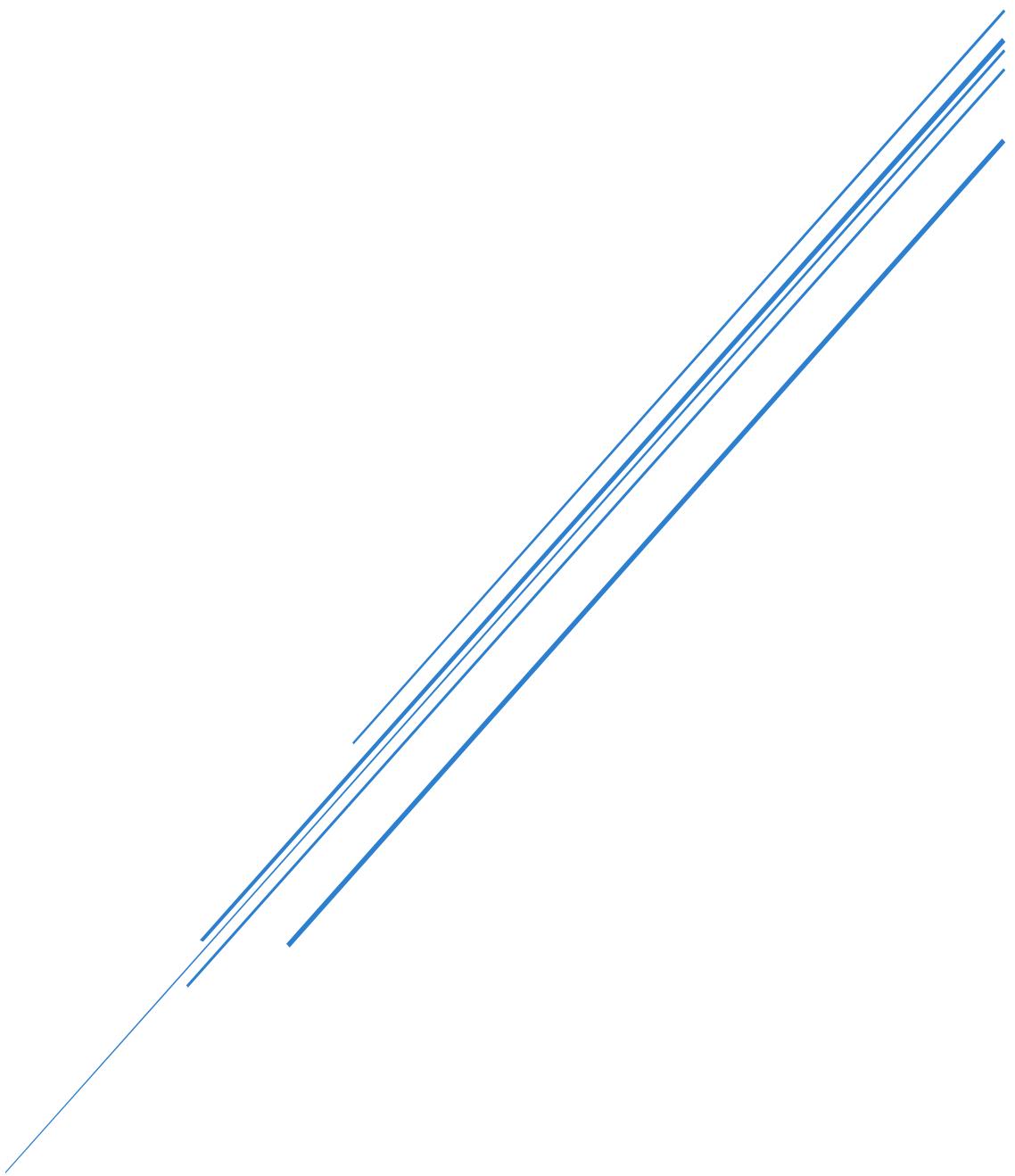


System Verilog project

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

