [2:0] current_cmd;          // Store current command being executed

        function new(string name = "SLAVE_item");
            super.new(name);
            prev_SS_n = 1'b1;
            count = 0;
            bit_index = 0;
            stored_MOSI_array = 11'b000_0000_0000;
            current_cmd = 3'b000;
        endfunction

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

        function string convert2string_stimulus();
            return $sformatf("rst_n=%b, MOSI=%b, SS_n=%b, tx_valid=%b, tx_data=%h, cmd=%b, count=
, bit_idx=                rst_n, MOSI, SS_n, tx_valid, tx_data, current_cmd, count, bit_index);
",      endfunction

        // Constraint 1: Reset signal deasserted most of the time
        constraint reset_c {
            rst_n dist {1 := 99, 0 := 1};
        }

        // Constraint 2: SS_n timing based on command type
        // For WRITE (000) and READ_ADD (001): SS_n high every 13 cycles
        // For READ_DATA (110, 111): SS_n high every 23 cycles
        constraint SS_n_timing_c {
            if (current_cmd inside {3'b000, 3'b001 ,3'b110}) {
                // Write or Read Address: 13 cycle period (12 low, 1 high)
                SS_n == (count == 12);
            }
            else if (current_cmd inside { 3'b111}) {
                // Read Data: 23 cycle period (22 low, 1 high)
                SS_n == (count == 22);
            }
            else {
                // Default/initial case
                SS_n == (count == 12);
            }
        }
```

```
        // Constraint 3: MOSI_array randomization
        // Only randomize command bits [10:8] when SS_n transitions from 1 to 0
        // Valid commands: 000 (write), 001 (write), 110 (read addr), 111 (read data)
        constraint MOSI_array_c {
            if (prev_SS_n && !SS_n) {
                // Falling edge of SS_n - randomize with valid commands
                MOSI_array[10:8] inside {3'b000, 3'b001, 3'b110, 3'b111};
                // Data bits [7:0] can be any value
            }
            else {
                // Not a falling edge - keep stored value
                MOSI_array == stored_MOSI_array;
            }
        }

        // Constraint 4: tx_valid signal
        // High only for READ_DATA command (111)
        constraint tx_valid_c {
            if (current_cmd == 3'b111) {
                tx_valid == 1;
            }
            else {
                tx_valid == 0;
            }
        }

        // Constraint: tx_data can be any value
        // constraint tx_data_c {
        //     tx_data inside {[8'h00:8'hFF]};
        // }

        // Post-randomize to handle bit-by-bit MOSI assignment and state updates
        function void post_randomize();
            logic is_falling_edge;

            is_falling_edge = (prev_SS_n && !SS_n);

            // Handle SS_n falling edge - start new transaction
            if (is_falling_edge) begin
                stored_MOSI_array = MOSI_array;
                current_cmd = MOSI_array[10:8];
                bit_index = 0;
                count = 0;
                MOSI = stored_MOSI_array[10];  // Start with MSB

                `uvm_info("SLAVE_ITEM", $sformatf(
                    "New Transaction: cmd=%b, MOSI_array=%b, count=
",
                    current_cmd, stored_MOSI_array, count
                ), UVM_HIGH)
            end
            // Handle SS_n active (low) - continue shifting bits
            else if (!SS_n) begin
                if (bit_index < 11) begin
                    MOSI = stored_MOSI_array[10 - bit_index];
                    bit_index++;
                end
                else begin
                    MOSI = 1'b0;  // Default after all bits sent
                end
            end
            // Handle SS_n inactive (high) - idle
            else begin
                MOSI = 1'b0;
                bit_index = 0;
            end

            // Update counter for SS_n timing
            if (current_cmd inside {3'b000, 3'b001 ,3'b110}) begin
                // 13 cycle period
                if (count >= 12) count = 0;
                else count++;
            end
            else if (current_cmd inside { 3'b111}) begin
                // 23 cycle period
                if (count >= 22) count = 0;
                else count++;
            end
            else begin
                // Default: 13 cycle period
                if (count >= 12) count = 0;
                else count++;
            end

            // Update prev_SS_n for next cycle
            prev_SS_n = SS_n;
        endfunction

    endclass
endpackage
```

SLAVE sequencer file :

```
package SLAVE_sequencer_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SLAVE_item_pkg::*;
    class SLAVE_sequencer extends uvm_sequencer #(SLAVE_item);
        `uvm_component_utils(SLAVE_sequencer)

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

SLAVE reset_seq file :

```
package SLAVE_reset_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SLAVE_item_pkg::*;
    class SLAVE_seq_reset extends uvm_sequence #(SLAVE_item);
        `uvm_object_utils(SLAVE_seq_reset)
        SLAVE_item seq_item ;

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

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

SLAVE main_seq file :

```
package SLAVE_seq_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SLAVE_item_pkg::*;
    class SLAVE_seq extends uvm_sequence #(SLAVE_item);
        `uvm_object_utils(SLAVE_seq)
        SLAVE_item seq_item ;

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

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

SLAVE configuration file:

```
package SLAVE_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class SLAVE_config extends uvm_object;
        `uvm_object_utils(SLAVE_config)
        virtual SLAVE_if SLAVE_vif ;
        function new(string name = "SLAVE_config");
            super.new(name) ;
        endfunction
    endclass
endpackage
```

SLAVE env file :

```
package SLAVE_env_pkg;
  import SLAVE_scoreboard_pkg::*;
  import SLAVE_agent_pkg::*;
  import SLAVE_coverage_pkg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class SLAVE_env extends uvm_env;

    `uvm_component_utils(SLAVE_env)
    SLAVE_agent agt ;
    SLAVE_coverage cov ;
    SLAVE_scoreboard sb ;

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

    function void build_phase (uvm_phase phase);
      super.build_phase(phase) ;
      agt = SLAVE_agent::type_id::create("agt",this);
      cov = SLAVE_coverage::type_id::create("cov",this);
      sb  = SLAVE_scoreboard::type_id::create("sb",this);
    endfunction

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

SLAVE agent file :

```
package SLAVE_agent_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SLAVE_sequencer_pkg::*;
    import SLAVE_driver_pkg::*;
    import SLAVE_config_pkg::*;
    import SLAVE_monitor_pkg::*;
    import SLAVE_item_pkg::*;

    class SLAVE_agent extends uvm_agent;
        `uvm_component_utils(SLAVE_agent)
        SLAVE_sequencer sqr ;
        SLAVE_driver drv ;
        SLAVE_monitor mon;
        SLAVE_config SLAVE_cfg;
        uvm_analysis_port #(SLAVE_item) agt_ap;

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

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(SLAVE_config)::get(this, "", "CFG", SLAVE_cfg)) begin
                `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
            end
            sqr = SLAVE_sequencer::type_id::create("sqr",this);
            drv = SLAVE_driver::type_id::create("drv",this);
            mon = SLAVE_monitor::type_id::create("mon",this);
            agt_ap = new("agt_ap",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            drv.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
            mon.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

SLAVE driver file :

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

    class SLAVE_driver extends uvm_driver #(SLAVE_item);
        `uvm_component_utils(SLAVE_driver)
        virtual SLAVE_if SLAVE_vif;
        SLAVE_item stim_seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = SLAVE_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                SLAVE_vif.rst_n = stim_seq_item.rst_n ;
                SLAVE_vif.MOSI = stim_seq_item.MOSI;
                SLAVE_vif.SS_n = stim_seq_item.SS_n;
                SLAVE_vif.tx_valid = stim_seq_item.tx_valid;
                SLAVE_vif.tx_data = stim_seq_item.tx_data;
                @(negedge SLAVE_vif.clk);
                seq_item_port.item_done();
            end
        endtask
    endclass
endpackage
```

SLAVE monitor file :

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

    class SLAVE_monitor extends uvm_monitor;
        `uvm_component_utils(SLAVE_monitor)
        virtual SLAVE_if SLAVE_vif;
        SLAVE_item rsp_seq_item;
        uvm_analysis_port #(SLAVE_item) mon_ap;

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                rsp_seq_item = SLAVE_item::type_id::create("rsp_seq_item");
                rsp_seq_item.rst_n = SLAVE_vif.rst_n ;
                rsp_seq_item.SS_n = SLAVE_vif.SS_n ;
                rsp_seq_item.MOSI = SLAVE_vif.MOSI ;
                rsp_seq_item.tx_valid = SLAVE_vif.tx_valid ;
                rsp_seq_item.tx_data = SLAVE_vif.tx_data ;
                rsp_seq_item.rx_data = SLAVE_vif.rx_data ;
                rsp_seq_item.rx_valid = SLAVE_vif.rx_valid ;
                rsp_seq_item.MISO = SLAVE_vif.MISO ;
                @(negedge SLAVE_vif.clk);
                mon_ap.write(rsp_seq_item);
            end
        endtask
    endclass
endpackage
```

SLAVE scoreboard file :

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

    class SLAVE_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(SLAVE_scoreboard)
        uvm_analysis_export #(SLAVE_item) sb_export;
        uvm_tlm_analysis_fifo #(SLAVE_item) sb_fifo;
        SLAVE_item seq_item_sb;
        logic [9:0]  rx_data_ref;
        logic rx_valid_ref ;
        bit MISO_ref ;

        localparam IDLE      = 3'b000;
        localparam WRITE     = 3'b001;
        localparam CHK_CMD   = 3'b010;
        localparam READ_ADD  = 3'b011;
        localparam READ_DATA = 3'b100;
        logic [2:0] cs, ns;
        logic received_address ;
        logic [3:0] counter;

        int error_count   = 0 ;
        int correct_count = 0 ;

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

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

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

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("SCOREBOARD", $sformatf(
            "Simulation Summary: correct_count=
, error_countorrect_count, error_count
",          ), UVM_NONE)
        endfunction

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item_sb);
                ref_seq_item_chk_model(seq_item_sb);
                if (rx_data_ref === seq_item_sb.rx_data && rx_valid_ref === seq_item_sb.rx_valid && seq_item_sb.MISO === MISO_ref) begin
                    correct_count++;
                end
                else begin
                    `uvm_error("SCOREBOARD", $sformatf("Mismatch: DUT_data=%0h REF_data=%0h ||
                                                        DUT_rx_valid=%b REF_rx_valid=%b ||
                                                        DUT_MISO=%b REF_MISO=%b"
                                    , seq_item_sb.rx_data, rx_data_ref , seq_item_sb.rx_valid ,
                                    rx_valid_ref , seq_item_sb.MISO , MISO_ref));
                    error_count++;
                end
            end
        endtask : run_phase
```

```
        task ref_seq_item_chk_model(SLAVE_item seq_item_chk);
            if (!seq_item_chk.rst_n) begin
                cs = IDLE ;
                rx_data_ref = 0 ;
                rx_valid_ref = 0 ;
                MISO_ref = 0 ;
                counter = 0 ;
                received_address = 0 ;
            end
            else begin
                case (cs)
                    IDLE: begin
                        rx_valid_ref = 0 ;
                        if (seq_item_chk.SS_n) ns = IDLE ;
                        else ns = CHK_CMD ;
                    end
                    CHK_CMD : begin
                        if (seq_item_chk.SS_n) ns = IDLE ;
                        else begin
                            if (!seq_item_chk.MOSI) ns = WRITE ;
                            else begin
                                if (received_address) ns = READ_DATA ;
                                else ns = READ_ADD ;
                            end
                        end
                        counter = 10 ;
                    end
                    WRITE : begin
                        if (seq_item_chk.SS_n) ns = IDLE ;
                        else ns = WRITE ;
                        if (counter > 0) begin
                            rx_data_ref [counter-1] = seq_item_chk.MOSI;
                            counter = counter - 1 ;
                        end
                        else rx_valid_ref = 1 ;
                    end
                    READ_ADD : begin
                        if (seq_item_chk.SS_n) ns = IDLE ;
                        else ns = READ_ADD;
                        if (counter > 0) begin
                            rx_data_ref [counter-1] = seq_item_chk.MOSI;
                            counter = counter - 1 ;
                        end
                        else begin
                            rx_valid_ref = 1 ;
                            received_address = 1 ;
                        end
                    end
                    READ_DATA : begin
                        if (seq_item_chk.SS_n) ns = IDLE ;
                        else ns = READ_DATA ;
                        if (seq_item_chk.tx_valid) begin
                            rx_valid_ref = 0 ;
                            if (counter > 0) begin
                                MISO_ref = seq_item_chk.tx_data[counter-1];
                                counter = counter - 1 ;
                            end
                            else received_address = 0 ;
                        end
                        else begin
                            if (counter > 0) begin
                                rx_data_ref [counter-1] = seq_item_chk.MOSI;
                                counter = counter - 1 ;
                            end
                            else begin
                                rx_valid_ref = 1 ;
                                counter = 8 ;
                            end
                        end
                    end
                endcase
                cs = ns;
            end
        endtask
    endclass

endpackage
```

SLAVE coverage file :

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

    class SLAVE_coverage extends uvm_component;
        `uvm_component_utils(SLAVE_coverage)
        uvm_analysis_export #(SLAVE_item) cov_export;
        uvm_tlm_analysis_fifo #(SLAVE_item) cov_fifo;
        SLAVE_item seq_item_cov;

        covergroup covcode;
            A_cp : coverpoint seq_item_cov.rx_data[9:8] {
                bins rx_00 = {00};
                bins rx_01 = {01};
                bins rx_10 = {10};
                bins rx_11 = {11};
            }

            SS_n: coverpoint seq_item_cov.SS_n {
                bins normal_transaction = (1 => 0 [*12] => 1);
                bins extended_transaction = (1 => 0 [*22] => 1);
                bins communication_on  = {0} ;
            }

            MOSI: coverpoint seq_item_cov.MOSI {
                bins write_address = (0 => 0 => 0);
                bins write_DATA    = (0 => 0 => 1);
                bins read_ADDRESS  = (1 => 1 => 0);
                bins read_DATA     = (1 => 1 => 1);
            }

            cross SS_n , MOSI {
                option.cross_auto_bin_max = 0;
                bins write_add_normal = binsof (MOSI.write_address) && binsof (SS_n.communication_on);
                bins write_data_normal = binsof (MOSI.write_DATA)    && binsof (SS_n.communication_on);
                bins read_add_normal = binsof (MOSI.read_ADDRESS)  && binsof (SS_n.communication_on);
                bins read_data_extended = binsof (MOSI.read_DATA)     && binsof (SS_n.communication_on);
            }
        endgroup

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

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

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

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                covcode.sample();
            end
        endtask
    endclass
endpackage
```

Do file:

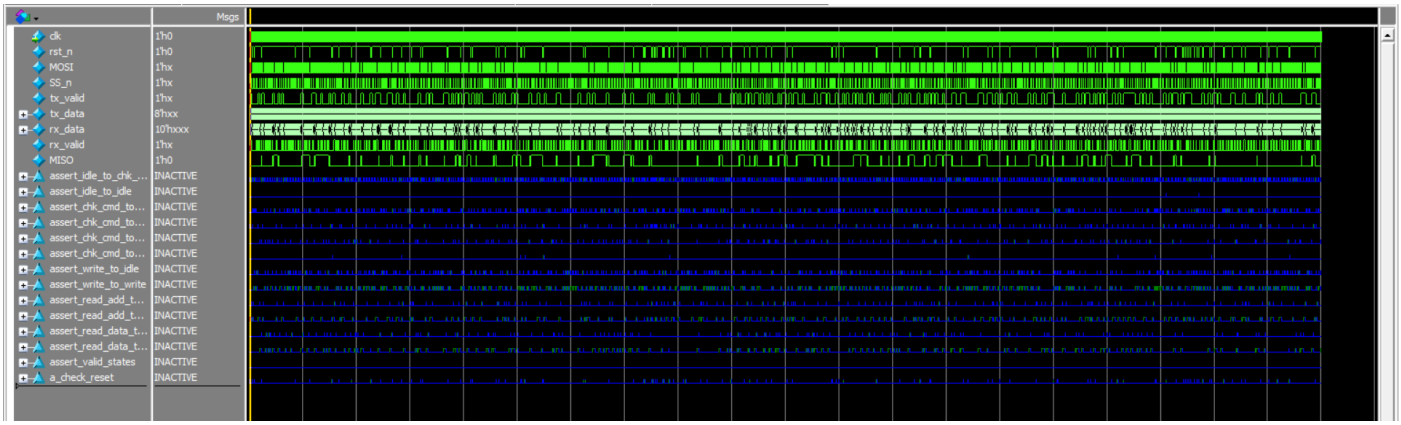
```
vlib work
vlog +define+SIM +cover *.sv
vsim -voptargs=+acc work.SLAVE_top -classdebug -uvmcontrol=all
add wave -position insertpoint sim:/SLAVE_top/inst_if/*
add wave /SLAVE_top/inst_DUT/assert_idle_to_chk_cmd /
SLAVE_top/inst_DUT/assert_idle_to_idle /SLAVE_top/
inst_DUT/assert_chk_cmd_to_write /SLAVE_top/inst_DUT/
assert_chk_cmd_to_read_add /SLAVE_top/inst_DUT/
assert_chk_cmd_to_read_data /SLAVE_top/inst_DUT/
assert_chk_cmd_to_idle /SLAVE_top/inst_DUT/assert_write_to_idle /SLAVE_top
/inst_DUT/assert_write_to_write /SLAVE_top/inst_DUT/assert_read_add_to_idle
/SLAVE_top/inst_DUT/assert_read_add_to_read_add /SLAVE_top/inst_DUT/
 assert_read_data_to_idle /SLAVE_top/inst_DUT/assert_read_data_to_read_data
 /SLAVE_top/inst_DUT/assert_valid_states /SLAVE_top/inst_DUT/a_check_reset
coverage save -instance /SLAVE_top/inst_DUT SLAVE.ucdb -onexit
run -all
```

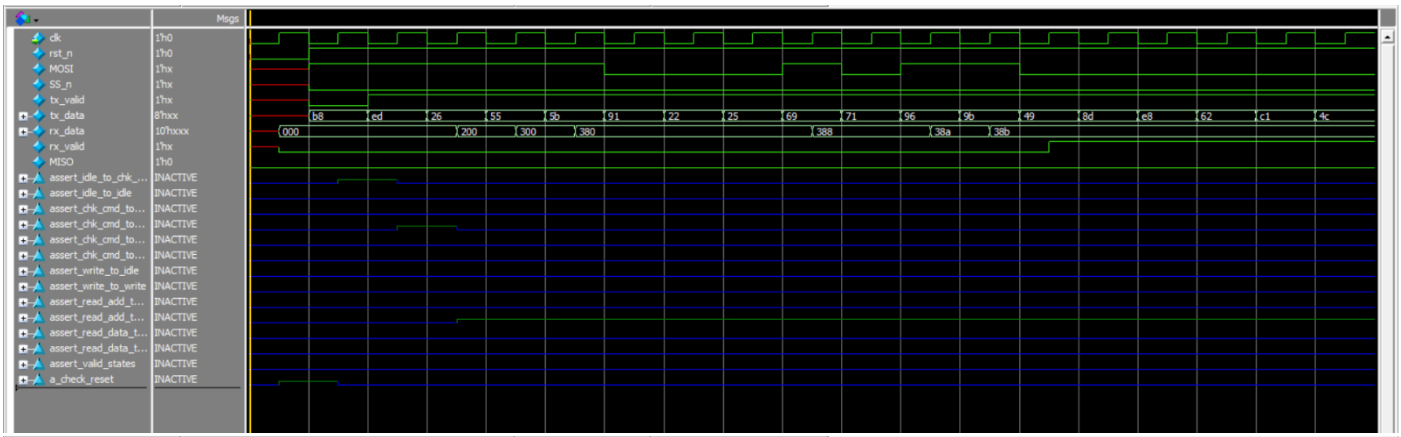
Second: Snippets from Simulation results:

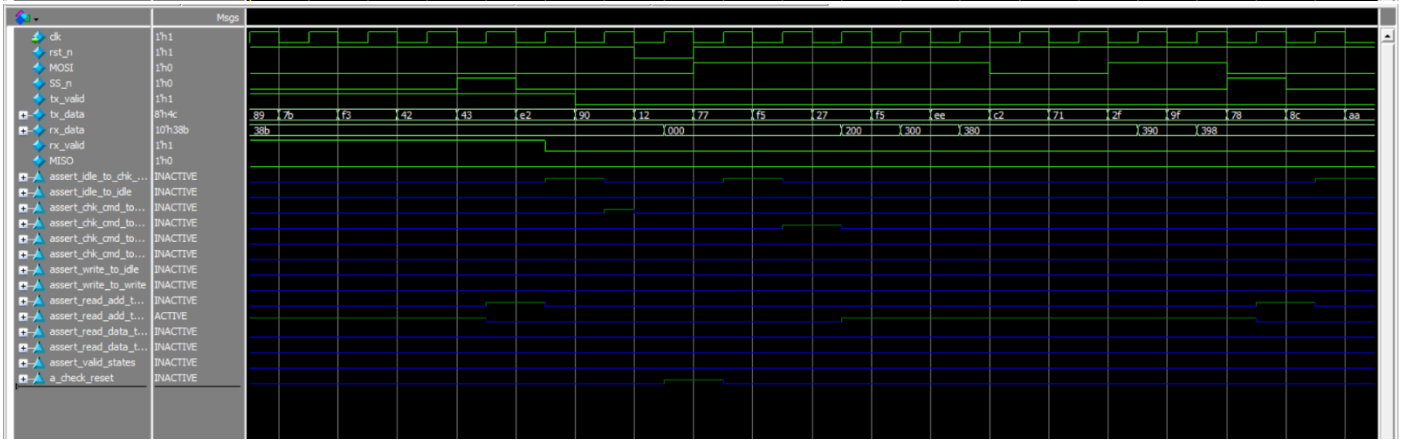
Transcript:

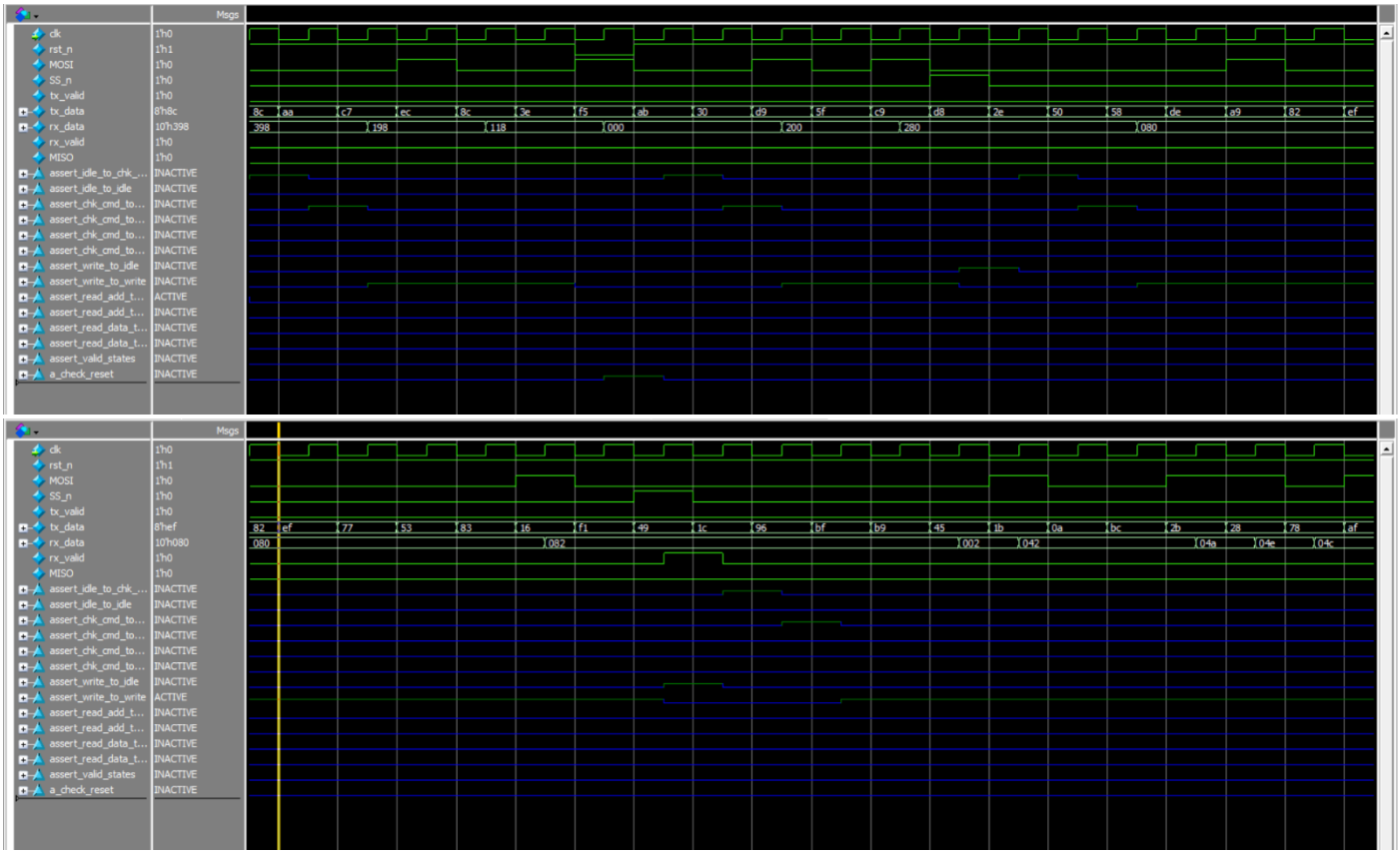
```
# UVM_INFO SLAVE_scoreboard.sv(44) @ 20002: uvm_test_top.env.sb [SCOREBOARD] Simulation Summary: correct_count=10001, error_count=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     1
# [TEST_DONE]     1
# [run_phase]     4
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 20002 ns  Iteration: 61  Instance: /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

VSIM 2>
```

Waveforms:







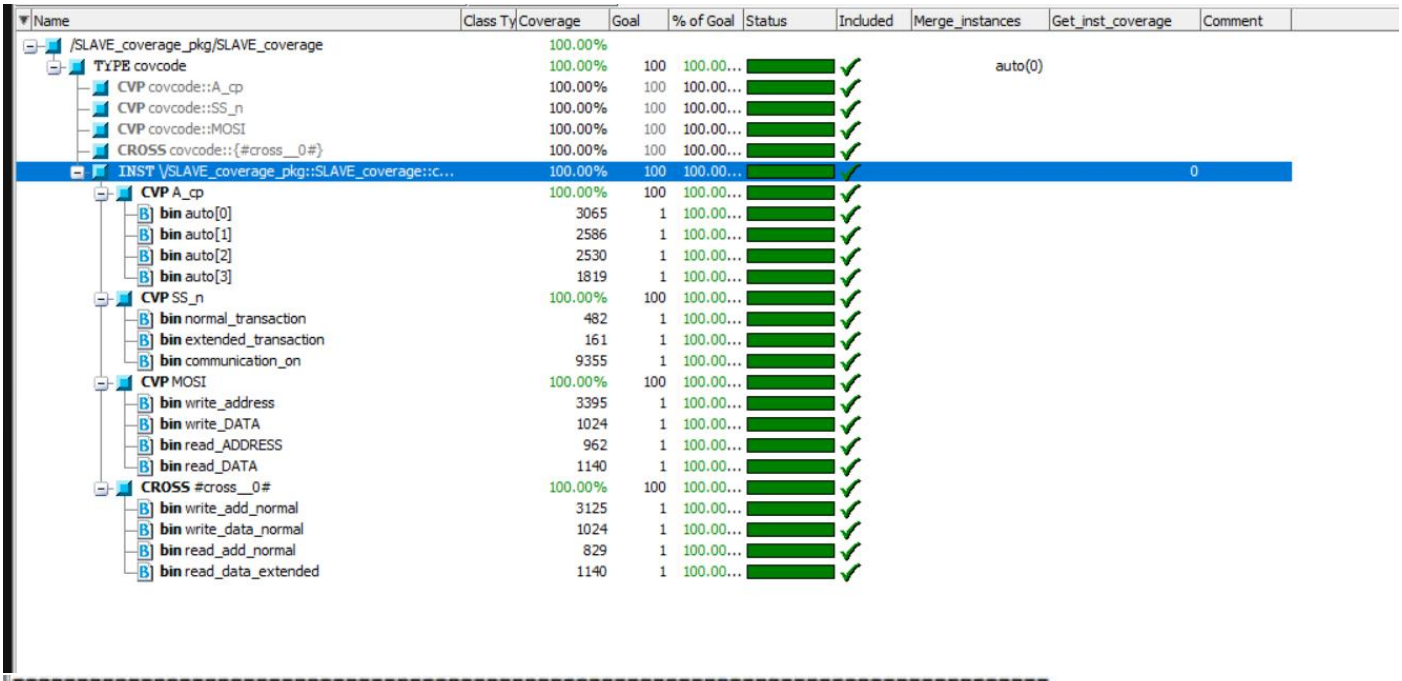


Assertions coverage:

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/SLAVE_top/inst_DUT/cover_idle_to_chk_cmd	SVA	✓	Off	733	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_chk_cmd_to_write	SVA	✓	Off	381	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_chk_cmd_to_read_add	SVA	✓	Off	179	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_chk_cmd_to_read_data	SVA	✓	Off	154	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_write_to_idle	SVA	✓	Off	341	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_read_add_to_idle	SVA	✓	Off	144	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/cover_read_data_to_idle	SVA	✓	Off	134	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_top/inst_DUT/c_check_reset	SVA	✓	Off	114	1	Unli...	1	100%		✓	0	0	0 ns	0
/SLAVE_seq_pkg::SLAVE_seq::body/#ublk#29478439#15/immed__17			Immediate	SVA	on		0	1	-	-	-	-	off	assert (randomize(...))
/SLAVE_top/inst_DUT/assert_idle_to_chk_cmd			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_idle_to_idle			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_chk_cmd_to_write			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_chk_cmd_to_read_add			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_chk_cmd_to_read_data			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_chk_cmd_to_idle			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_write_to_idle			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_write_to_write			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_read_add_to_idle			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_read_add_to_read_add			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_read_data_to_idle			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_read_data_to_read_data			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/assert_valid_states			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) disable iff (...
/SLAVE_top/inst_DUT/a_check_reset			Concurrent	SVA	on		0	1	-	0B	0B	0 ns	0 off	assert(@(posedge clk) (~rst_n)|=...

```
# ================================================================================
# === Instance: /SLAVE_top/inst_DUT
# === Design Unit: work.SLAVE
# ================================================================================
#
# Assertion Coverage:
#     Assertions                      14        14         0   100.00%
# --------------------------------------------------------------------------------
# Name                 File(Line)                    Failure      Pass
#                                                    Count        Count
# --------------------------------------------------------------------------------
# /SLAVE_top/inst_DUT/assert_idle_to_chk_cmd
#                      SPI_slave.sv(220)                    0          1
# /SLAVE_top/inst_DUT/assert_idle_to_idle
#                      SPI_slave.sv(223)                    0          1
# /SLAVE_top/inst_DUT/assert_chk_cmd_to_write
#                      SPI_slave.sv(226)                    0          1
# /SLAVE_top/inst_DUT/assert_chk_cmd_to_read_add
#                      SPI_slave.sv(229)                    0          1
# /SLAVE_top/inst_DUT/assert_chk_cmd_to_read_data
#                      SPI_slave.sv(232)                    0          1
# /SLAVE_top/inst_DUT/assert_chk_cmd_to_idle
#                      SPI_slave.sv(235)                    0          1
# /SLAVE_top/inst_DUT/assert_write_to_idle
#                      SPI_slave.sv(238)                    0          1
# /SLAVE_top/inst_DUT/assert_write_to_write
#                      SPI_slave.sv(241)                    0          1
# /SLAVE_top/inst_DUT/assert_read_add_to_idle
#                      SPI_slave.sv(244)                    0          1
# /SLAVE_top/inst_DUT/assert_read_add_to_read_add
#                      SPI_slave.sv(247)                    0          1
# /SLAVE_top/inst_DUT/assert_read_data_to_idle
#                      SPI_slave.sv(250)                    0          1
# /SLAVE_top/inst_DUT/assert_read_data_to_read_data
#                      SPI_slave.sv(253)                    0          1
# /SLAVE_top/inst_DUT/assert_valid_states
#                      SPI_slave.sv(256)                    0          1
# /SLAVE_top/inst_DUT/a_check_reset
#                      SPI_slave.sv(259)                    0          1
```

Function coverage:

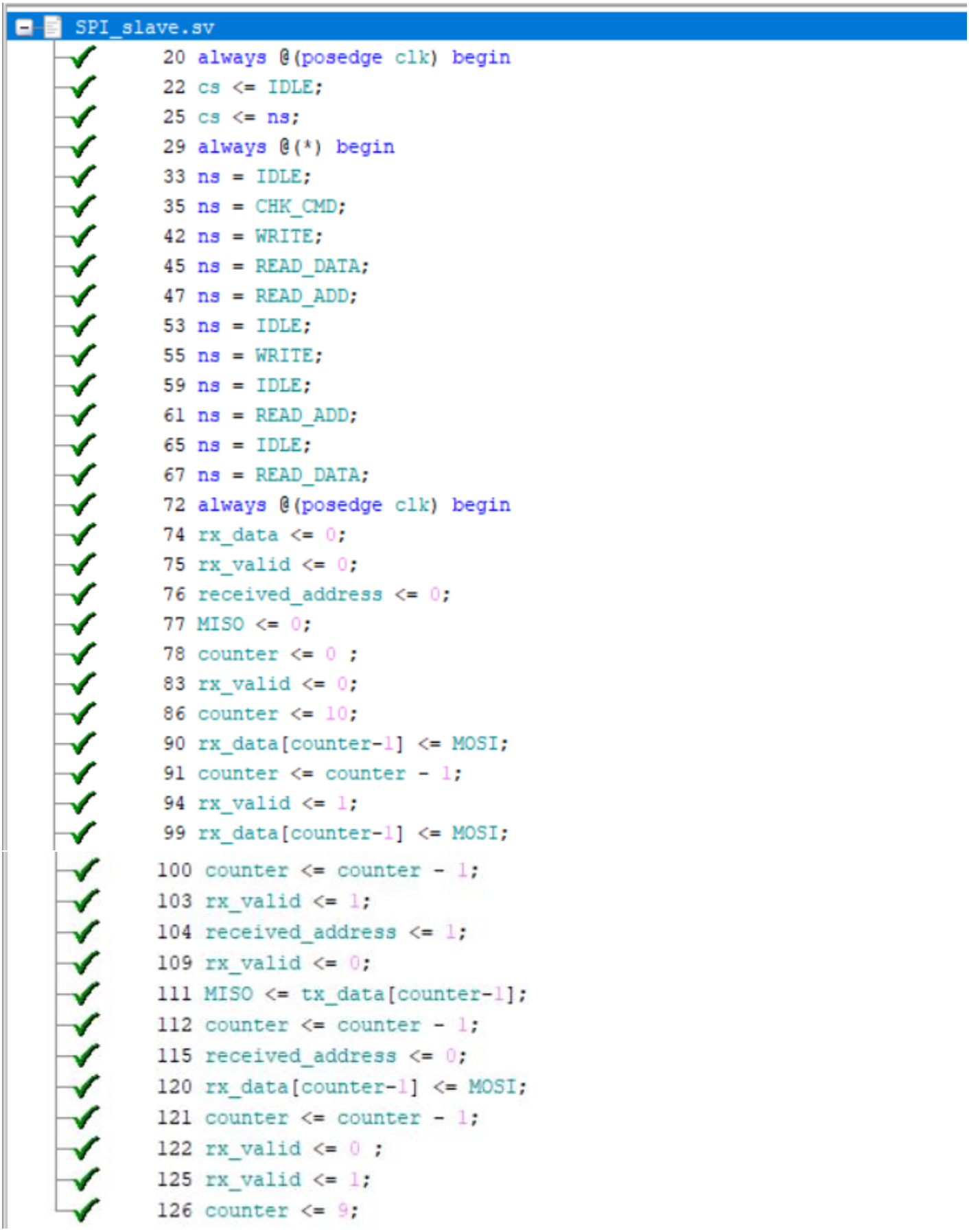


Name	Class Ty	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/SLAVE_coverage_pkg/SLAVE_coverage		100.00%							
TYPE covcode		100.00%	100	100.00...		✓	auto(0)		
CVP covcode::A_cp		100.00%	100	100.00...		✓			
CVP covcode::SS_n		100.00%	100	100.00...		✓			
CVP covcode::MOSI		100.00%	100	100.00...		✓			
CROSS covcode::{#cross__0#}		100.00%	100	100.00...		✓			
INST \/SLAVE_coverage_pkg::SLAVE_coverage::c...		100.00%	100	100.00...		✓			0
CVP A_cp		100.00%	100	100.00...		✓			
bin auto[0]		3065	1	100.00...		✓			
bin auto[1]		2586	1	100.00...		✓			
bin auto[2]		2530	1	100.00...		✓			
bin auto[3]		1819	1	100.00...		✓			
CVP SS_n		100.00%	100	100.00...		✓			
bin normal_transaction		482	1	100.00...		✓			
bin extended_transaction		161	1	100.00...		✓			
bin communication_on		9355	1	100.00...		✓			
CVP MOSI		100.00%	100	100.00...		✓			
bin write_address		3395	1	100.00...		✓			
bin write_DATA		1024	1	100.00...		✓			
bin read_ADDRESS		962	1	100.00...		✓			
bin read_DATA		1140	1	100.00...		✓			
CROSS #cross__0#		100.00%	100	100.00...		✓			
bin write_add_normal		3125	1	100.00...		✓			
bin write_data_normal		1024	1	100.00...		✓			
bin read_add_normal		829	1	100.00...		✓			
bin read_data_extended		1140	1	100.00...		✓			

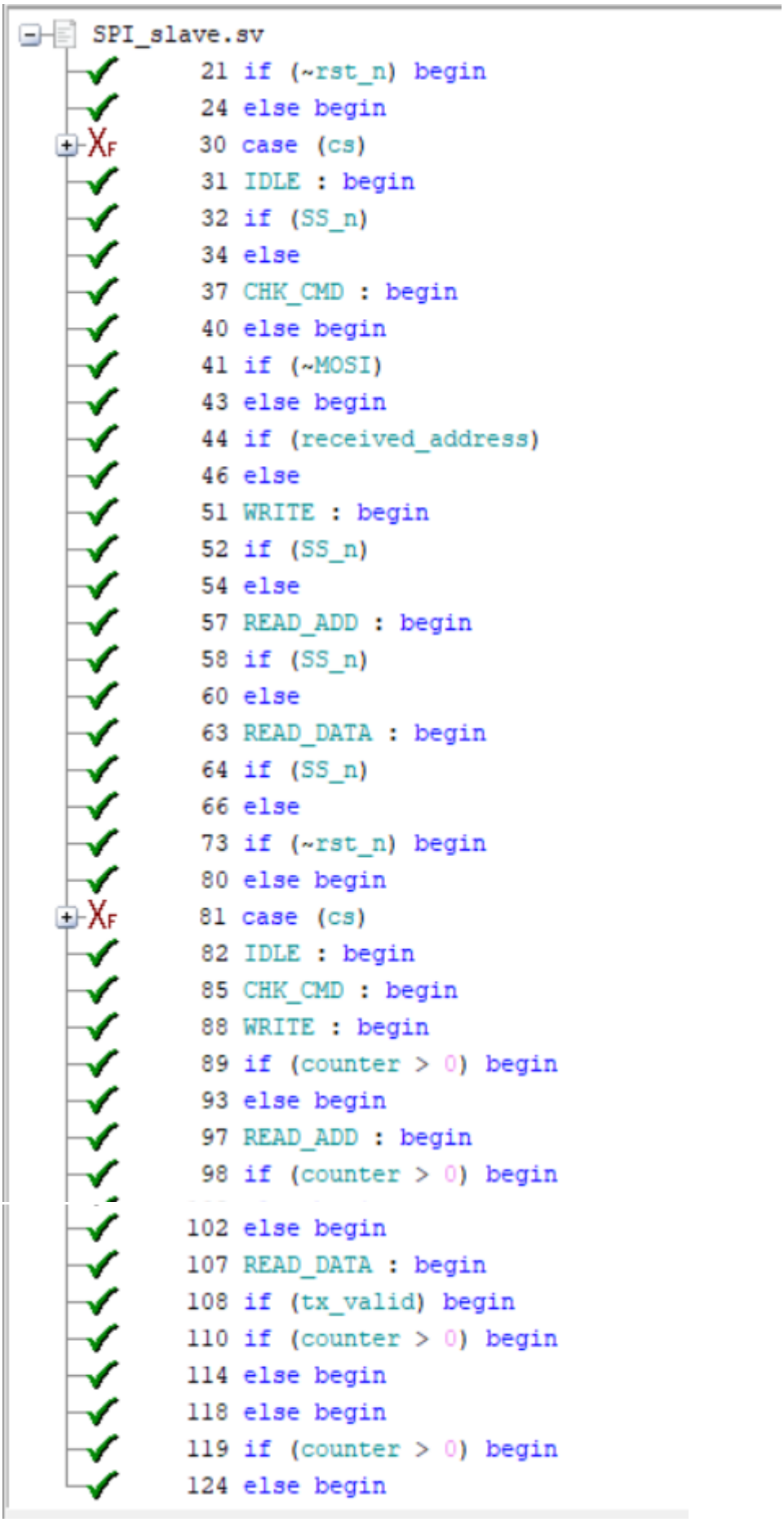
```
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses     4        na        na        na
            Covergroup Bins    15        15         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
 TYPE /SLAVE_coverage_pkg/SLAVE_coverage/covcode    100.00%        100          -    Covered
    covered/total bins:                                  15         15          -
    missing/total bins:                                   0         15          -
    % Hit:                                           100.00%        100          -
    Coverpoint A_cp                                  100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
    Coverpoint SS_n                                  100.00%        100          -    Covered
        covered/total bins:                                3          3          -
        missing/total bins:                                0          3          -
        % Hit:                                       100.00%        100          -
    Coverpoint MOSI                                  100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
    Cross #cross__0#                                 100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
```

Code coverage:

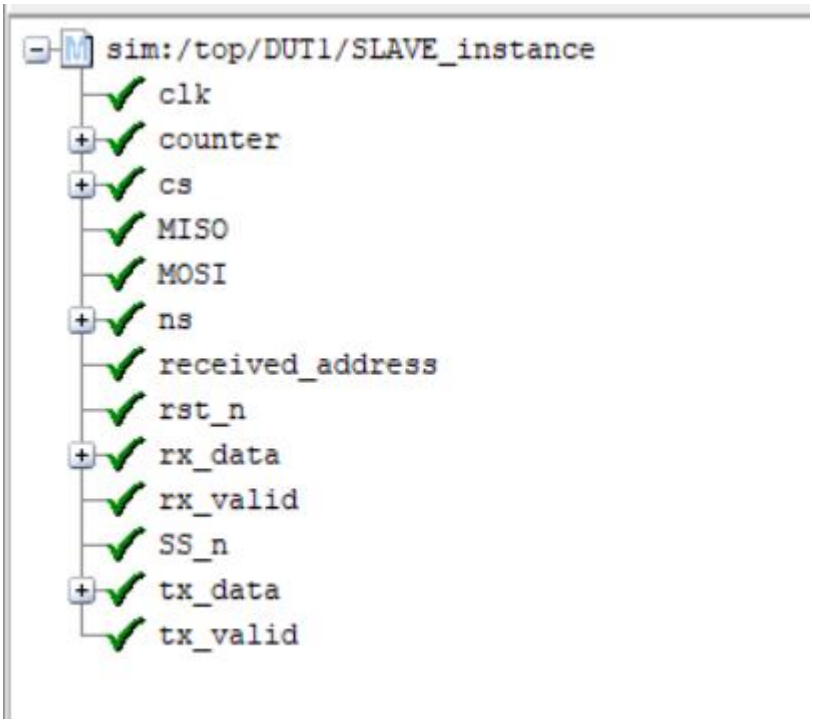
-Statements:



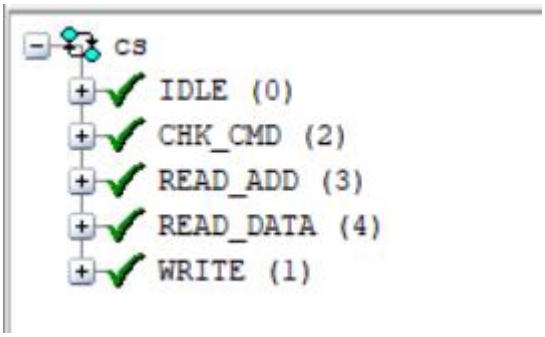
-Branches:


SPI_slave.sv
21 if (~rst_n) begin
24 else begin
30 case (cs)
31 IDLE : begin
32 if (SS_n)
34 else
37 CHK_CMD : begin
40 else begin
41 if (~MOSI)
43 else begin
44 if (received_address)
46 else
51 WRITE : begin
52 if (SS_n)
54 else
57 READ_ADD : begin
58 if (SS_n)
60 else
63 READ_DATA : begin
64 if (SS_n)
66 else
73 if (~rst_n) begin
80 else begin
81 case (cs)
82 IDLE : begin
85 CHK_CMD : begin
88 WRITE : begin
89 if (counter > 0) begin
93 else begin
97 READ_ADD : begin
98 if (counter > 0) begin
102 else begin
107 READ_DATA : begin
108 if (tx_valid) begin
110 if (counter > 0) begin
114 else begin
118 else begin
119 if (counter > 0) begin
124 else begin

-Toggles:



-FSMs:



Verification Plan

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SLAVE_0	Reset Behavior: When the active-low reset (rst_n) is asserted, All output signals reset to 0.	Randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the score board to make sure the output is correct and cocurrent assertions to check it too.
SLAVE_1	The SS_n signal to be high for one cycle every 13 cycles for all cases except read data to be high for one cycle every 23 cycles	Randomized during simulation due to constrains in the main sequence class	Added coverpoints to : • Check full transaction duration: 1 → 0 [*13] → 1 for normal operations. • Check extended transaction: 1 → 0 [*23] → 1 for READ_DATA.	Output Checked against refrence model in the scoreboard and SVA inside SVA file to check internal signals
SLAVE_2	Declare a randomized array of 11 bits to drive bit by bit the MOSI and the first 3 bits sent serially to the MOSI when the SS_n falls are valid combinations only (000, 001, 110, 111)	Randomized during simulation.	Added coverpoints on MOSI to validate correct transitions: • 000 (Write Address) • 001 (Write Data) • 110 (Read Address) • 111 (Read Data)	Output Checked against refrence model in the scoreboard and SVA inside SVA file to check internal signals
RAM_3	The tx_valid signal to be high in case of read data	Randomized with constraint that drive the inputs to match required specifications		Output Checked against refrence model in the scoreboard and SVA inside SVA file to check internal signals

RAM part

First: Snippets from codes:

Design and bugs found:

- Bug

```
else
        if (rx_valid) begin
            case (din[9:8])
                2'b00 : Wr_Addr <= din[7:0];
                2'b01 : MEM[Wr_Addr] <= din[7:0];
                2'b10 : Rd_Addr <= din[7:0];
                2'b11 : dout <= MEM[Wr_Addr];
                default : dout <= 0;
            endcase
        end
        tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;

```

```
else  begin
            if (rx_valid) begin //add begin/end
                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];//bug fixed
                    default : dout <= 0;
                endcase
                tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;
            end
        end
```

• Read Pointer Address Handling:
During the `READ_DATA` operation, the address source is switched from the write pointer to the read pointer. This ensures that data is fetched from the correct memory location corresponding to the most recently stored read address, maintaining proper data integrity and preventing overlap with ongoing write operations.

Complete design without bugs :

```
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;

    integer i ;
    always @(posedge clk) begin
        if (~rst_n) begin
            dout     <= 0;
            tx_valid <= 0;
            Rd_Addr  <= 0;
            Wr_Addr  <= 0;
        end
        else  begin
            if (rx_valid) begin //add begin/end
                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];//bug fixed
                    default : dout <= 0;
                endcase
                tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;
            end
        end
    end
endmodule
```

RAM golden model :

```
module RAM_golden (din,clk,rst_n,rx_valid,dout_ref,tx_valid_ref);

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

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

    output reg [7:0] dout_ref;
    output reg tx_valid_ref;

    reg [ADDR_SIZE-1 : 0] addr_rd, addr_wr;
    reg [7:0] mem [MEM_DEPTH-1 : 0];

    integer i;
    always @(posedge clk ) begin
        if (!rst_n) begin
            dout_ref     <= 0 ;
            tx_valid_ref <= 0 ;
            addr_rd  <= 0 ;
            addr_wr  <= 0 ;
        end
        else begin
            if (rx_valid) begin
                case (din[9:8])
                    2'b00 : begin
                        addr_wr       <= din[7:0] ;
                        tx_valid_ref  <= 1'b0 ;
                    end
                    2'b01 : begin
                        mem[addr_wr]  <= din[7:0] ;
                        tx_valid_ref  <= 1'b0 ;
                    end
                    2'b10 : begin
                        addr_rd      <= din[7:0] ;
                        tx_valid_ref <= 1'b0 ;
                    end
                    2'b11 : begin
                        dout_ref     <= mem[addr_rd] ;
                        tx_valid_ref <= 1'b1 ;
                    end
                    default: dout_ref <= 0;
                endcase
            end
        end
    end
endmodule
```

RAM assertion file :

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

    // 1) reset
    property p_reset;
        @(posedge clk)
            (!rst_n) |=> (dout == 0 && tx_valid == 0);
    endproperty
    a_reset: assert property (p_reset)
        else $error("Assertion 1 failed");
    c_reset: cover property (p_reset);

    // 2) tx_valid is low
    property p_tx_low;
        @(posedge clk) disable iff (!rst_n)
        ( (!din[9]) && (!din[8]) && rx_valid) |=> (tx_valid == 0);
    endproperty
    a_tx_low: assert property (p_tx_low)
        else $error("Assertion 2 failed");
    c_tx_low: cover property (p_tx_low);

    // 3) tx_valid rises then falls
    property p_tx_high;
        @(posedge clk) disable iff (!rst_n)
            ((din[8] && din[9]) && rx_valid) |=> (tx_valid == 1) |=> $fell(tx_valid)[->1] ;
    endproperty
    a_tx_high: assert property (p_tx_high)
    else $error("Assertion 3 failed");
    c_tx_high: cover property (p_tx_high);

    // 4) write address => write data
    property p_write;
        @(posedge clk) disable iff (!rst_n)
            (rx_valid && din[9:8] == 2'b00) |=>##1 (din[9:8] inside {2'b00,2'b01});
    endproperty
    a_write: assert property (p_write) else $error("Assertion write failed");
    c_write: cover property (p_write);

    // 5) read address => read data
    property p_read;
        @(posedge clk) disable iff (!rst_n)
            (din[9:8] == 2'b10) |=>##[0:2] ( din[9:8] inside {2'b10,2'b11});
    endproperty
    a_read: assert property (p_read) else $error("Assertion read failed ");
    c_read: cover property (p_read);

endmodule
```

RAM Interface :

```
interface RAM_intf (input logic clk);
  parameter MEM_DEPTH = 256;
  parameter ADDR_SIZE = 8;

  // Inputs
  logic rst_n;
  logic [9:0] din;
  logic rx_valid;

  // Outputs DUT
  logic tx_valid;
  logic [7:0] dout;

  // Outputs Golden Model
  logic tx_valid_ref;
  logic [7:0] dout_ref;

  // Modports
  modport SVA (
    input  clk, rst_n, din, rx_valid,
    input  dout, tx_valid
  );
endinterface : RAM_intf

```

RAM Top module:

```
1  import uvm_pkg::*;
2  `include "uvm_macros.svh"
3  import RAM_test_pkg::*;
4
5  module top();
6    // Clock generation
7    bit clk = 0;
8    initial begin
9      forever
10       #1 clk =~clk ;
11   end
12
13   // Instantiate the interface and DUT
14   RAM_intf RAM_if (clk);
15
16   RAM DUT
17   (
18     .clk         (RAM_if.clk),
19     .rst_n       (RAM_if.rst_n),
20     .din         (RAM_if.din),
21     .rx_valid    (RAM_if.rx_valid),
22     .tx_valid    (RAM_if.tx_valid),
23     .dout        (RAM_if.dout)
24   );
25
26   // Golden Model
27   RAM_golden GOLDEN
28   (
29     .clk             (RAM_if.clk),
30     .rst_n           (RAM_if.rst_n),
31     .din             (RAM_if.din),
32     .rx_valid        (RAM_if.rx_valid),
33     .tx_valid_ref    (RAM_if.tx_valid_ref),
34     .dout_ref        (RAM_if.dout_ref)
35   );
36
37   //Assertions
38   bind RAM RAM_sva sva_inst (RAM_if.SVA);
39
40   initial begin
41     uvm_config_db# (virtual RAM_intf)::set(null,"","RAM_IF", RAM_if) ;
42     run_test ("RAM_test") ;
43   end
44 endmodule
45
```

RAM test file:

```
package RAM_test_pkg;
  import RAM_env_pkg::*;
  import seq_item_pkg::*;
  import RAM_sequencer_pkg::*;
  import scoreboard_pkg::*;
  import RAM_read_only_seq_pkg::*;
  import RAM_write_only_seq_pkg::*;
  import RAM_read_write_seq_pkg::*;
  import RAM_reset_seq_pkg::*;
  import pkg_cfg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class RAM_test extends uvm_test;
    `uvm_component_utils(RAM_test)

    RAM_config_obj RAM_cfg ;
    virtual RAM_if RAM_vif ;
    RAM_env env ;
    RAM_read_only_seq read_only_seq ;
    RAM_write_only_seq write_only_seq ;
    RAM_read_write_seq read_write_seq ;
    RAM_reset_seq reset_seq ;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      env               = RAM_env             :: type_id :: create ("env",this) ;
      RAM_cfg           = RAM_config_obj      :: type_id :: create("RAM_cfg");
      reset_seq         = RAM_reset_seq       :: type_id :: create("reset_seq",this);
      read_only_seq     = RAM_read_only_seq   :: type_id :: create("read_only_seq",this);
      write_only_seq    = RAM_write_only_seq  :: type_id :: create("write_only_seq",this);
      read_write_seq    = RAM_read_write_seq  :: type_id :: create("read_only_seq",this);

      if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_config_vif))begin
        `uvm_fatal("build_phase", "test -unable to get the virtual interface");
      end
      uvm_config_db #(RAM_config_obj)::set(this,"*", "CFG", RAM_cfg);
    endfunction

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

      //reset sequence
      `uvm_info("run_phase","reset_asserted",UVM_LOW)
      reset_seq.start(env.agt.sqr);
      `uvm_info("run_phase","reset_deasserted",UVM_LOW)

      // create and start write only sequence
      `uvm_info("run_phase", "Starting write only sequence...", UVM_MEDIUM);
      write_only_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "write only sequence finished.", UVM_MEDIUM);

      // create and start read/write sequence
      `uvm_info("run_phase", "Starting read/write sequence...", UVM_MEDIUM);
      read_write_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "read/write sequence finished.", UVM_MEDIUM);

      // create and start read only sequence
      `uvm_info("run_phase", "Starting read only sequence...", UVM_MEDIUM);
      read_only_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "read only sequence finished.", UVM_MEDIUM);

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

RAM seq_item file:

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

    class RAM_seq_item extends uvm_sequence_item;
        `uvm_object_utils(RAM_seq_item)

        parameter MEM_DEPTH = 256;
        parameter ADDR_SIZE = 8;

        //input signals
        rand bit  rst_n;
        rand bit  rx_valid;
        rand bit  [9 : 0] din;
        //output signals
        logic tx_valid ;
        logic tx_valid_ref ;
        logic [7 : 0] dout;
        logic [7 : 0] dout_ref;

        bit [1:0] prev_op = 2'b00;
        // 1 for the read data operation and 0 for the read address operation

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

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

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

        function void post_randomize();
            prev_op = din[9:8];
        endfunction
        //-----------------constraints---------------

        // The reset signal (rst_n) shall be deasserted most of the time.
        constraint c_rst {rst_n dist {1 := 99 , 0 := 1};};

        //The rx_valid signal shall be asserted most of time.
        constraint c_rx {rx_valid dist {1 := 98 , 0 := 2};};

        // For a write-only sequence, every Write Address operation shall always be followed by either
        //Write Address or Write Data operation.
        constraint c_write_only { din[9:8] inside {2'b00, 2'b01};};

        //  For a read-only sequence, only Read Address or Read Data operations are allowed.
        constraint c_read_only
        {
            if (prev_op == 2'b10)          // Read address => Read data
            {
                din[9:8] == 2'b11;
            }
            else if (prev_op == 2'b11)     // Read data (2'b11) => Read address
            {
                din[9:8] == 2'b10;
            }
            else
            {
                din[9:8] inside {2'b11, 2'b10};
            }
        };

        //  For a read-write sequence
        constraint c_read_write
        {
            if (prev_op == 2'b00)                           //After Write Address operation
            {
                din[9:8] inside { 2'b00, 2'b01};
            }
            else if (prev_op == 2'b01)                      //After Write Data operation
            {
                din[9:8] dist { 2'b00 := 40, 2'b10 := 60};
            }
            else if (prev_op == 2'b10)                      //After Read Address operation
            {
                din[9:8] inside { 2'b10, 2'b11};
            }
            else                                            //After Read Data operation
            {
                din[9:8] dist {  2'b00 := 60, 2'b10 := 40};
            }
        };
    endclass
endpackage
```

RAM sequencer file :

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

    class RAM_sequencer extends uvm_sequencer #(RAM_seq_item) ;
        `uvm_component_utils(RAM_sequencer)

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

RAM reset_seq file :

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

    class RAM_reset_seq extends uvm_sequence #(RAM_seq_item) ;
    `uvm_object_utils(RAM_reset_seq)

    RAM_seq_item seq_item ;

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

    task body();
            seq_item = RAM_seq_item::type_id::create("seq_item");
            start_item(seq_item);

            seq_item.constraint_mode(0);
            assert(seq_item.randomize() with { seq_item.rst_n == 0; seq_item.din[9:8] == 2'b00;});

            finish_item(seq_item);
    endtask
 endclass
endpackage
```

RAM write_seq file :

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

    class RAM_write_only_seq extends uvm_sequence #(RAM_seq_item) ;
    `uvm_object_utils(RAM_write_only_seq)

    RAM_seq_item seq_item ;

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

    task body();
            seq_item = RAM_seq_item::type_id::create("seq_item");
        repeat(10000) begin
            start_item(seq_item);

            seq_item.constraint_mode(0);
            seq_item.c_rx.constraint_mode(1);
            seq_item.c_rst.constraint_mode(1);
            seq_item.c_write_only.constraint_mode(1);
            assert (seq_item.randomize());

            finish_item(seq_item);
        end
    endtask
 endclass
endpackage
```

RAM read_seq file :

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

    class RAM_read_only_seq extends uvm_sequence #(RAM_seq_item) ;
`uvm_object_utils(RAM_read_only_seq)

    RAM_seq_item seq_item ;

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

    task body();
        seq_item = RAM_seq_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            seq_item.constraint_mode(0);
            seq_item.c_rx.constraint_mode(1);
            seq_item.c_rst.constraint_mode(1);
            seq_item.c_read_only.constraint_mode(1);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
 endclass
endpackage
```

RAM read_write_seq file :

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

    class RAM_read_write_seq extends uvm_sequence #(RAM_seq_item) ;
`uvm_object_utils(RAM_read_write_seq)

    RAM_seq_item seq_item ;

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

    task body();
        seq_item = RAM_seq_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            seq_item.constraint_mode(0);
            seq_item.c_rx.constraint_mode(1);
            seq_item.c_rst.constraint_mode(1);
            seq_item.c_read_write.constraint_mode(1);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
 endclass
endpackage
```

RAM configuration file:

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

    class RAM_config_obj extends uvm_object;
        `uvm_object_utils(RAM_config_obj)

        virtual RAM_intf RAM_config_vif  ;

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

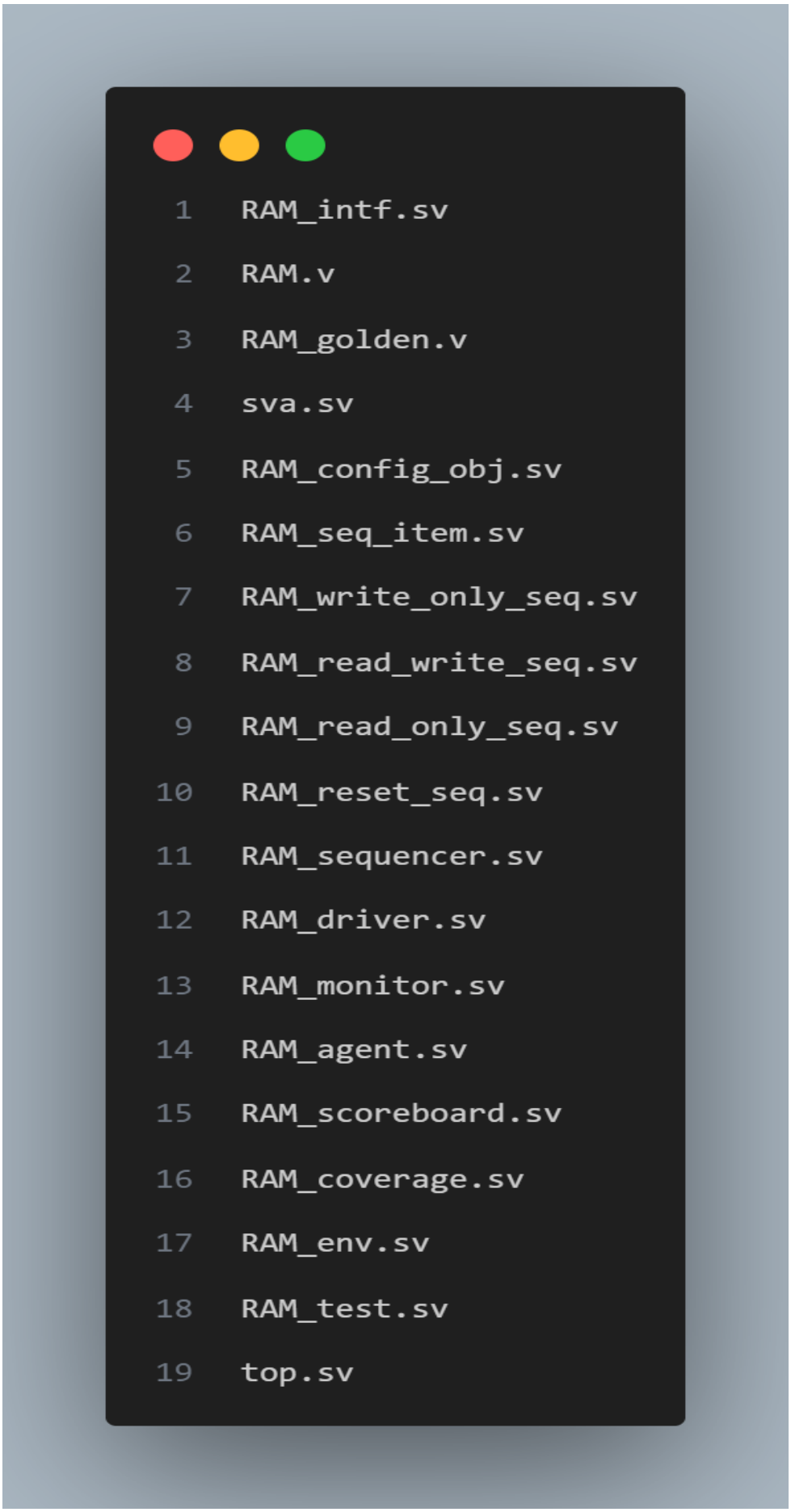
RAM env file :

```
package RAM_env_pkg;
  import scoreboard_pkg::*;
  import agent_pkg::*;
  import RAM_coverage_pkg::*;
  import RAM_sequencer_pkg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class RAM_env extends uvm_env;
    `uvm_component_utils(RAM_env)

    RAM_agent agt ;
    RAM_coverage cov ;
    RAM_scoreboard sb ;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      agt = RAM_agent      :: type_id :: create("agt", this);
      cov = RAM_coverage   :: type_id :: create("cov", this);
      sb  = RAM_scoreboard :: type_id :: create("sb", this);
    endfunction : build_phase

    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
```

Do file :

```
vlib work
vlog -f src_files.list +cover +covercells +coveropt
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
vsim -sv_seed 1906057213 work.top
add wave -r sim:/top/RAM_if/*
coverage save RAM.ucdb -onexit
run -all
```

Src file :


1 RAM_intf.sv
2 RAM.v
3 RAM_golden.v
4 sva.sv
5 RAM_config_obj.sv
6 RAM_seq_item.sv
7 RAM_write_only_seq.sv
8 RAM_read_write_seq.sv
9 RAM_read_only_seq.sv
10 RAM_reset_seq.sv
11 RAM_sequencer.sv
12 RAM_driver.sv
13 RAM_monitor.sv
14 RAM_agent.sv
15 RAM_scoreboard.sv
16 RAM_coverage.sv
17 RAM_env.sv
18 RAM_test.sv
19 top.sv

RAM agent file :

```
package agent_pkg;
    import pkg_cfg::*;
    import seq_item_pkg::*;
    import RAM_sequencer_pkg::*;
    import RAM_monitor_pkg::*;
    import pkg_driver::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    class RAM_agent extends uvm_agent;
        `uvm_component_utils(RAM_agent)

        RAM_driver drv ;
        RAM_monitor mon ;
        RAM_config_obj RAM_cfg ;
        RAM_sequencer sqr  ;
        uvm_analysis_port #(RAM_seq_item) agt_ap ;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(RAM_config_obj)::get(this, "", "CFG", RAM_cfg))
              `uvm_fatal("build_phase", " unable to get configuration object");

            sqr = RAM_sequencer :: type_id :: create("sqr", this) ;
            drv = RAM_driver    :: type_id :: create("drv", this) ;
            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);
                drv.RAM_vif  = RAM_cfg.RAM_config_vif ;
                mon.RAM_vif  = RAM_cfg.RAM_config_vif ;
                drv.seq_item_port.connect(sqr.seq_item_export);
                mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage

```

RAM driver file :

```
package pkg_driver;
    import pkg_cfg::*;
    import seq_item_pkg::*;
    import RAM_read_only_seq_pkg::*;
    import RAM_write_only_seq_pkg::*;
    import RAM_read_write_seq_pkg::*;
    import RAM_reset_seq_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    class RAM_driver extends uvm_driver#(RAM_seq_item);
        `uvm_component_utils(RAM_driver)

        virtual RAM_intf RAM_vif ;
        RAM_seq_item stim_seq_item ;

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

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

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

                @(negedge RAM_vif.clk);
                RAM_vif.rst_n       =  stim_seq_item.rst_n ;
                RAM_vif.din         =  stim_seq_item.din ;
                RAM_vif.rx_valid    =  stim_seq_item.rx_valid ;

                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
            end
        endtask

    endclass
endpackage
```

RAM monitor file :

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

    class RAM_monitor extends uvm_monitor ;
    `uvm_component_utils(RAM_monitor)

    RAM_seq_item rsp_seq_item ;
    virtual RAM_intf RAM_vif ;
    uvm_analysis_port #(RAM_seq_item) mon_ap ;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = RAM_seq_item::type_id::create ("rsp_seq_item");

            rsp_seq_item.rst_n         = RAM_vif.rst_n ;
            rsp_seq_item.din           = RAM_vif.din ;
            rsp_seq_item.rx_valid      = RAM_vif.rx_valid ;
            rsp_seq_item.dout          = RAM_vif.dout ;
            rsp_seq_item.tx_valid      = RAM_vif.tx_valid ;
            rsp_seq_item.dout_ref      = RAM_vif.dout_ref ;
            rsp_seq_item.tx_valid_ref  = RAM_vif.tx_valid_ref ;

            @(negedge RAM_vif.clk);
            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
        end
    endtask
 endclass
endpackage
```

RAM scoreboard file :

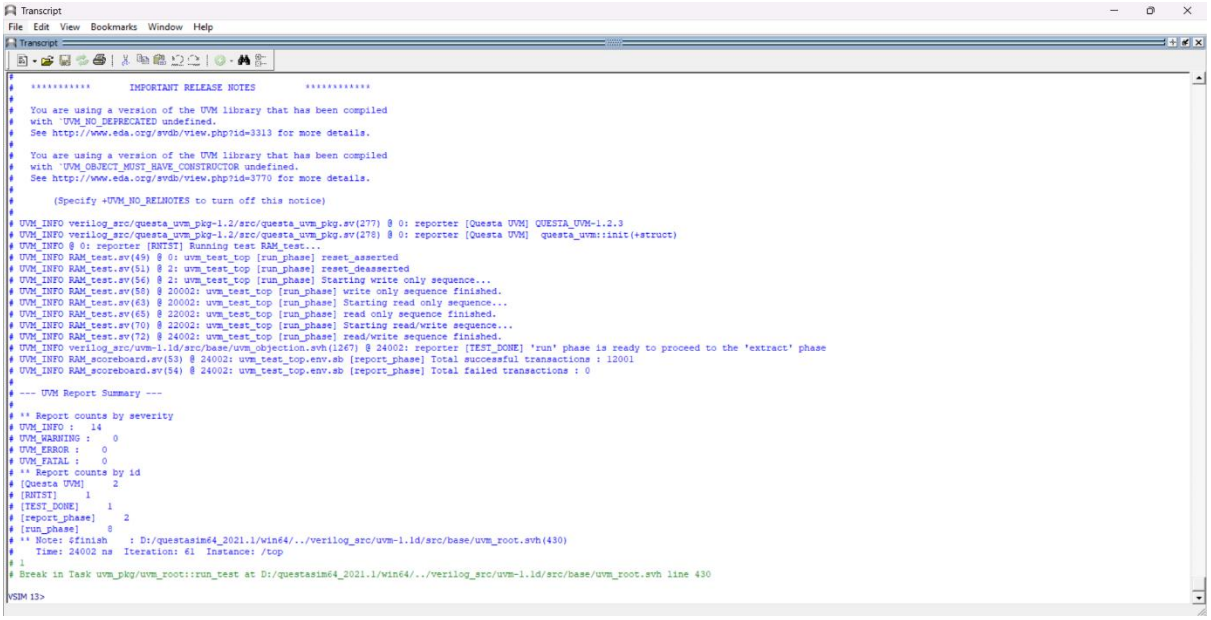
```
package scoreboard_pkg;
    import seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    class RAM_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(RAM_scoreboard)

        uvm_analysis_export #(RAM_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(RAM_seq_item) sb_fifo;

        RAM_seq_item seq_item_sb;

        int error_count = 0;
        int correct_count = 0;

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

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(seq_item_sb);
                if (seq_item_sb.dout !== seq_item_sb.dout_ref || seq_item_sb.tx_valid !==
seq_item_sb.tx_valid_ref) begin
                    `uvm_error("run_phase",
                       $sformatf("%s comparison failed, DUT dout=%0b tx_valid=%0b, REF dout=%0b
tx_valid=%0b",
                           seq_item_sb.convert2string(), seq_item_sb.dout, seq_item_sb.tx_valid,
seq_item_sb.dout_ref, seq_item_sb.tx_valid_ref))
                    error_count++;
                end
                else begin
                    `uvm_info("run_phase",
                       $sformatf("%s correct dout=%0b tx_valid=%0b",
                           seq_item_sb.convert2string(), seq_item_sb.dout, seq_item_sb.tx_valid),
UVM_HIGH)
                    correct_count++;
                end
            end
        endtask

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

RAM coverage file :

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

    class RAM_coverage extends uvm_component;
        `uvm_component_utils(RAM_coverage)

        uvm_analysis_export   #(RAM_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(RAM_seq_item) cov_fifo;
        RAM_seq_item seq_item_cov;

        covergroup cg ;
            //Coverpoint to check transaction ordering for din[9:8]
            cp_din : coverpoint seq_item_cov.din[9:8] {
                bins din_00 = {2'b00};
                bins din_01 = {2'b01} ;
                bins din_10 = {2'b10} ;
                bins din_11 = {2'b11} ;
                bins write_address_data = (2'b00 => 2'b01) ;
                bins read_address_data  = (2'b10 => 2'b11) ;
                bins write_read_data    = (2'b00 => 2'b01 => 2'b10 => 2'b11) ;
            }

            //Coverpoint rx_valid
            cp_rx : coverpoint seq_item_cov.rx_valid {bins rx_valid_1 = {1};}

            //Coverpoint tx_valid
            cp_tx : coverpoint seq_item_cov.tx_valid {bins tx_valid_1 = {1};}

            //----------------cross coverage----------------
            //Between all bins of din[9:8] and rx_valid signal when it is high
            cross_rx_din : cross cp_din,cp_rx {
                bins rx_din_00 = binsof(cp_din.din_00) && binsof(cp_rx.rx_valid_1);
                bins rx_din_01 = binsof(cp_din.din_01) && binsof(cp_rx.rx_valid_1);
                bins rx_din_10 = binsof(cp_din.din_10) && binsof(cp_rx.rx_valid_1);
                bins rx_din_11 = binsof(cp_din.din_11) && binsof(cp_rx.rx_valid_1);
                option.cross_auto_bin_max = 0;
            }

            //Between din[9:8] when it equals read data and tx_valid when it is high
            cross_tx_din : cross cp_din,cp_tx {
                bins tx_din_11 = binsof(cp_din.din_11) && binsof(cp_tx.tx_valid_1);
                option.cross_auto_bin_max = 0;
            }
        endgroup

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

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

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

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                cg.sample();
            end
        endtask: run_phase
    endclass
endpackage
```

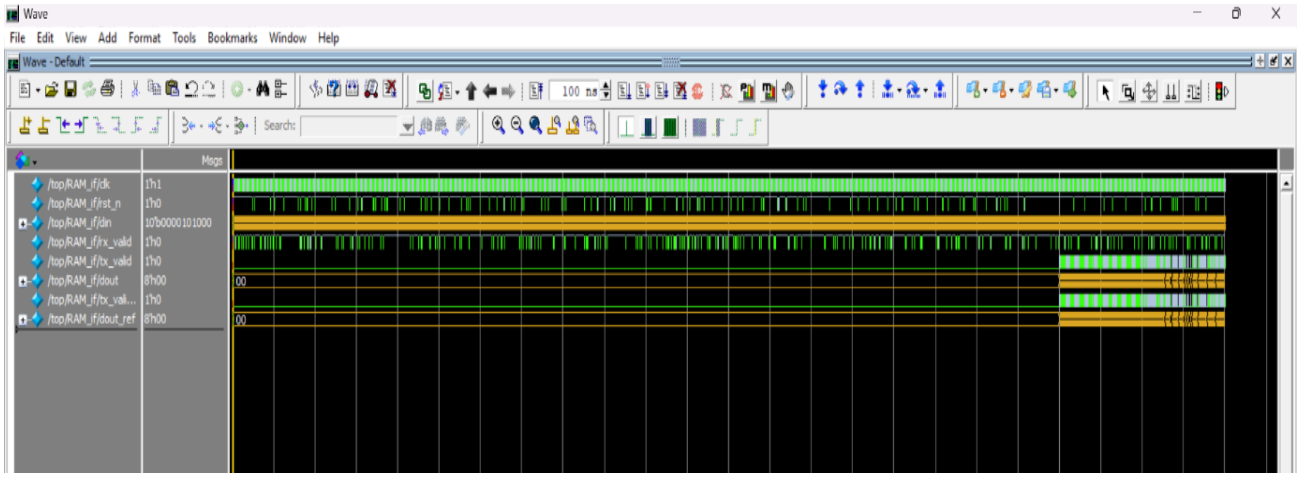
Second: Snippets from Simulation results:

Transcript:

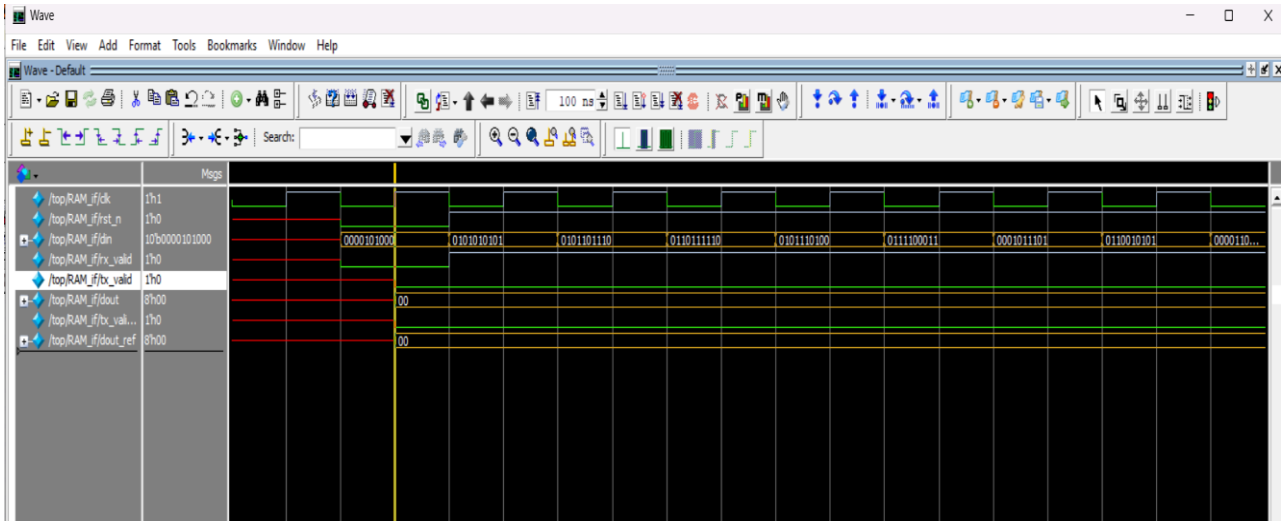


Waveforms:

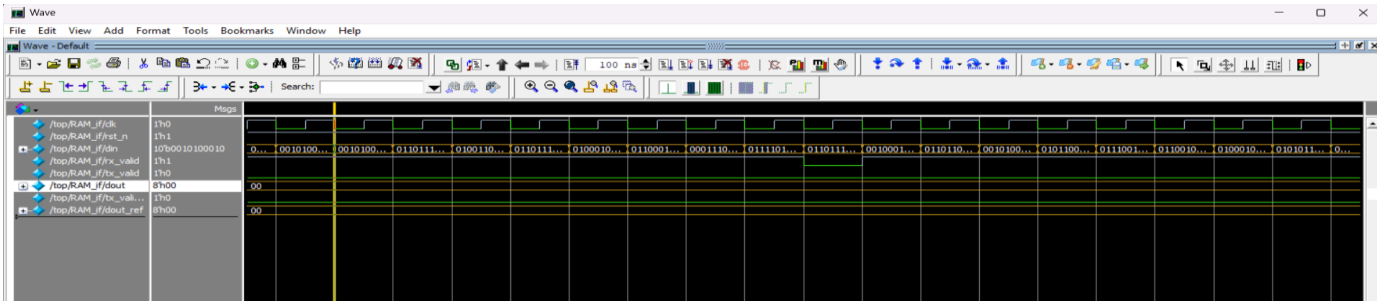
- Full wave:

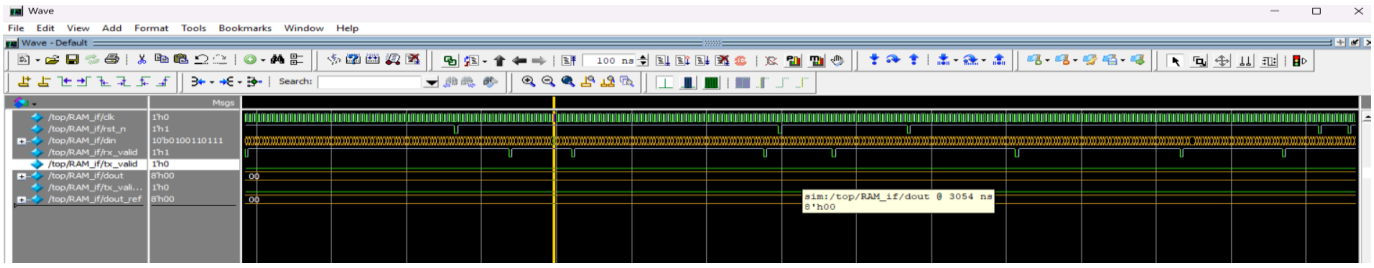


- Reset seq:

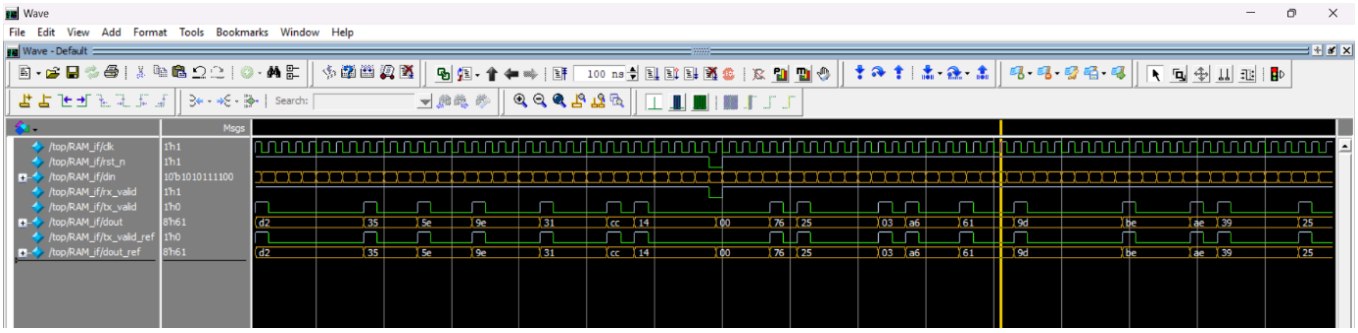


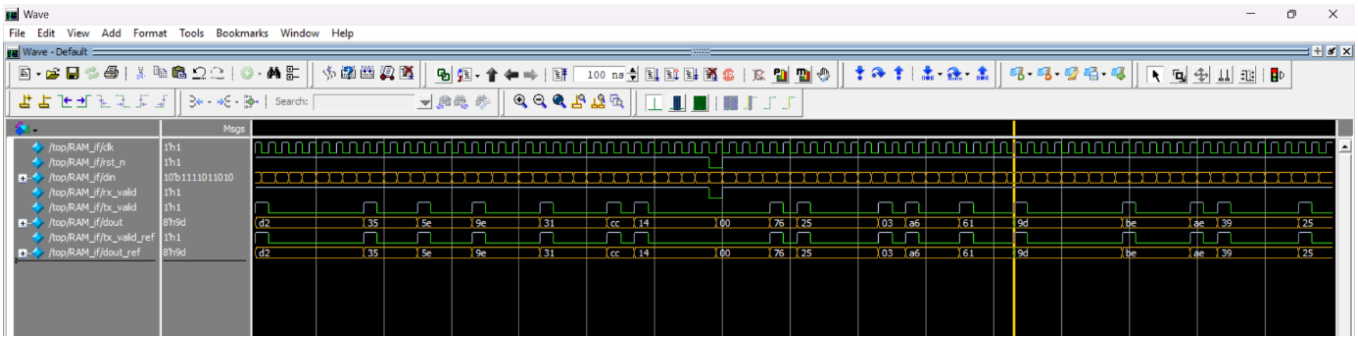
- Write only seq:



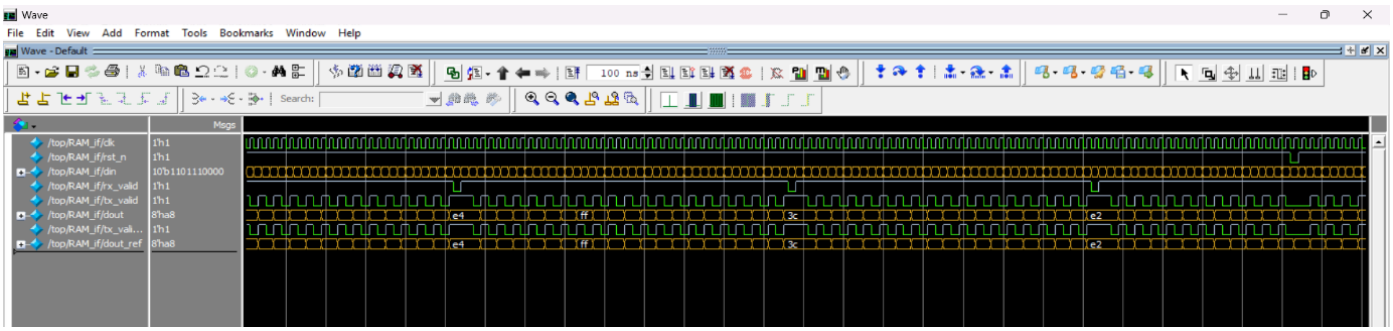


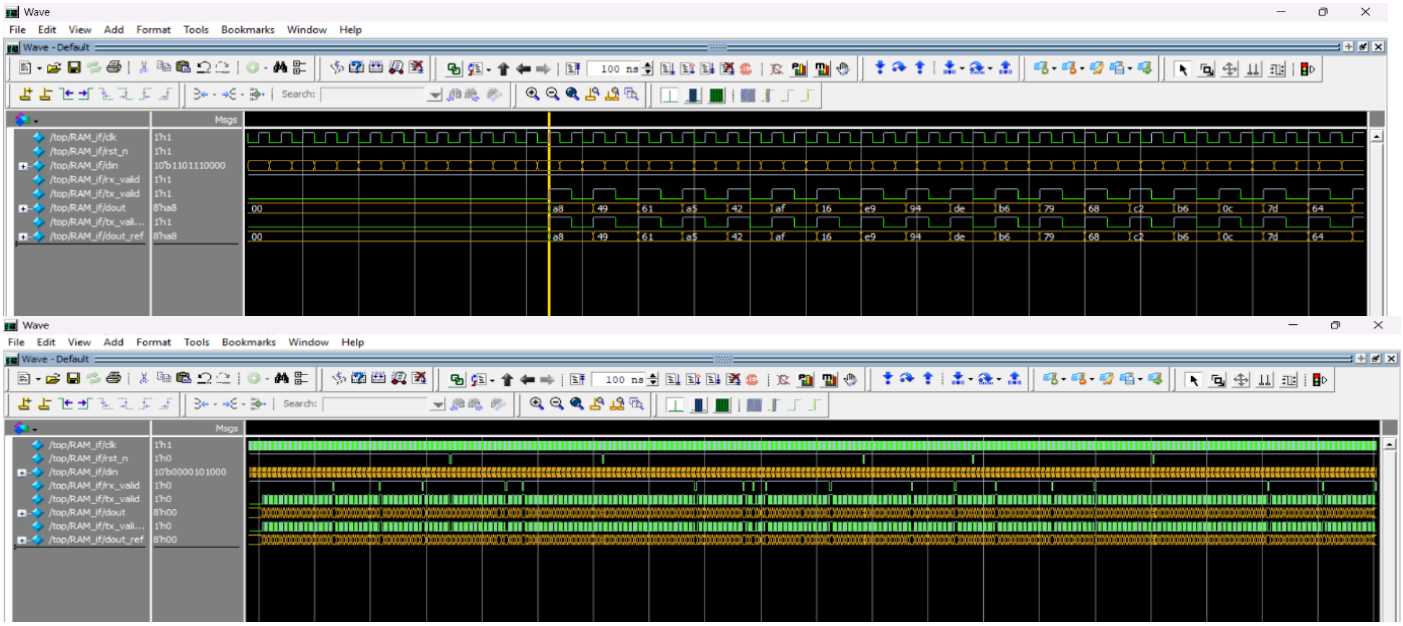
- READ ONLY SEQ:



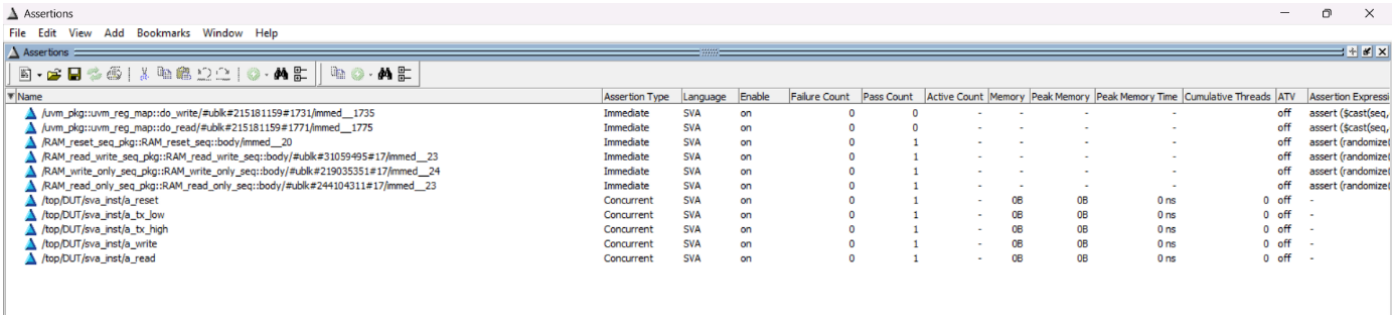


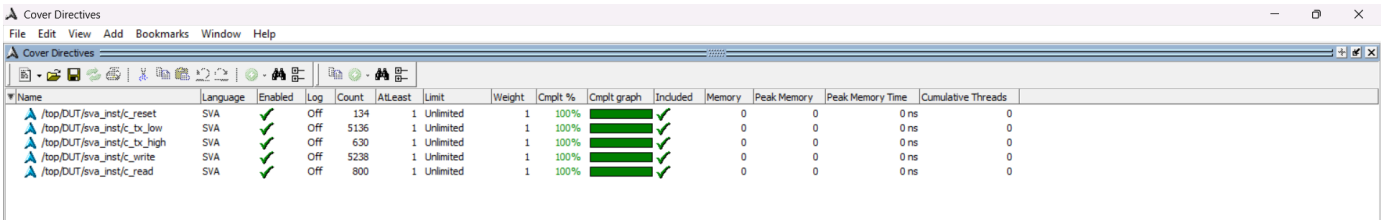
- WRITE-READ SEQ:



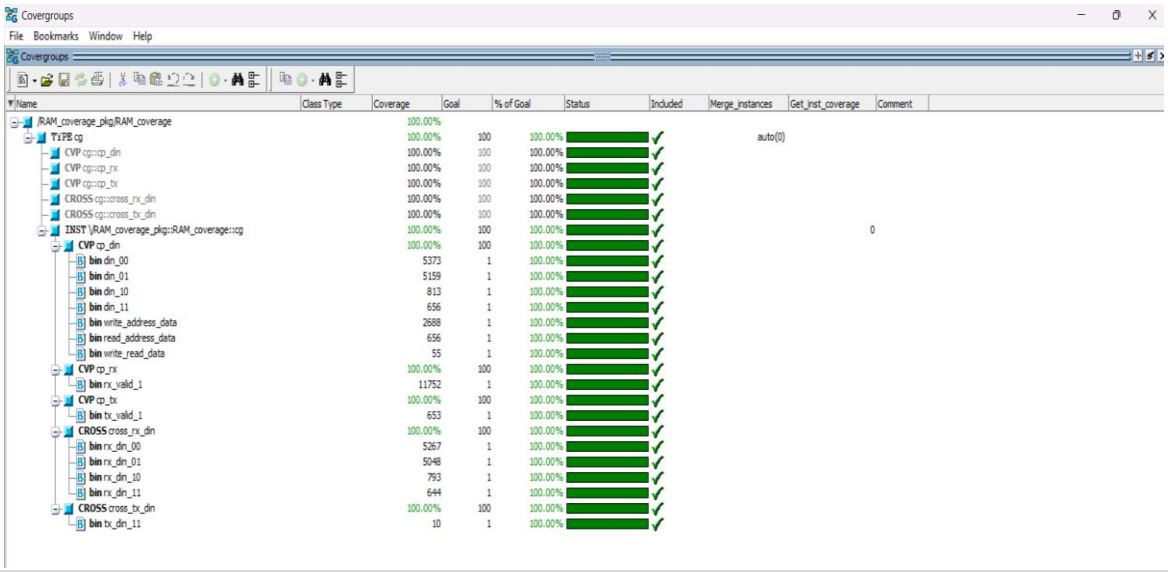


Assertions coverage:



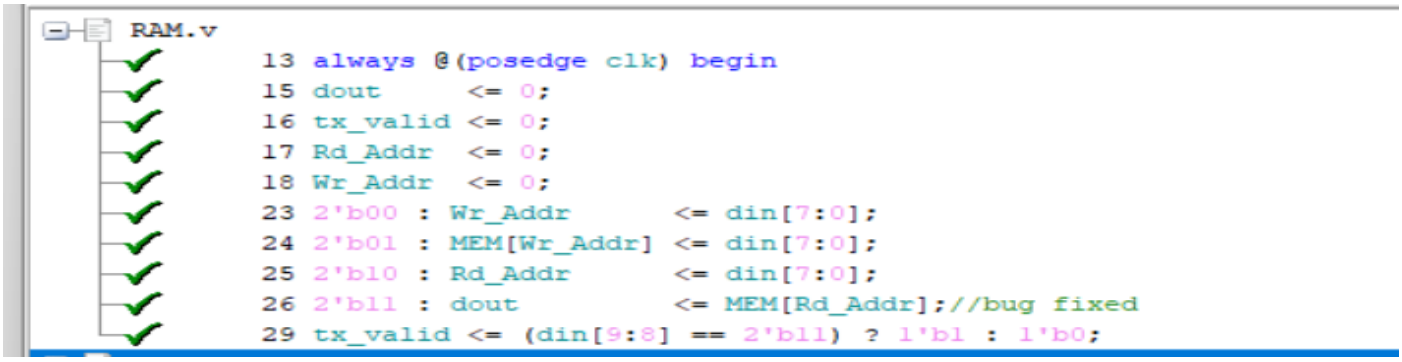


Function coverage:

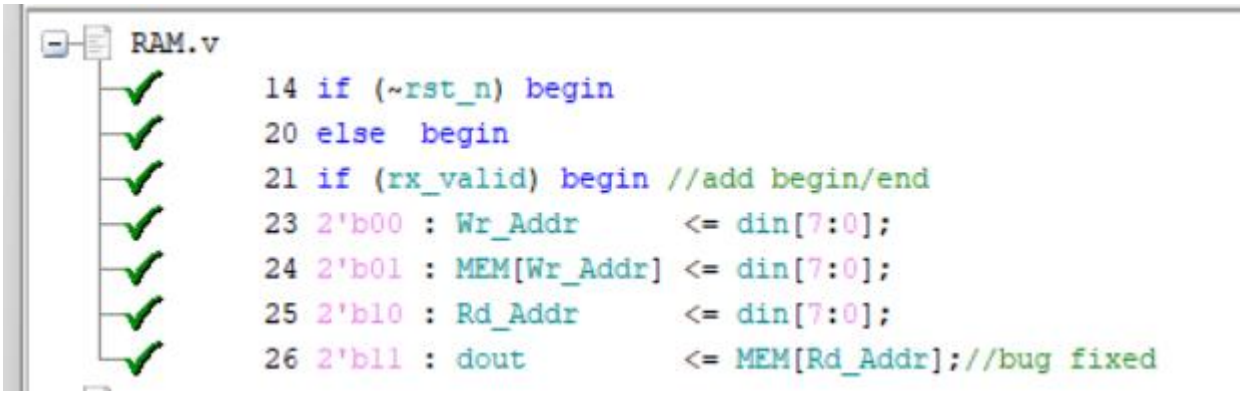


Code coverage:

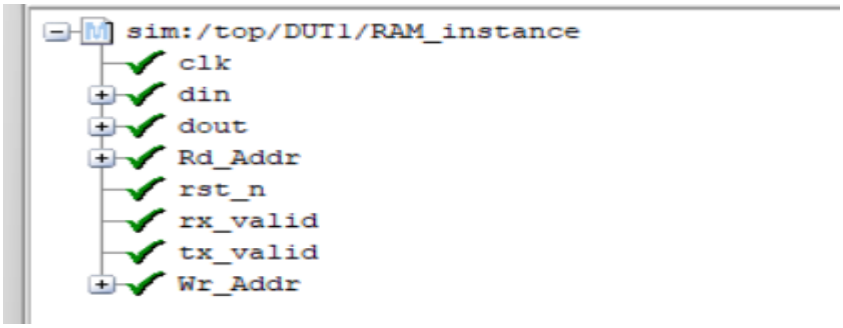
-Statements:



-Branches:



-Toggles:



Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
RAM_0	Reset Behavior: When the active-low reset (rst_n) is asserted, the din[9:8] ,write address (wr_addr), and read address(rd_addr) must be reset to 0. All output signals reset to 0.	Randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the score board to make sure the output is correct and cocurrent assertions to check it too.
RAM_1	For a write-only sequence, every Write Address operation shall always be followed by either Write Address or Write Data operation.	Randomized during simulation due to constrains to make din[9:8] inside {2'b00(write_address), 2'b01(write_data)}	Check write data after write address	Output Checked against refrence model in the score bord and SVA inside SVA file to check internal signals
RAM_2	For a read-only sequence, every Read Address operation shall always be followed by Read Data. After a Read Data operation shall always be followed by Read Address.	Randomized during simulation due to constrains to make din[9:8] change to {2'b11(read_data) after 2'b10(read_address)} and vice versa.	Check read data after read address	Output Checked against refrence model in the score bord and SVA inside SVA file to check internal signals
RAM_3	• Every Write Address operation shall always be followed by either Write Address or Write Data operation. • After a Write Data, the next operation shall be chosen with the following probability distribution: 60% → Read Address & 40% → Write Address • Every Read Address operation shall always be followed by either Read Address or Read Data operation.After a Read Data, the next operation shall be chosen with the following probability distribution: 60% → Write Address & 40% → Read Address	Randomized with constraint that drive the din[9:8] to match required specifications	• 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	Output Checked against refrence model in the score bord and SVA inside SVA file to check internal signals

Wrapper part

First : Changed files in RAM and SLAVE env.

SLAVE configuration object edited :

```
package SLAVE_config_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class SLAVE_config extends uvm_object;
        `uvm_object_utils(SLAVE_config)
        virtual SLAVE_if SLAVE_vif ;
        uvm_active_passive_enum is_active;
        function new(string name = "SLAVE_config");
            super.new(name) ;
        endfunction
    endclass
endpackage
```

RAM configuration object edited :

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

    class RAM_config_obj extends uvm_object;
        `uvm_object_utils(RAM_config_obj)

        virtual RAM_if RAM_config_vif  ;
        uvm_active_passive_enum is_active;
        function new (string name = "RAM_config_obj");
            super.new(name);
        endfunction
    endclass
endpackage
```

SLAVE agent file_edited :

```
package SLAVE_agent_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import SLAVE_sequencer_pkg::*;
    import SLAVE_driver_pkg::*;
    import SLAVE_config_pkg::*;
    import SLAVE_monitor_pkg::*;
    import SLAVE_item_pkg::*;

    class SLAVE_agent extends uvm_agent;
        `uvm_component_utils(SLAVE_agent)
        SLAVE_sequencer sqr ;
        SLAVE_driver drv ;
        SLAVE_monitor mon;
        SLAVE_config SLAVE_cfg;
        uvm_analysis_port #(SLAVE_item) agt_ap;

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

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(SLAVE_config)::get(this, "", "CFG", SLAVE_cfg)) begin
                `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
            end
             if(SLAVE_cfg.is_active== UVM_ACTIVE) begin
                sqr = SLAVE_sequencer::type_id::create("sqr",this);
                drv = SLAVE_driver::type_id::create("drv",this);
             end
            mon = SLAVE_monitor::type_id::create("mon",this);
            agt_ap = new("agt_ap",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
             if(SLAVE_cfg.is_active== UVM_ACTIVE) begin
                drv.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
             end
            mon.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

RAM agent file_edited :

```
package agent_pkg;
    import pkg_cfg::*;
    import seq_item_pkg::*;
    import RAM_sequencer_pkg::*;
    import RAM_monitor_pkg::*;
    import pkg_driver::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    class RAM_agent extends uvm_agent;
        `uvm_component_utils(RAM_agent)

        RAM_driver drv ;
        RAM_monitor mon ;
        RAM_config_obj RAM_cfg ;
        RAM_sequencer sqr  ;
        uvm_analysis_port #(RAM_seq_item) agt_ap ;

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

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(RAM_config_obj)::get(this, "", "CFG", RAM_cfg))
              `uvm_fatal("build_phase", " unable to get configuration object");

            if(RAM_cfg.is_active== UVM_ACTIVE) begin
                sqr = RAM_sequencer::type_id::create("sqr",this);
                drv = RAM_driver::type_id::create("drv",this);
            end
            mon = RAM_monitor::type_id::create("mon",this);
            agt_ap = new("agt_ap",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            if(RAM_cfg.is_active== UVM_ACTIVE) begin
                drv.RAM_vif  = RAM_cfg.RAM_config_vif;
                drv.seq_item_port.connect(sqr.seq_item_export);
            end
                mon.RAM_vif  = RAM_cfg.RAM_config_vif ;
                mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage

```

Second : Wrapper env. Files

Wrapper interface file :

```
interface wrapper_if(clk);
  input logic clk;
  logic MOSI,MISO_ref,SS_n,rst_n;
  bit MISO;
  logic tx_valid;
  logic [7:0] tx_data;
  logic [9:0] rx_data;
  logic rx_valid;
  modport wrp_sva (input  rx_data, rx_valid, clk, MOSI, MISO, SS_n, rst_n, tx_data, tx_valid);
endinterface
```

Wrapper design & assertion file :

```
module wrapper (wrapper_if wrapperif);
  logic  [9:0]  rx_data;
  logic         rx_valid;
  logic         tx_valid;
  logic [7:0]   tx_data;

  RAM   RAM_instance (rx_data,wrapperif.clk,wrapperif.rst_n,rx_valid,
                      tx_data,tx_valid);

  SLAVE SLAVE_instance (wrapperif.MOSI,wrapperif.MISO,wrapperif.SS_n,wrapperif.clk,
                        wrapperif.rst_n,rx_data,rx_valid,tx_data,tx_valid);

  assign wrapperif.rx_data  = rx_data;
  assign wrapperif.rx_valid = rx_valid;
  assign wrapperif.tx_data  = tx_data;
  assign wrapperif.tx_valid = tx_valid;

// assertions section
// assertion 1 : An assertion ensures that whenever reset is asserted, the output (MISO) is inactive.
  property p_reset_outputs_inactive;
  @(posedge wrapperif.clk) !wrapperif.rst_n |=> (wrapperif.MISO == 0 &&
                                                 wrapperif.rx_valid == 0 &&
                                                 wrapperif.rx_data  == 0);
  endproperty

  assert property (p_reset_outputs_inactive);
  cover property (p_reset_outputs_inactive);
// assertion 2 : An assertion to make sure that the MISO remains with a stable value eventually as long
//               as it is not a read data operation
  property stable_miso;
    @(wrapperif.clk) disable iff (!wrapperif.rst_n) (wrapperif.rx_data[9:8] != 2'b11) |=> $stable(wrapperif.MISO);
  endproperty

  assert property (stable_miso);
  cover property (stable_miso);

endmodule
```

Wrapper golden model :

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

localparam S_IDLE      = 3'b000;
localparam S_WRITE     = 3'b001;
localparam S_CMDCHK    = 3'b010;
localparam S_RDADDR    = 3'b011;
localparam S_RDDATA    = 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;
output bit       MISO;

// renamed internal signals
reg [3:0] bit_counter;
reg       addr_received;

reg [2:0] state_curr, state_next;

// State register
always @(posedge clk) begin
    if (~rst_n)
        state_curr <= S_IDLE;
    else
        state_curr <= state_next;
end

// Next state logic
always @(*) begin
    case (state_curr)
        S_IDLE: begin
            if (SS_n)
                state_next = S_IDLE;
            else
                state_next = S_CMDCHK;
        end

        S_CMDCHK: begin
            if (SS_n)
                state_next = S_IDLE;
            else begin
                if (~MOSI)
                    state_next = S_WRITE;
                else begin
                    if (addr_received)
                        state_next = S_RDDATA;
                    else
                        state_next = S_RDADDR;
                end
            end
        end

        S_WRITE: begin
            if (SS_n)
                state_next = S_IDLE;
            else
                state_next = S_WRITE;
        end

        S_RDADDR: begin
            if (SS_n)
                state_next = S_IDLE;
            else
                state_next = S_RDADDR;
        end

        S_RDDATA: begin
            if (SS_n)
                state_next = S_IDLE;
            else
                state_next = S_RDDATA;
        end
    endcase
end
```

```
// Output and data handling
always @(posedge clk) begin
    if (~rst_n) begin
        rx_data       <= 0;
        rx_valid      <= 0;
        addr_received <= 0;
        MISO          <= 0;
        bit_counter   <= 0;
    end
    else begin
        case (state_curr)
            S_IDLE: begin
                rx_valid <= 0;
            end

            S_CMDCHK: begin
                bit_counter <= 10;
            end

            S_WRITE: begin
                if (bit_counter > 0) begin
                    rx_data[bit_counter-1] <= MOSI;
                    bit_counter <= bit_counter - 1;
                end
                else begin
                    rx_valid <= 1;
                end
            end

            S_RDADDR: begin
                if (bit_counter > 0) begin
                    rx_data[bit_counter-1] <= MOSI;
                    bit_counter <= bit_counter - 1;
                end
                else begin
                    rx_valid      <= 1;
                    addr_received <= 1;
                end
            end

            S_RDDATA: begin
                if (tx_valid) begin
                    rx_valid <= 0;
                    if (bit_counter > 0) begin
                        MISO <= tx_data[bit_counter-1];
                        bit_counter <= bit_counter - 1;
                    end
                    else begin
                        addr_received <= 0;
                    end
                end
                else begin
                    if (bit_counter > 0) begin
                        rx_data[bit_counter-1] <= MOSI;
                        bit_counter <= bit_counter - 1;
                        rx_valid <= 0;
                    end
                    else begin
                        rx_valid <= 1;
                        bit_counter <= 9;
                    end
                end
            end
        endcase
    end
end

endmodule
```

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

    parameter MEM_DEPTH = 256;
    parameter ADDR_SIZE = 8;

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

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

    reg [ADDR_SIZE-1 : 0] addr_rd, addr_wr;
    reg [7:0] mem [MEM_DEPTH-1 : 0];

    integer i;
    always @(posedge clk ) begin
        if (!rst_n) begin
            dout     <= 0 ;
            tx_valid <= 0 ;
            addr_rd  <= 0 ;
            addr_wr  <= 0 ;
        end
        else begin
            if (rx_valid) begin
                case (din[9:8])
                    2'b00 : begin
                        addr_wr       <= din[7:0] ;
                        tx_valid  <= 1'b0 ;
                    end
                    2'b01 : begin
                        mem[addr_wr]  <= din[7:0] ;
                        tx_valid <= 1'b0 ;
                    end
                    2'b10 : begin
                        addr_rd       <= din[7:0] ;
                        tx_valid <= 1'b0 ;
                    end
                    2'b11 : begin
                        dout     <= mem[addr_rd] ;
                        tx_valid <= 1'b1 ;
                    end
                    default: dout <= 0;
                endcase
            end
        end
    end
endmodule


module golden_model_SPI_wrapper (
    input  logic MOSI_ref,
    output  logic MISO_ref,
    input  logic SS_n_ref,
    input  logic clk,
    input logic rst_n_ref
);

    // Internal signals connecting RAM and SLAVE
    logic [9:0] rx_data_din_ref;
    logic       rx_valid_ref;
    logic [7:0] tx_data_dout_ref;
    logic       tx_valid_ref;

    // Instantiate the golden model RAM and SLAVE (with identical logic)
    golden_model_RAM RAM_ref (
        .din(rx_data_din_ref),
        .clk(clk),
        .rst_n(rst_n_ref),
        .rx_valid(rx_valid_ref),
        .dout(tx_data_dout_ref),
        .tx_valid(tx_valid_ref)
    );

    golden_model_SLAVE SLAVE_ref (
        .MOSI(MOSI_ref),
        .MISO(MISO_ref),
        .SS_n(SS_n_ref),
        .clk(clk),
        .rst_n(rst_n_ref),
        .rx_data(rx_data_din_ref),
        .rx_valid(rx_valid_ref),
        .tx_data(tx_data_dout_ref),
        .tx_valid(tx_valid_ref)
);

endmodule
```

Wrapper top file :

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

module top();
  bit clk;

  initial begin
    forever
      #1 clk =~clk ;
  end

  wrapper_if wrapperif (clk);
  SLAVE_if slaveif (clk);
  RAM_if ramif (clk);

  // Instantiate the wrapper (which contains both RAM and SLAVE internally)
  wrapper DUT1 (wrapperif);

  // Golden model for wrapper
  golden_model_SPI_wrapper wrapper_ins (
      .MOSI_ref(wrapperif.MOSI),
      .MISO_ref(wrapperif.MISO_ref),
      .SS_n_ref(wrapperif.SS_n),
      .clk(wrapperif.clk),
      .rst_n_ref(wrapperif.rst_n)
  );

    bind DUT1.RAM_instance RAM_sva RAM_assertion (
      .clk(wrapperif.clk),
      .rst_n(wrapperif.rst_n),
      .tx_valid(wrapperif.tx_valid),
      .dout(wrapperif.tx_data),
      .din(wrapperif.rx_data),
      .rx_valid(wrapperif.rx_valid)
  );

  // Connect SLAVE interface to wrapper's internal SLAVE signals for monitoring
  assign slaveif.rst_n = wrapperif.rst_n;
  assign slaveif.SS_n = wrapperif.SS_n;
  assign slaveif.MOSI = wrapperif.MOSI;
  assign slaveif.MISO = wrapperif.MISO;
  assign slaveif.rx_data = DUT1.rx_data;
  assign slaveif.rx_valid = DUT1.rx_valid;
  assign slaveif.tx_data = DUT1.tx_data;
  assign slaveif.tx_valid = DUT1.tx_valid;

  // Connect RAM interface to wrapper's internal RAM signals for monitoring
  assign ramif.rst_n = wrapperif.rst_n;
  assign ramif.din = DUT1.rx_data;
  assign ramif.rx_valid = DUT1.rx_valid;
  assign ramif.dout = DUT1.tx_data;
  assign ramif.tx_valid = DUT1.tx_valid;

  // Connect RAM golden interface to wrapper's internal RAM signals for monitoring
  assign ramif.dout_ref = DUT1.tx_data;
  assign ramif.tx_valid_ref = DUT1.tx_valid;

  initial begin
    uvm_config_db #(virtual wrapper_if)::set(null, "uvm_test_top" , "WRAPPER_IF" ,wrapperif);
    uvm_config_db #(virtual SLAVE_if)::set(null, "uvm_test_top" , "SLAVE_IF" ,slaveif);
    uvm_config_db #(virtual RAM_if)::set(null, "uvm_test_top" , "RAM_IF" ,ramif);
    run_test("wrapper_test");
  end

endmodule
```

Wrapper test file :

```
package wrapper_test_pkg;
  import SLAVE_env_pkg::*;
  import SLAVE_config_pkg::*;
  import RAM_env_pkg::*;
  import pkg_cfg::*;
  import wrapper_env_pkg::*;
  import wrapper_config_pkg::*;
  import wrapper_reset_seq_pkg::*;
  import uvm_pkg::*;
  import wrapper_read_only_seq_pkg::*;
  import wrapper_write_only_seq_pkg::*;
  import wrapper_read_write_seq_pkg::*;
  `include "uvm_macros.svh"

  class wrapper_test extends uvm_test;
    `uvm_component_utils(wrapper_test)
    wrapper_env env ;
    wrapper_config wrapper_cfg ;
    wrapper_reset_seq rst_seq;
    RAM_env env_ram ;
    SLAVE_env env_slave ;
    SLAVE_config slave_cfg ;
    RAM_config_obj ram_cfg ;
    wrapper_read_only_seq ro_seq;
    wrapper_write_only_seq wo_seq;
    wrapper_read_write_seq rw_seq;

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

    function void build_phase (uvm_phase phase);
      super.build_phase(phase) ;
      env = wrapper_env::type_id::create("env",this);
      env_ram = RAM_env::type_id::create("env_ram",this);
      env_slave = SLAVE_env::type_id::create("env_slave",this);
      wrapper_cfg = wrapper_config::type_id::create("wrapper_cfg");
      ram_cfg = RAM_config_obj::type_id::create("ram_cfg");
      slave_cfg = SLAVE_config::type_id::create("slave_cfg");
      rst_seq = wrapper_reset_seq::type_id::create("rst_seq");
      ro_seq = wrapper_read_only_seq::type_id::create("ro_seq");
      wo_seq = wrapper_write_only_seq::type_id::create("wo_seq");
      rw_seq = wrapper_read_write_seq::type_id::create("rw_seq");

      if (!uvm_config_db #(virtual wrapper_if)::get(this, "" , "WRAPPER_IF" , wrapper_cfg.wrapper_vif))
        `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface");

      if(!uvm_config_db #(virtual SLAVE_if)::get(this, "" , "SLAVE_IF" , slave_cfg.SLAVE_vif))
        `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface of SLAVE");

      if(!uvm_config_db #(virtual RAM_if)::get(this, "" , "RAM_IF" , ram_cfg.RAM_config_vif))
        `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface of RAM");

      ram_cfg.is_active = UVM_PASSIVE;
      slave_cfg.is_active = UVM_PASSIVE;
      wrapper_cfg.is_active = UVM_ACTIVE;

      // Set config objects with correct keys that agents expect
      uvm_config_db #(RAM_config_obj)::set(this , "*" , "CFG" , ram_cfg);
      uvm_config_db #(SLAVE_config)::set(this , "*" , "CFG" , slave_cfg);
      uvm_config_db #(wrapper_config)::set(this , "*" , "CFG" , wrapper_cfg);

    endfunction

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

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

      `uvm_info("run_phase", "write only sequence started" , UVM_MEDIUM)
      wo_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "write only sequence finished" , UVM_MEDIUM)

      `uvm_info("run_phase", "read only sequence started" , UVM_MEDIUM)
      ro_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "read only sequence finished" , UVM_MEDIUM)

      `uvm_info("run_phase", "read/write sequence started" , UVM_MEDIUM)
      rw_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "read/write sequence finished" , UVM_MEDIUM)

      phase.drop_objection(this);
    endtask
  endclass

endpackage
```

Wrapper seq_item file :

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

       // Randomized inputs
        rand logic rst_n;
        rand logic SS_n;
        rand bit [10:0] MOSI_array;

        // Non-randomized inputs (driven from MOSI_array or constraints)
        logic MOSI;

        // DUT outputs (monitored)
        bit MISO, MISO_ref;
        bit [2:0] prev_op = 3'b000;

        // State tracking variables
        logic [4:0] count;              // Counter for SS_n timing (0-22)
        logic [3:0] bit_index;          // Track which bit of MOSI_array to send (0-10)
        logic prev_SS_n;                // Previous value of SS_n
        bit [10:0] stored_MOSI_array;   // Store the randomized array
        bit [2:0] prev_cmd;             // store the previous command

        function new(string name = "wrapper_seq_item");
            super.new(name);
            prev_SS_n = 1'b1;
            count = 0;
            bit_index = 0;
            stored_MOSI_array = 11'b000_0000_0000;
        endfunction

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

        function string convert2string_stimuls();
            return $sformatf("rst_n=%b, SS_n=%b, MOSI=%b, MISO=%b, MISO_ref=%b",
                              rst_n, SS_n, MOSI, MISO, MISO_ref);
        endfunction

        // constraint section
        // constraint 1 : reset constraint
        constraint reset_cons {
            rst_n dist {1 := 99, 0 := 1};
        }

        // Constraint 2: SS_n timing based on command type
        // For WRITE (000) and READ_ADD (001): SS_n high every 13 cycles
        // For READ_DATA (110, 111): SS_n high every 23 cycles
        constraint SS_n_timing_c {
            if (MOSI_array[10:8] inside {3'b000, 3'b001 ,3'b110}) {
                // Write or Read Address: 13 cycle period (12 low, 1 high)
                SS_n == (count == 12);
            }
            else if (MOSI_array[10:8] inside { 3'b111}) {
                // Read Data: 23 cycle period (22 low, 1 high)
                SS_n == (count == 22);
            }
            else {
                // Default/initial case
                SS_n == (count == 12);
            }
        }

        constraint addr_change {
            MOSI_array[7:0] dist { [8'h00:8'hFF] := 100 };
        }

        // For a write-only sequence, every Write Address operation shall always be followed by either
        //Write Address or Write Data operation.
        constraint c_write_only {
            if (prev_cmd == 3'b000) {
                MOSI_array[10:8] == 3'b001;
            }
            else if (prev_cmd == 3'b001) {
                MOSI_array[10:8] == 3'b000 ;
            }
            else {
                MOSI_array[10:8] inside {3'b000 , 3'b001};
            }
        };

        //  For a read-only sequence, only Read Address or Read Data operations are allowed.
        constraint c_read_only {
            if (prev_cmd == 3'b110) {
                MOSI_array[10:8] == 3'b111;
            }
            else if (prev_cmd == 3'b111) {
                MOSI_array[10:8] == 3'b110;
            }
            else {
                MOSI_array[10:8] inside {3'b110 , 3'b111 };
            }
        };
```

```
        //  For a read-only sequence, only Read Address or Read Data operations are allowed.
         constraint c_read_write {
             if (prev_cmd == 3'b000){
                 MOSI_array[10:8] inside {3'b000 , 3'b001};
             }
             else if (prev_cmd == 3'b001) {
                 MOSI_array[10:8] dist {3'b000:=40 , 3'b110:=60};
             }
             else if (prev_cmd == 3'b110) {
                 MOSI_array[10:8] == 3'b111;
             }
             else if (prev_cmd == 3'b111) {
                 MOSI_array[10:8] dist {3'b000:=60 , 3'b110:=40};
             }
             else {
                 MOSI_array[10:8] inside {3'b000 , 3'b001 , 3'b110 , 3'b111};
             }
         };

         // Post-randomize to handle bit-by-bit MOSI assignment and state updates
         function void post_randomize();
             logic is_falling_edge;

             is_falling_edge = (prev_SS_n && !SS_n);

             if (!rst_n) begin
                 count        = 0;
                 bit_index    = 0;
                 prev_SS_n    = 1'b1;
                 stored_MOSI_array = 11'b0;
                 MOSI_array[10:8]  = 3'b000;
                 prev_cmd = 0 ;
                 MOSI         = 1'b0;
                 `uvm_info("RESET", "Reset occurred: internal state reinitialized", UVM_HIGH)
                 return; // Skip normal transaction logic this cycle
             end

             // Handle SS_n falling edge - start new transaction
             if (is_falling_edge) begin
                 stored_MOSI_array = MOSI_array;
                 prev_cmd = MOSI_array[10:8];  // Store previous before updating
                 bit_index = 0;
                 count = 0;
                 MOSI = stored_MOSI_array[10];  // Start with MSB

                 `uvm_info("WRAPPER_ITEM", $sformatf(
                     "New Transaction: cmd=%b, MOSI_array=%b, count=
",                   prev_cmd, stored_MOSI_array, count
                 ), UVM_HIGH)
             end
             // Handle SS_n active (low) - continue shifting bits
             else if (!SS_n) begin
                 if (bit_index < 11) begin
                     MOSI = stored_MOSI_array[10 - bit_index];
                     bit_index++;
                 end
                 else begin
                     MOSI = 1'b0;  // Default after all bits sent
                 end
             end
             // Handle SS_n inactive (high) - idle
             else begin
                 MOSI = 1'b0;
                 bit_index = 0;
             end

             // Update counter for SS_n timing
             if (MOSI_array[10:8] inside {3'b000, 3'b001 ,3'b110}) begin
                 // 13 cycle period
                 if (count >= 12) begin
                     count = 0;
                 end
                 else count++;
             end
             else if (MOSI_array[10:8] inside { 3'b111}) begin
                 // 23 cycle period
                 if (count >= 22) begin
                     count = 0;
                 end
                 else count++;
             end
             else begin
                 // Default: 13 cycle period
                 if (count >= 12) begin
                     count = 0;
                 end
                 else count++;
             end
             // Update prev_SS_n for next cycle
             prev_SS_n = SS_n;
         endfunction
     endclass
endpackage
```

Wrapper configuration object:

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

Wrapper write_only seq:

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

    class wrapper_write_only_seq extends uvm_sequence #(wrapper_seq_item) ;
    `uvm_object_utils(wrapper_write_only_seq)

    wrapper_seq_item seq_item ;

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

    task body();
    seq_item = wrapper_seq_item::type_id::create("seq_item");
        repeat(90000) begin
            start_item(seq_item);
            seq_item.c_read_only.constraint_mode(0);
            seq_item.c_read_write.constraint_mode(0);
            seq_item.c_write_only.constraint_mode(1);
            assert (seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
 endclass
endpackage
```

Wrapper raed_only seq:

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

    class wrapper_read_only_seq extends uvm_sequence #(wrapper_seq_item);
        `uvm_object_utils(wrapper_read_only_seq)
        wrapper_seq_item seq_item;

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

        task body();
            seq_item = wrapper_seq_item::type_id::create("seq_item");
            repeat(10000) begin
                start_item(seq_item);
                seq_item.c_write_only.constraint_mode(0);
                seq_item.c_read_write.constraint_mode(0);
                seq_item.c_read_only.constraint_mode(1);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
endpackage
```

Wrapper raed_write seq:

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

    class wrapper_read_write_seq extends uvm_sequence #(wrapper_seq_item) ;
        `uvm_object_utils(wrapper_read_write_seq)
        wrapper_seq_item seq_item ;

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

        task body();
            seq_item = wrapper_seq_item::type_id::create("seq_item");
            repeat(10000) begin
                start_item(seq_item);
                seq_item.c_read_only.constraint_mode(0);
                seq_item.c_write_only.constraint_mode(0);
                seq_item.c_read_write.constraint_mode(1);
                assert (seq_item.randomize());
                finish_item(seq_item);
            end
        endtask
    endclass
endpackage
```

Wrapper sequencer file :

```
package wrapper_sequencer_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_seq_item_pkg::*;
    class wrapper_sequencer extends uvm_sequencer #(wrapper_seq_item);
        `uvm_component_utils(wrapper_sequencer)

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

    endclass
endpackage
```

Wrapper env file :

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

class wrapper_env extends uvm_env;

  `uvm_component_utils(wrapper_env)

  wrapper_agent agt ;
  wrapper_coverage cov ;
  wrapper_scoreboard sb ;

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

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

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

endclass
endpackage
```

Wrapper agent file :

```
package wrapper_agent_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import wrapper_sequencer_pkg::*;
    import wrapper_driver_pkg::*;
    import wrapper_config_pkg::*;
    import wrapper_monitor_pkg::*;
    import wrapper_seq_item_pkg::*;

    class wrapper_agent extends uvm_agent;
        `uvm_component_utils(wrapper_agent)
        wrapper_sequencer sqr ;
        wrapper_driver drv ;
        wrapper_monitor mon;
        wrapper_config wrapper_cfg;
        uvm_analysis_port #(wrapper_seq_item) agt_ap;

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

        function void build_phase (uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(wrapper_config)::get(this, "", "CFG", wrapper_cfg)) begin
                `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
            end
            sqr = wrapper_sequencer::type_id::create("sqr",this);
            drv = wrapper_driver::type_id::create("drv",this);
            mon = wrapper_monitor::type_id::create("mon",this);
            agt_ap = new("agt_ap",this);
        endfunction

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


    endclass
endpackage
```

Wrapper driver file :

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

    class wrapper_driver extends uvm_driver #(wrapper_seq_item);
        `uvm_component_utils(wrapper_driver)
        virtual wrapper_if wrapper_vif;
        wrapper_seq_item stim_seq_item;

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item = wrapper_seq_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                wrapper_vif.rst_n=stim_seq_item.rst_n;
                wrapper_vif.SS_n=stim_seq_item.SS_n;
                wrapper_vif.MOSI=stim_seq_item.MOSI;
              @(negedge wrapper_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string_stimuls(), UVM_HIGH);
            end
        endtask
    endclass
endpackage
```

Wrapper monitor file :

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

    class wrapper_monitor extends uvm_monitor;
        `uvm_component_utils(wrapper_monitor)
        virtual wrapper_if wrapper_vif;
        wrapper_seq_item rsp_seq_item;
        uvm_analysis_port #(wrapper_seq_item) mon_ap;

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

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

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                rsp_seq_item = wrapper_seq_item::type_id::create("rsp_seq_item");
                rsp_seq_item.rst_n=wrapper_vif.rst_n;
                rsp_seq_item.SS_n=wrapper_vif.SS_n;
                rsp_seq_item.MOSI=wrapper_vif.MOSI;
                rsp_seq_item.MISO=wrapper_vif.MISO;
                rsp_seq_item.MISO_ref=wrapper_vif.MISO_ref;
                @(negedge wrapper_vif.clk);
                mon_ap.write(rsp_seq_item);
            end
        endtask
    endclass
endpackage
```

Wrapper scoreboard:

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

    class wrapper_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(wrapper_scoreboard)
        uvm_analysis_export #(wrapper_seq_item) sb_export;
        uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;
        wrapper_seq_item seq_item_sb;

        int error_count = 0;
        int correct_count = 0;

        logic MISO_ref;

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

       function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("SCOREBOARD", $sformatf("Simulation Summary:
                correct_count=
, error_endfunction
",correct_count, error_count), UVM_NONE)
        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_ref) begin
                    `uvm_error("run_phase" , "comaprison fail");
                    error_count++;
                end
                else correct_count++;
            end
        endtask
    endclass

endpackage
```

Wrapper coverage file :

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

    class wrapper_coverage extends uvm_component;
        `uvm_component_utils(wrapper_coverage)
        uvm_analysis_export #(wrapper_seq_item) cov_export;
        uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
        wrapper_seq_item seq_item_cov;

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

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

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

        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
            end
        endtask
    endclass
endpackage
```

Do file :

```
vlib work
vlog -f src_files.list +define+SIM +cover +covercells +coveropt
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover

add wave /top/wrapperif/*
add wave -r /top/DUT1/SLAVE_instance/cs
add wave -r /top/DUT1/SLAVE_instance/ns
add wave -r /top/DUT1/RAM_instance/MEM
add wave /top/DUT1/assert__p_reset_outputs_inactive
add wave /top/DUT1/assert__stable_miso


run -all
```

Src file :


1 WRAPPER_if.sv
2 SLAVE_if.sv
3 RAM_if.sv
4 RAM.v
5 SPI_slave.sv
6 WRAPPER.sv
7 RAM_golden.sv
8 WRAPPER_golden.sv
9 RAM_assertions.sv
10 WRAPPER_config.sv
11 SLAVE_config.sv
12 RAM_config_obj.sv
13 WRAPPER_seq_item.sv
14 SLAVE_seq_item.sv
15 RAM_seq_item.sv
16 SLAVE_seq.sv
17 WRAPPER_reset_seq.sv
18 WRAPPER_read_only_seq.sv
19 WRAPPER_write_only_seq.sv
20 WRAPPER_read_write_seq.sv
21 SLAVE_reset_seq.sv
22 RAM_reset_seq.sv
23 RAM_read_only_seq.sv
24 RAM_write_only_seq.sv
25 RAM_read_write_seq.sv
26 RAM_sequencer.sv
27 SLAVE_sequencer.sv
28 WRAPPER_sequencer.sv
29 RAM_driver.sv
30 SLAVE_driver.sv
31 WRAPPER_driver.sv
32 RAM_monitor.sv
33 SLAVE_monitor.sv
34 WRAPPER_monitor.sv
35 RAM_agent.sv
36 SLAVE_agent.sv
37 WRAPPER_agent.sv
38 RAM_scoreboard.sv
39 SLAVE_scoreboard.sv
40 WRAPPER_scoreboard.sv
41 RAM_coverage.sv
42 SLAVE_coverage.sv
43 WRAPPER_coverage.sv
44 RAM_env.sv
45 SLAVE_env.sv
46 WRAPPER_env.sv
47 RAM_test.sv
48 SLAVE_test.sv
49 WRAPPER_test.sv
50 top.sv

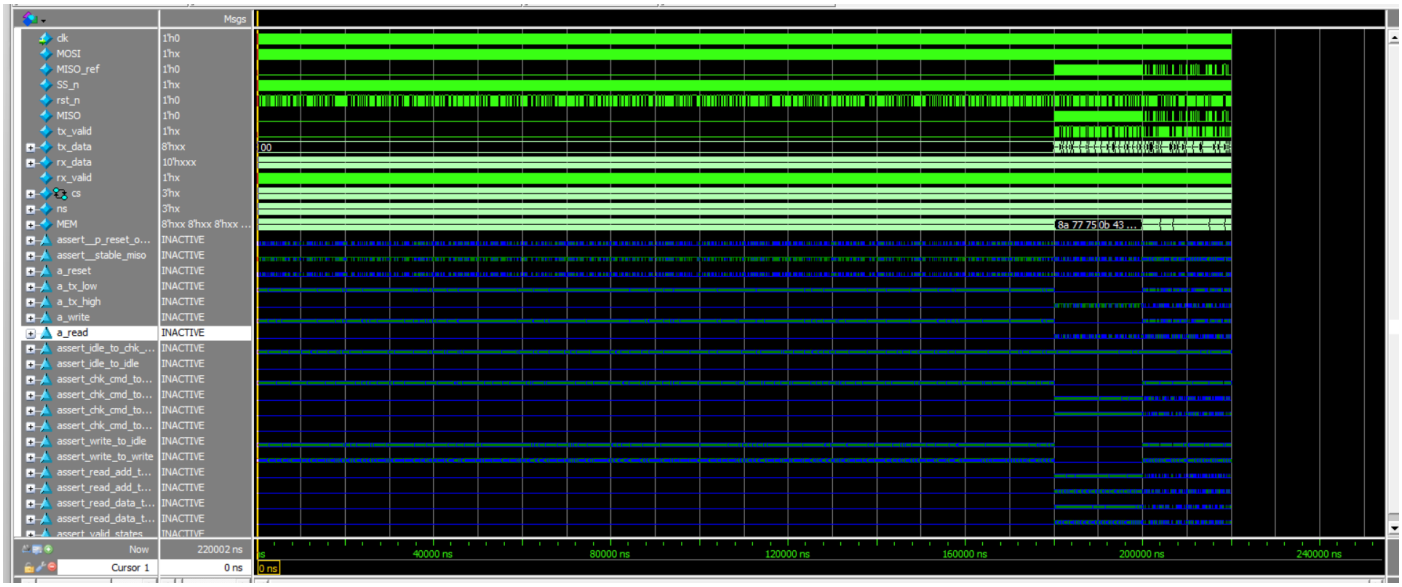
Third:Snippets from Simulation results:

Transcript:

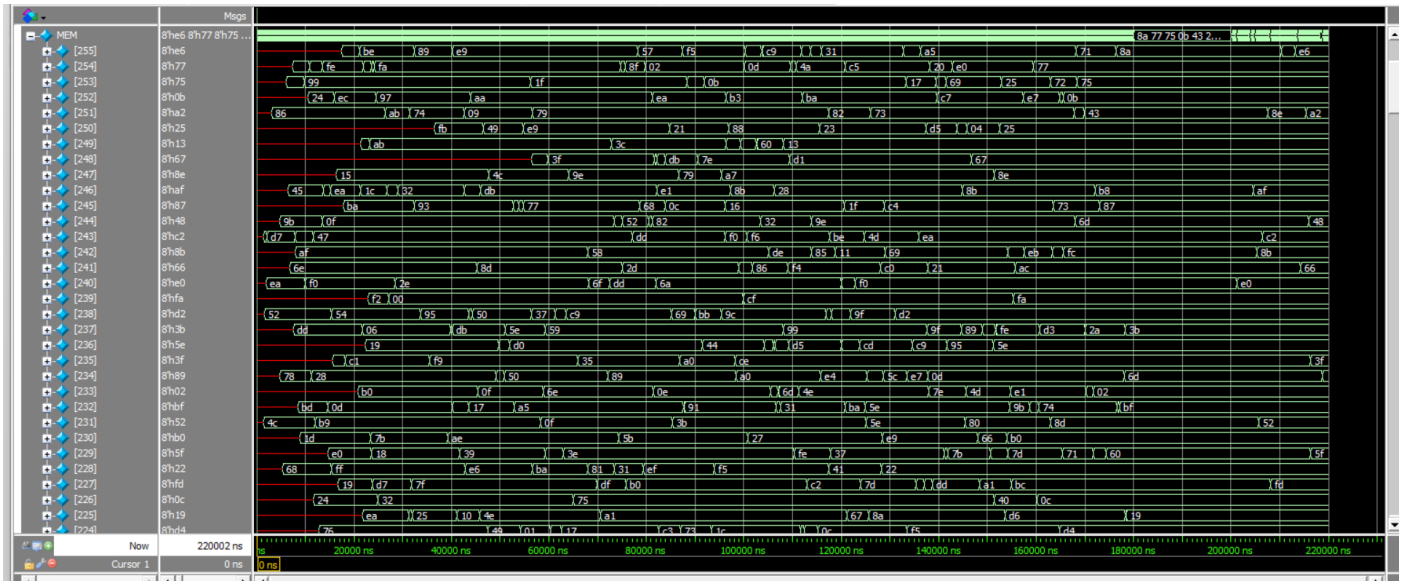
```
# ********************************************************************************
# UVM_INFO WRAPPER_test.sv(71) @ 2: uvm_test_top [run_phase] reset deasserted
# UVM_INFO WRAPPER_test.sv(73) @ 2: uvm_test_top [run_phase] write only sequence started
# UVM_INFO WRAPPER_test.sv(75) @ 180002: uvm_test_top [run_phase] write only sequence finished
# UVM_INFO WRAPPER_test.sv(77) @ 180002: uvm_test_top [run_phase] read only sequence started
# UVM_INFO WRAPPER_test.sv(79) @ 200002: uvm_test_top [run_phase] read only sequence finished
# UVM_INFO WRAPPER_test.sv(81) @ 200002: uvm_test_top [run_phase] read/write sequence started
# UVM_INFO WRAPPER_test.sv(83) @ 220002: uvm_test_top [run_phase] read/write sequence finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 220002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO WRAPPER_scoreboard.sv(23) @ 220002: uvm_test_top.env.sb [SCOREBOARD] Simulation Summary:
#
#               correct_count=110001, error_count=0
# UVM_INFO RAM_scoreboard.sv(53) @ 220002: uvm_test_top.env_ram.sb [report_phase] Total successful transactions : 110001
# UVM_INFO RAM_scoreboard.sv(54) @ 220002: uvm_test_top.env_ram.sb [report_phase] Total failed transactions : 0
# UVM_INFO SLAVE_scoreboard.sv(44) @ 220002: uvm_test_top.env_slave.sb [SCOREBOARD] Simulation Summary: correct_count=110001, error_count=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [SCOREBOARD]     2
# [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: 220002 ns  Iteration: 61  Instance: /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
```

Waveforms:

- Complete waveform:

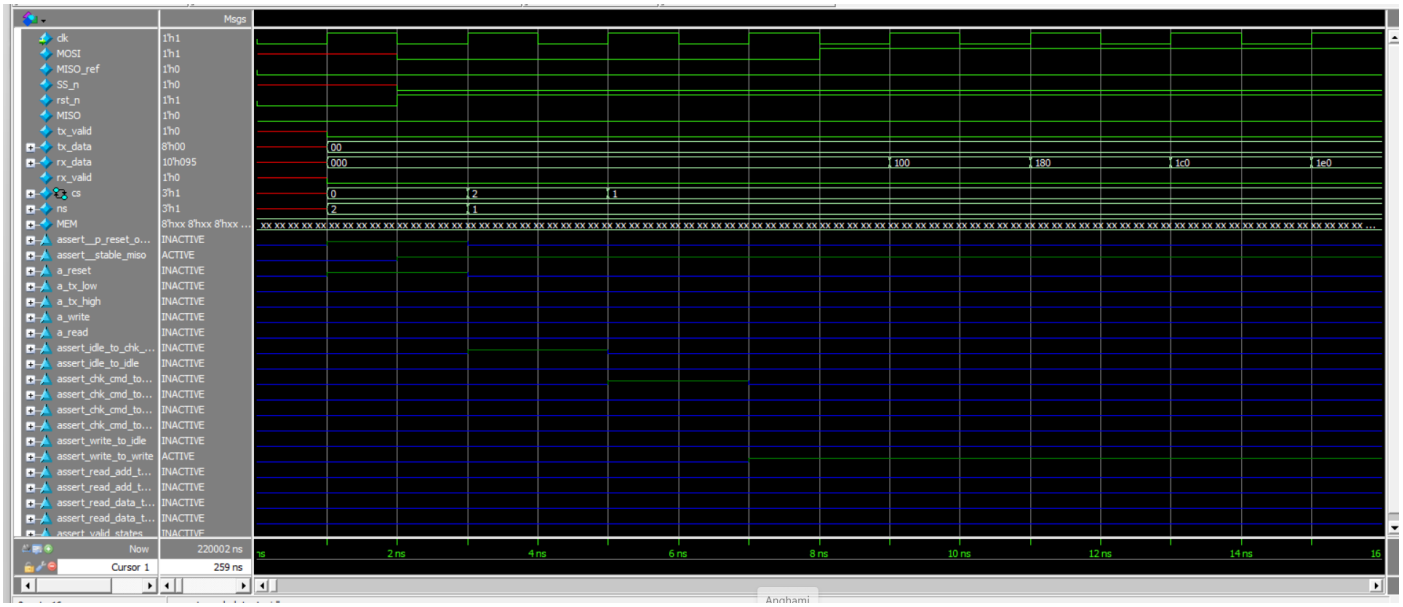


➢ RAM behavior

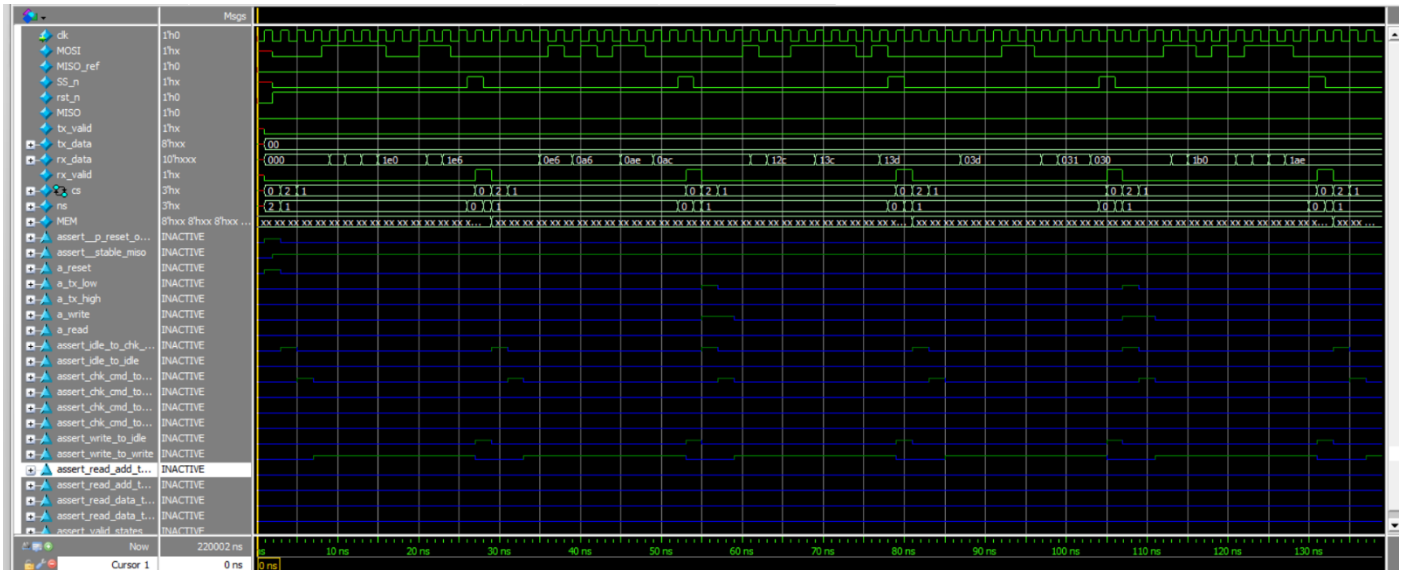


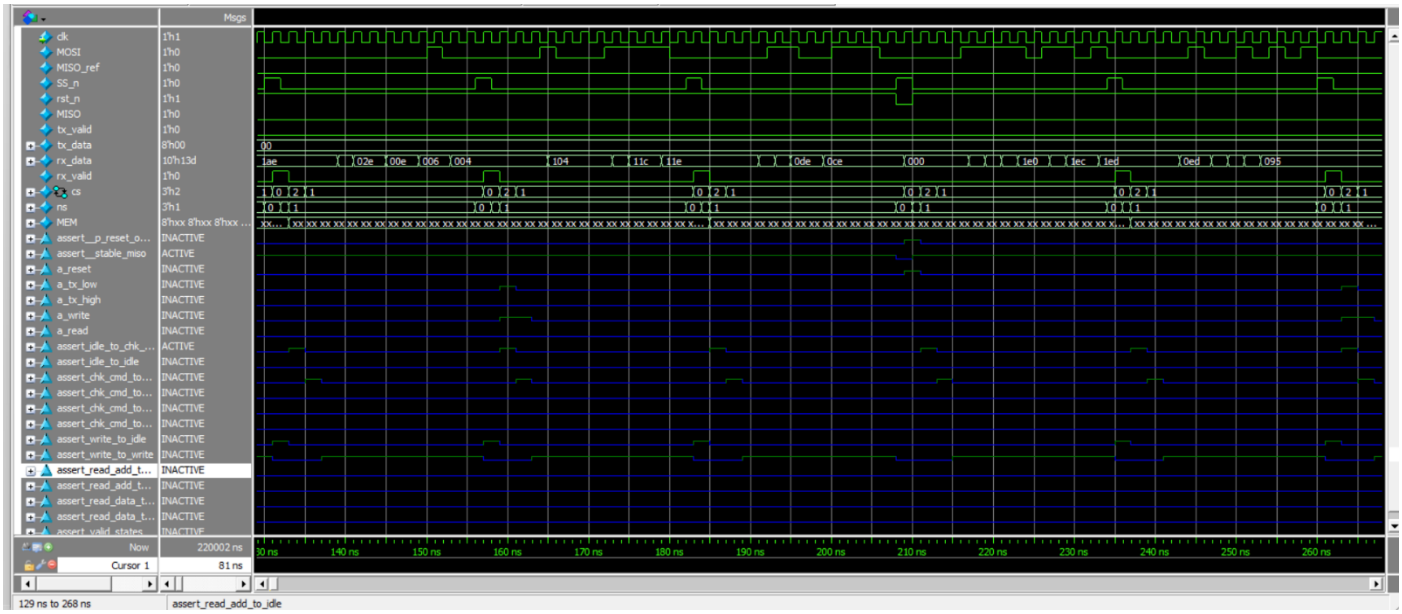
During the time interval from **180,000 ns to 200,000 ns**, the design operates exclusively in a **read-only sequence**, resulting in **no modifications to the RAM content**. This confirms that memory integrity is preserved during read operations, with data being accessed but not altered.

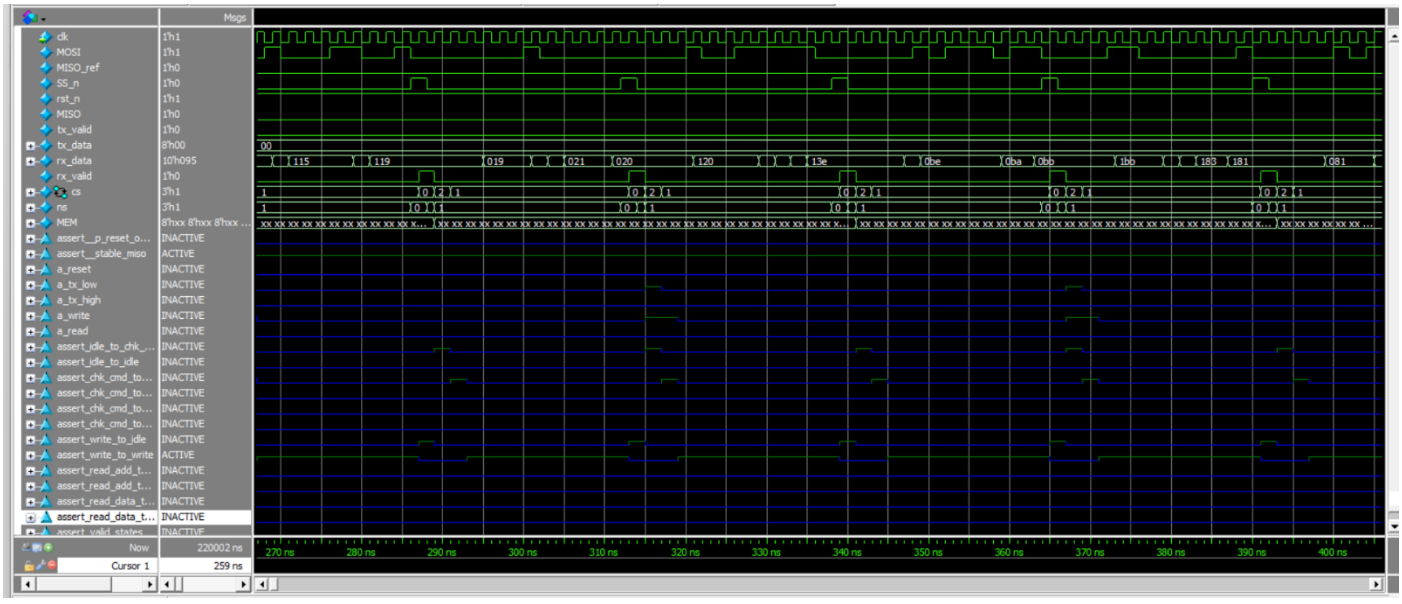
➢ Rest sequence:



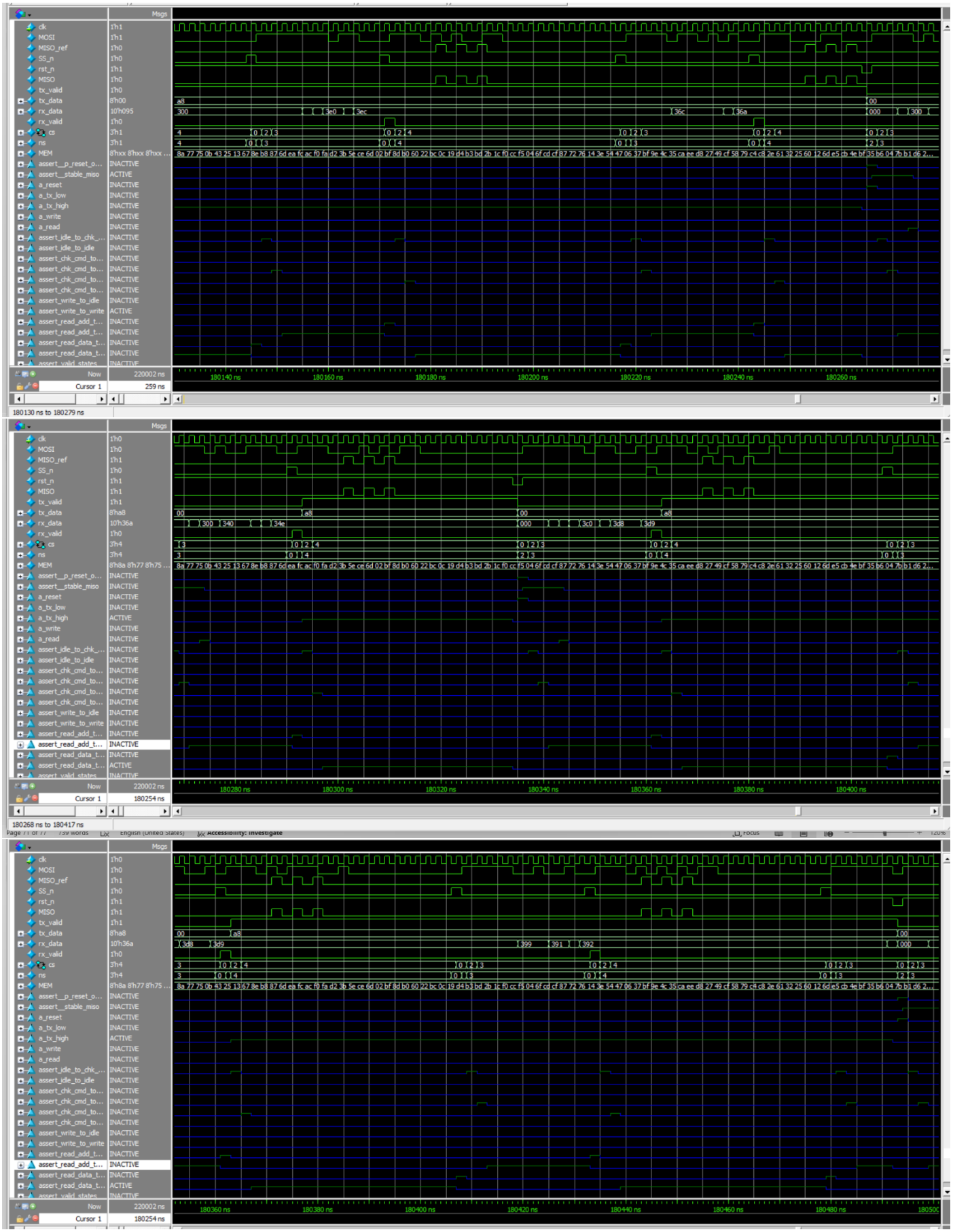
➢ Write only seq :



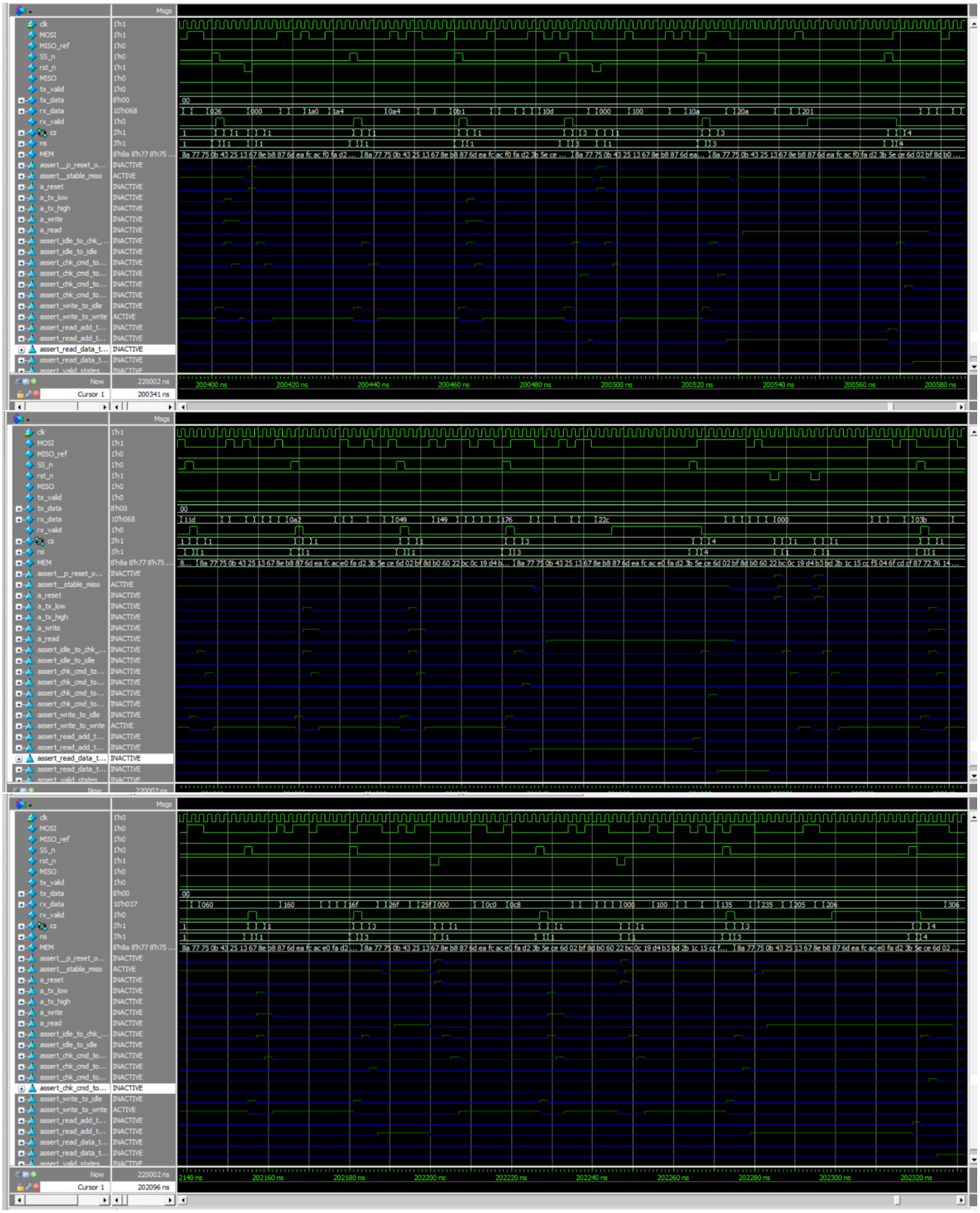




- Read only seq:



➢ Read_write seq :



Assertions coverage:

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT1/cover__stable_miso	SVA	✓	Off	95925	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/cover__p_reset_outputs_inactive	SVA	✓	Off	1179	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/RAM_instance/RAM_assertion/c_reset	SVA	✓	Off	1179	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/RAM_instance/RAM_assertion/c_tx_low	SVA	✓	Off	3181	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/RAM_instance/RAM_assertion/c_tx_high	SVA	✓	Off	55	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/RAM_instance/RAM_assertion/c_write	SVA	✓	Off	3148	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/RAM_instance/RAM_assertion/c_read	SVA	✓	Off	3052	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_idle_to_chk_cmd	SVA	✓	Off	8530	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_write	SVA	✓	Off	7611	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_add	SVA	✓	Off	399	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_data	SVA	✓	Off	423	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_write_to_idle	SVA	✓	Off	6782	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_read_add_to_idle	SVA	✓	Off	336	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/DUT1/SLAVE_instance/cover_read_data_to_idle	SVA	✓	Off	343	1	Unli...	1	100%		✓	0	0	0 ns	0

/top/DUT1/assert__p_reset_outputs_inactive	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge wrapperif.clk) (... ✓
/top/DUT1/assert__stable_miso	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(wrapperif.clk) disable iff ... ✓
/top/DUT1/RAM_instance/RAM_assertion/a_reset	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (~rst_n)|=... ✓
/top/DUT1/RAM_instance/RAM_assertion/a_tx_low	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/RAM_instance/RAM_assertion/a_tx_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/RAM_instance/RAM_assertion/a_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/RAM_instance/RAM_assertion/a_read	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_idle_to_chk_cmd	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_idle_to_idle	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_chk_cmd_to_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_chk_cmd_to_read_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_chk_cmd_to_read_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_chk_cmd_to_idle	Concurrent	SVA	on	0	0	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_write_to_idle	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_write_to_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_read_add_to_idle	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_read_add_to_read_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_read_data_to_idle	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_read_data_to_read_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓
/top/DUT1/SLAVE_instance/assert_valid_states	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (... ✓

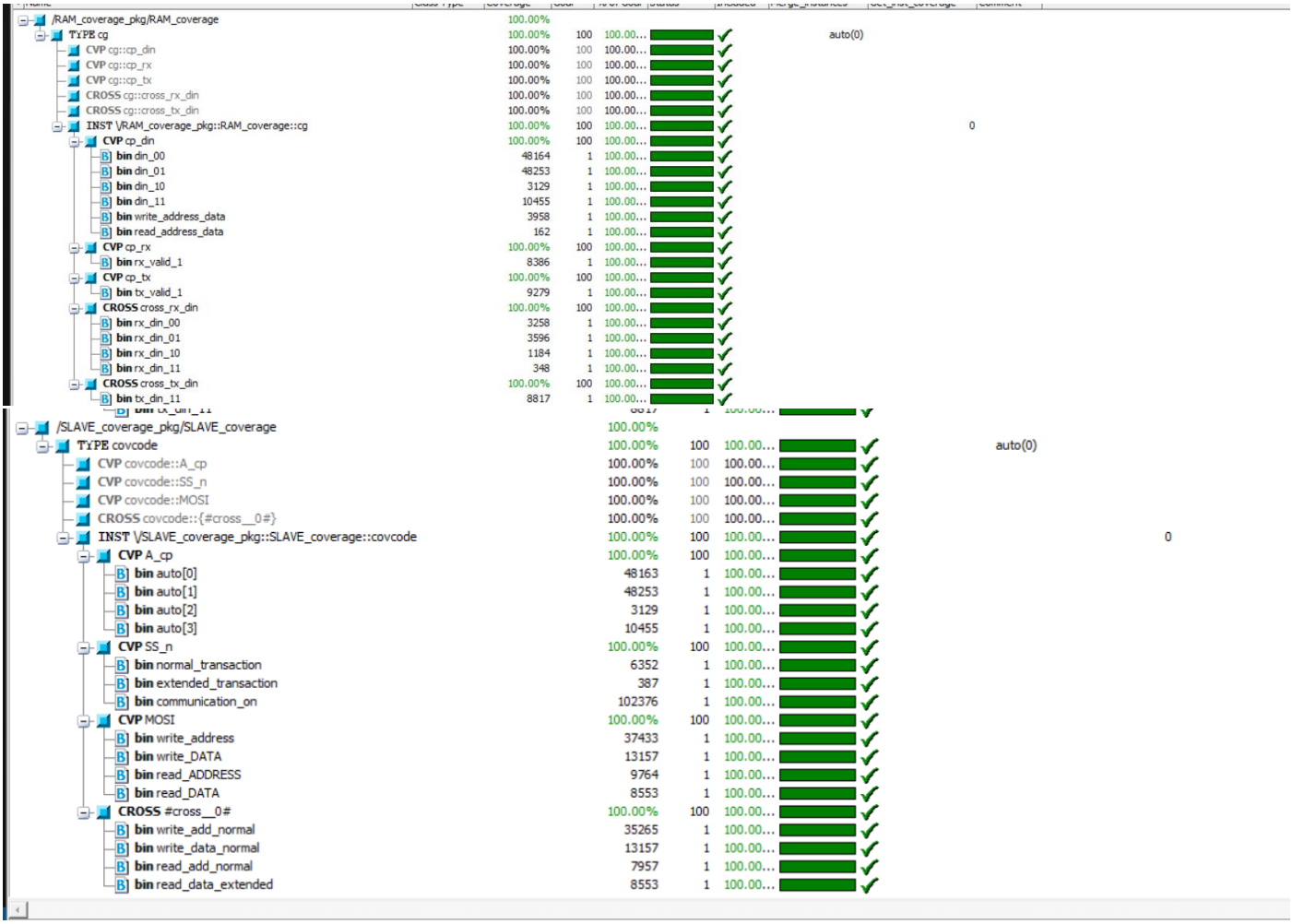
Note: The assertions idle_to_idle and chk_cmd_to_idle were not hit during simulation. This behavior is expected, as the verification environment constrains the SS_n signal to remain low throughout active SPI operations. Consequently, transitions involving SS_n being high in the middle of a transaction cannot occur, and these assertions remain untriggered by design.

```
================================================================================
=== Instance: /top/DUT1/SLAVE_instance
=== Design Unit: work.SLAVE
================================================================================

Directive Coverage:
    Directives                       7         7         0    100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)       Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------------
/top/DUT1/SLAVE_instance/cover_idle_to_chk_cmd
                                         SLAVE  Verilog  SVA  SPI_slave.sv(263)
                                                                      8530 Covered
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_write
                                         SLAVE  Verilog  SVA  SPI_slave.sv(264)
                                                                      7611 Covered
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_add
                                         SLAVE  Verilog  SVA  SPI_slave.sv(265)
                                                                       399 Covered
/top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_data
                                         SLAVE  Verilog  SVA  SPI_slave.sv(266)
                                                                       423 Covered
/top/DUT1/SLAVE_instance/cover_write_to_idle
                                         SLAVE  Verilog  SVA  SPI_slave.sv(267)
                                                                      6782 Covered
/top/DUT1/SLAVE_instance/cover_read_add_to_idle
                                         SLAVE  Verilog  SVA  SPI_slave.sv(268)
                                                                       336 Covered
/top/DUT1/SLAVE_instance/cover_read_data_to_idle
                                         SLAVE  Verilog  SVA  SPI_slave.sv(269)
                                                                       343 Covered
```

```
================================================================================
=== Instance: /top/DUT1/RAM_instance/RAM_assertion
=== Design Unit: work.RAM_sva
================================================================================

Directive Coverage:
    Directives                       5         5         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/top/DUT1/RAM_instance/RAM_assertion/c_reset
                                         RAM_sva Verilog  SVA  RAM_assertions.sv(15)
                                                                      1179 Covered
/top/DUT1/RAM_instance/RAM_assertion/c_tx_low
                                         RAM_sva Verilog  SVA  RAM_assertions.sv(24)
                                                                      3181 Covered
/top/DUT1/RAM_instance/RAM_assertion/c_tx_high
                                         RAM_sva Verilog  SVA  RAM_assertions.sv(33)
                                                                        55 Covered
/top/DUT1/RAM_instance/RAM_assertion/c_write
                                         RAM_sva Verilog  SVA  RAM_assertions.sv(41)
                                                                      3148 Covered
/top/DUT1/RAM_instance/RAM_assertion/c_read
                                         RAM_sva Verilog  SVA  RAM_assertions.sv(49)
                                                                      3052 Covered

--------------------------------------------------------------------------------
```

```
================================================================================
=== Instance: /top/DUT1
=== Design Unit: work.wrapper
================================================================================

Directive Coverage:
    Directives                       2         2         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/top/DUT1/cover__stable_miso             wrapper Verilog  SVA  WRAPPER.sv(35)  195925 Covered
/top/DUT1/cover__p_reset_outputs_inactive
                                         wrapper Verilog  SVA  WRAPPER.sv(27)  1179 Covered
```

Function coverage:

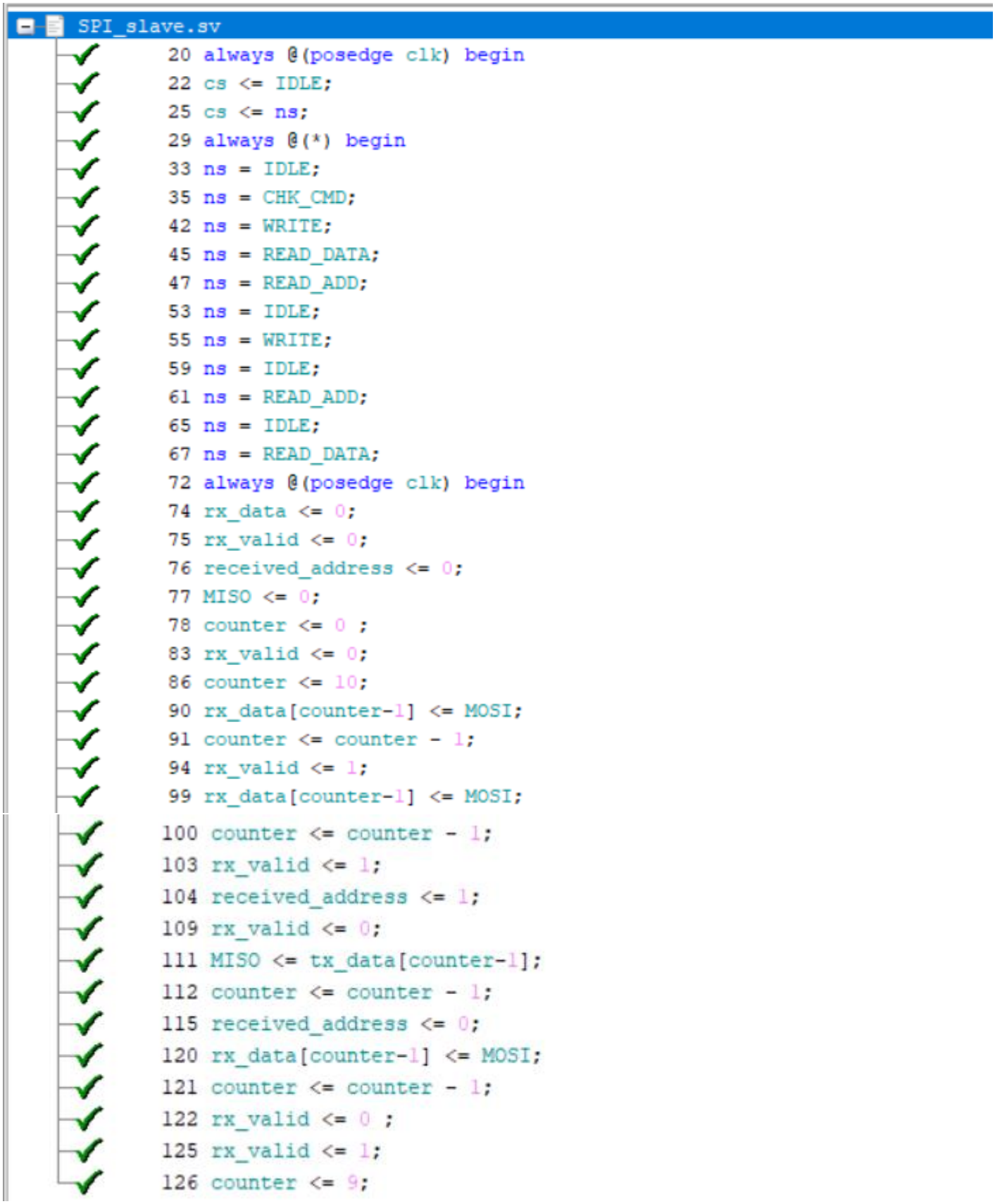



```
================================================================================
=== Instance: /RAM_coverage_pkg
=== Design Unit: work.RAM_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses     5        na        na        na
            Covergroup Bins    13        13         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
 TYPE /RAM_coverage_pkg/RAM_coverage/cg             100.00%        100          -    Covered
    covered/total bins:                                  13         13          -
    missing/total bins:                                   0         13          -
    % Hit:                                           100.00%        100          -
    Coverpoint cp_din                                100.00%        100          -    Covered
        covered/total bins:                               6          6          -
        missing/total bins:                               0          6          -
        % Hit:                                       100.00%        100          -
    Coverpoint cp_rx                                 100.00%        100          -    Covered
        covered/total bins:                               1          1          -
        missing/total bins:                               0          1          -
        % Hit:                                       100.00%        100          -
    Coverpoint cp_tx                                 100.00%        100          -    Covered
        covered/total bins:                               1          1          -
        missing/total bins:                               0          1          -
        % Hit:                                       100.00%        100          -
    Cross cross_rx_din                               100.00%        100          -    Covered
        covered/total bins:                               4          4          -
        missing/total bins:                               0          4          -
        % Hit:                                       100.00%        100          -
    Cross cross_tx_din                               100.00%        100          -    Covered
        covered/total bins:                               1          1          -
        missing/total bins:                               0          1          -
        % Hit:                                       100.00%        100          -
```

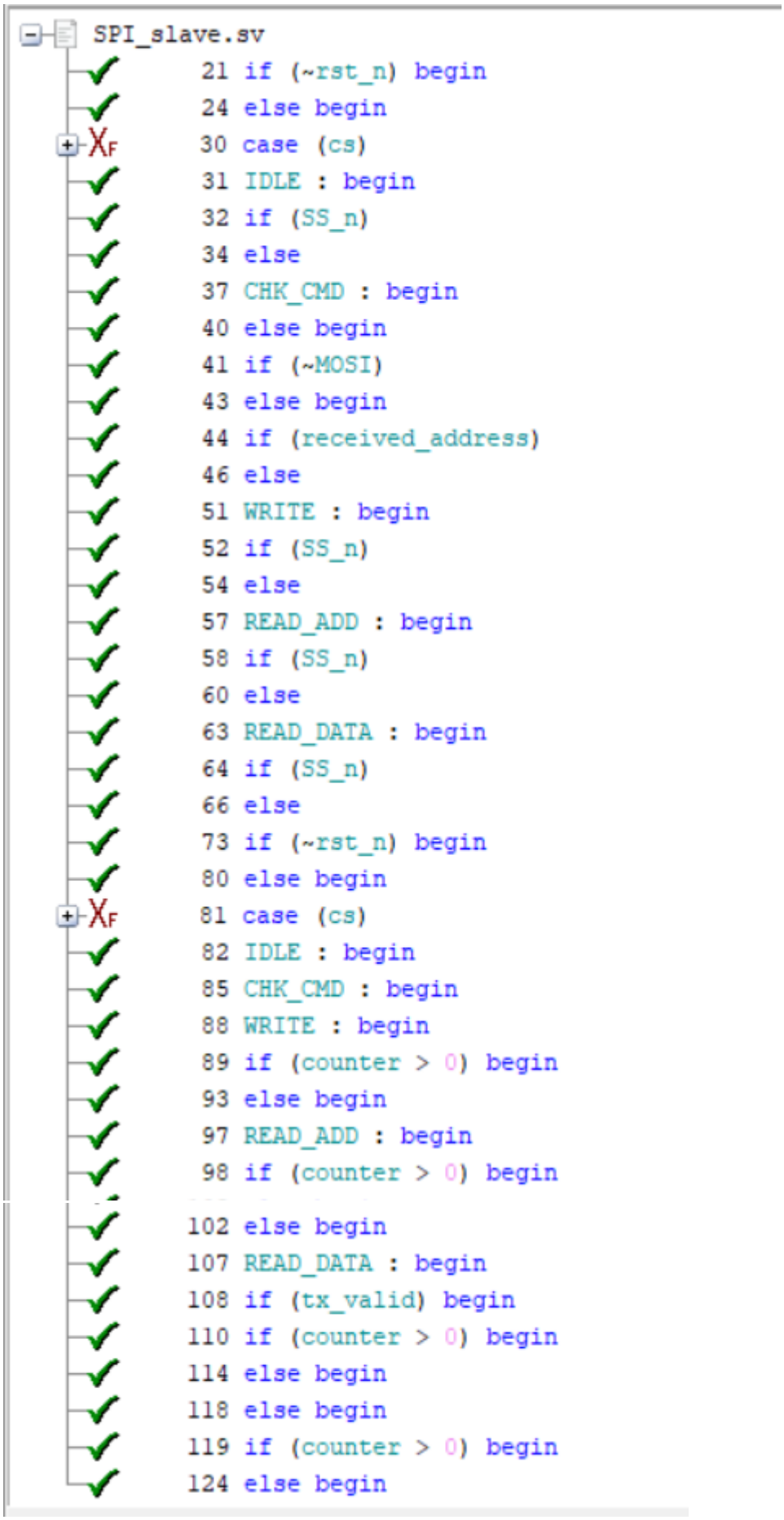
```
================================================================================
=== Instance: /SLAVE_coverage_pkg
=== Design Unit: work.SLAVE_coverage_pkg
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses         4        na        na        na
            Covergroup Bins        15        15         0   100.00%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
 TYPE /SLAVE_coverage_pkg/SLAVE_coverage/covcode    100.00%        100          -    Covered
    covered/total bins:                                  15         15          -
    missing/total bins:                                   0         15          -
    % Hit:                                           100.00%        100          -
    Coverpoint A_cp                                  100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
    Coverpoint SS_n                                  100.00%        100          -    Covered
        covered/total bins:                                3          3          -
        missing/total bins:                                0          3          -
        % Hit:                                       100.00%        100          -
    Coverpoint MOSI                                  100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
    Cross #cross__0#                                 100.00%        100          -    Covered
        covered/total bins:                                4          4          -
        missing/total bins:                                0          4          -
        % Hit:                                       100.00%        100          -
 Covergroup instance \/SLAVE_coverage_pkg::SLAVE_coverage::covcode
```

Code coverage:

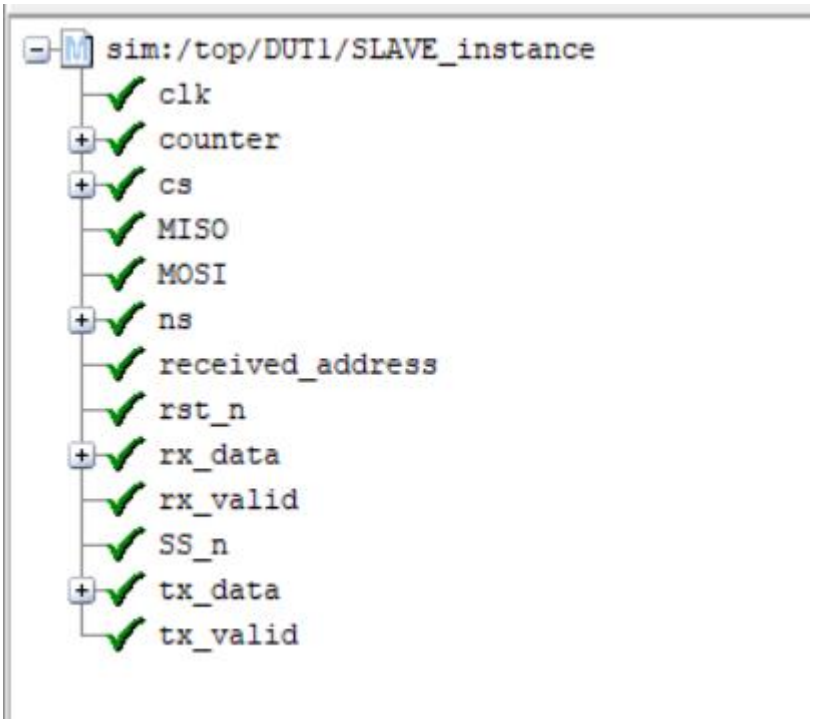
- Slave Statements:



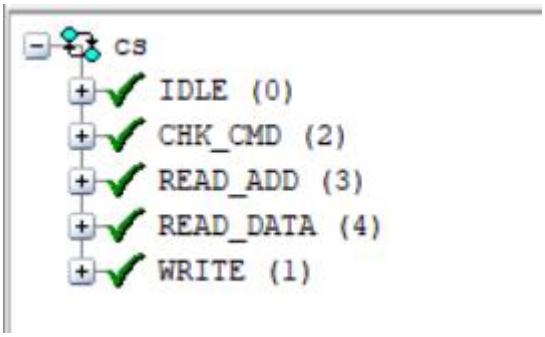
-Slave Branches:



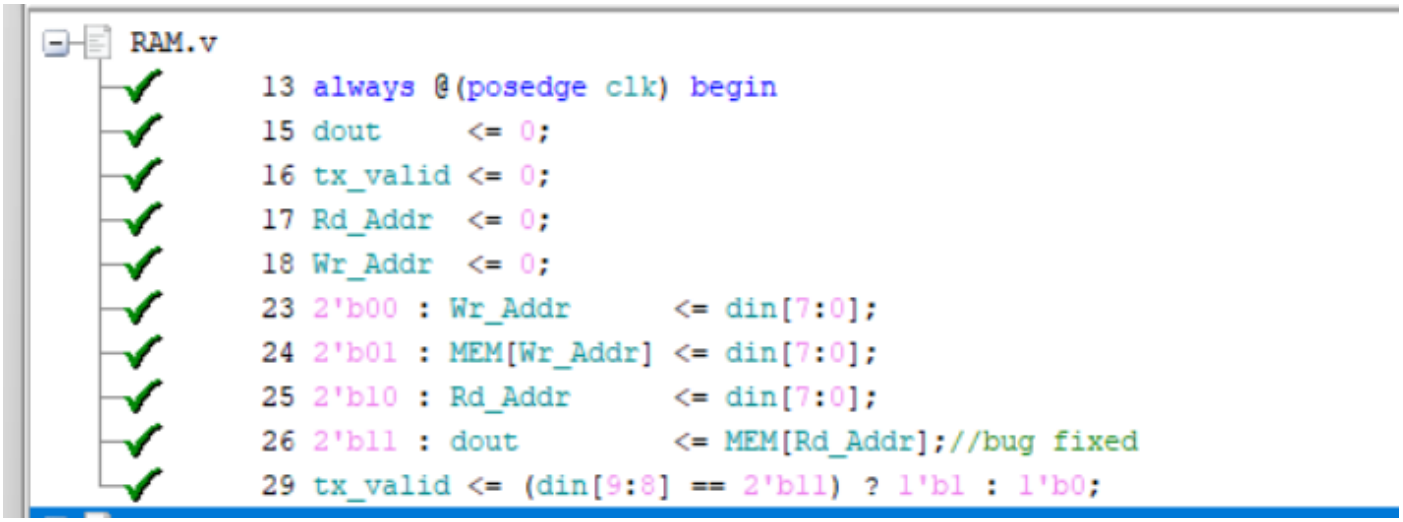
-Slave Toggles:



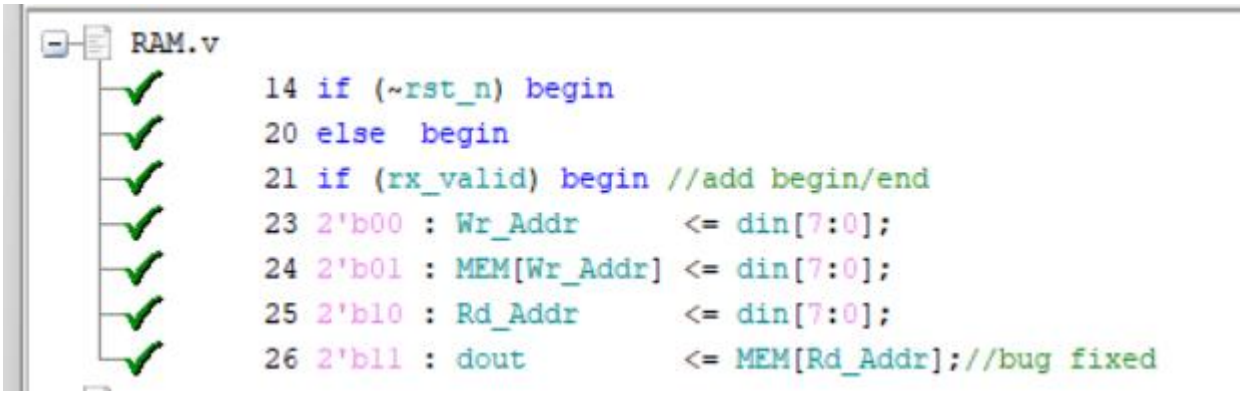
-Slave FSMs:



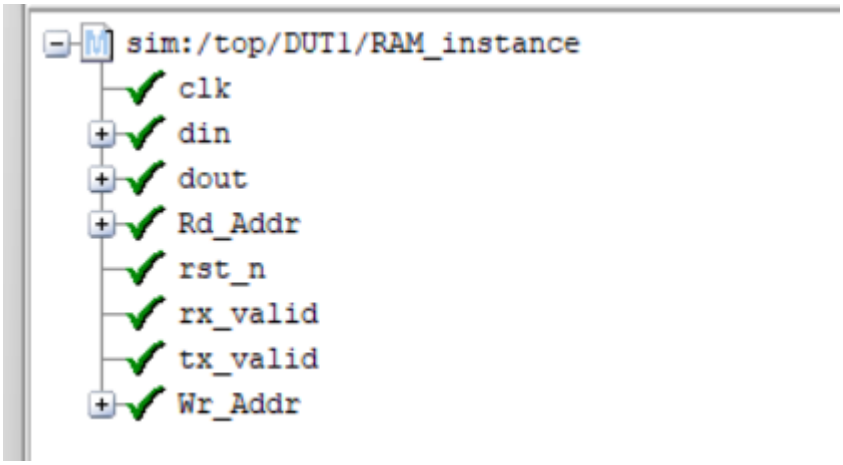
-RAM Statements:



-RAM Branches:



-RAM Toggles:



Verification plan:

Column1	Column2	Column3	Column4	Column5
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
WRAPPER_0	Write-Only Sequence:Every Write Address operation shall always be followed by Write Data operation	Constrained randomization withc_write_onlyconstraint enabled:If prev_cmd =000→ next cmd =001If prev_cmd =001→ next cmd =000	Cross coverage between consecutive commandsVerify only000→001and001→000transitions occurCount sequence length and verify 90,000 iterations	Scoreboard checks MISO vs MISO_refSVA verifies no read operations occurMonitor validates command sequence ordering
WRAPPER_1	Read-Only Sequence:Only Read Address or Read Data operations are allowed	Constrained randomization withc_read_onlyconstraint enabled:If prev_cmd =110→ next cmd =111If prev_cmd =111→ next cmd =110	Cross coverage between consecutive commandsVerify only110→111and111→110transitions occurCount sequence length and verify 10,000 iterationsMonitor tx_valid assertion for read data	Scoreboard validates MISO data integritySVA verifies no write operations occurCheck MISO stability during non-read operations
WRAPPER_2	Read-Write Mixed Sequence:Random mix of all operation types with realistic transitions	Constrained randomization withc_read_writeconstraint:If prev =000→ next in {000, 001}If prev =001→000(40%) or110(60%)If prev =110→ next =111If prev =111→000(60%) or110(40%)	Cross coverage of all valid command transitionsCover address range distribution (0x00-0xFF)Track write-then-read to same addressMonitor transition distribution matches constraintsVerify 10,000 mixed operations completed	Scoreboard validates all MISO outputsCheck data integrity: write then read same addressSVA verifies timing for both 13 and 23 cycle periodsValidate RAM state consistency
WRAPPER_3	Address Range Coverage:All memory addresses (0x00 to 0xFF) should be accessed	Address randomization with distribution constraint:MOSI_array[7:0] dist {[8'h00:8'hFF] := 100}	Coverpoint for all 256 address valuesSeparate bins for boundary addresses: 0x00, 0xFFTrack addresses accessed during write vs read	Verify coverage report shows 100% address coverageCheck memory initialization and read-back
WRAPPER_4	Reset During Transaction:Reset assertion during active transaction should safely abort operation	Reset constraint with occasional assertion:rst_n dist {1 := 99, 0 := 1}	Cover reset during each state (IDLE, WRITE, CMDCHK, RDADDR, RDDATA)Monitor recovery after reset de-assertion	SVA:!rst_n |=> (MISO==0 && rx_valid==0 && rx_data==0)Verify all internal state resets properly
WRAPPER_5	MISO Stability:MISO remains stable when not performing read data operation	Generated automatically through all sequences (write-only, read-only, mixed)	Cover MISO changes only during read dataCover MISO stable during write operations	SVA:(rx_data[9:8] != 2'b11) |=> $stable(MISO)Monitor MISO transitions in scoreboard

Fourth : assertion table:

RAM_sva Module Assertions

Assertion Name	Property Name	Description	Purpose
a_reset	p_reset	Reset check: dout=0 and tx_valid=0 when !rst_n	Verify outputs are cleared on reset
a_tx_low	p_tx_low	tx_valid is low when din[9:8]=00 and rx_valid=1	Check tx_valid low for write address command
a_tx_high	p_tx_high	tx_valid rises then falls when din[9:8]=11 and rx_valid=1	Verify read data produces valid tx pulse
a_write	p_write	Write address followed by write data (00→00/01)	Ensure proper write sequence
a_read	p_read	Read address followed by read data (10→10/11)	Ensure proper read sequence

---

FSM Transition Assertions (Conditional - ifdef SIM)

Assertion Name	Property Name	Transition	Condition	Description
assert_idle_to_chk_cmd	idle_to_chk_cmd	IDLE → CHK_CMD	SS_n = 0	Start of SPI transaction
assert_idle_to_idle	idle_to_idle	IDLE → IDLE	SS_n = 1	Remain idle when not selected
assert_chk_cmd_to_write	chk_cmd_to_write	CHK_CMD → WRITE	MOSI=0, SS_n=0	Write command detected
assert_chk_cmd_to_read_add	chk_cmd_to_read_add	CHK_CMD → READ_ADD	MOSI=1, !received_address, SS_n=0	Read address phase start
assert_chk_cmd_to_read_data	chk_cmd_to_read_data	CHK_CMD → READ_DATA	MOSI=1, received_address, SS_n=1	Read data phase start
assert_chk_cmd_to_idle	chk_cmd_to_idle	CHK_CMD → IDLE	SS_n = 1	Transaction aborted early

Assertion Name	Property Name	Transition	Condition	Description
assert_write_to_idle	write_to_idle	WRITE → IDLE	SS_n = 1	Write operation completed
assert_write_to_write	write_to_write	WRITE → WRITE	SS_n = 0	Continue write operation
assert_read_add_to_idle	read_add_to_idle	READ_ADD → IDLE	SS_n = 1	Read address aborted
assert_read_add_to_read_add	read_add_to_read_add	READ_ADD → READ_ADD	SS_n = 0	Continue read address
assert_read_data_to_idle	read_data_to_idle	READ_DATA → IDLE	SS_n = 1	Read data completed
assert_read_data_to_read_data	read_data_to_read_data	READ_DATA → READ_DATA	SS_n = 0	Continue read data
assert_valid_states	valid_states_only	State Check	Always	FSM is in valid state only

Wrapper Interface Assertions

Assertion Name	Property Name	Description	Purpose
Assert_reset	p_reset_outputs_inactive	MISO=0, rx_valid=0, rx_data=0 when !rst_n	Verify all outputs inactive on reset
Miso_stability	stable_miso	MISO stable when not read data (rx_data[9:8]≠11)	MISO should only change during read data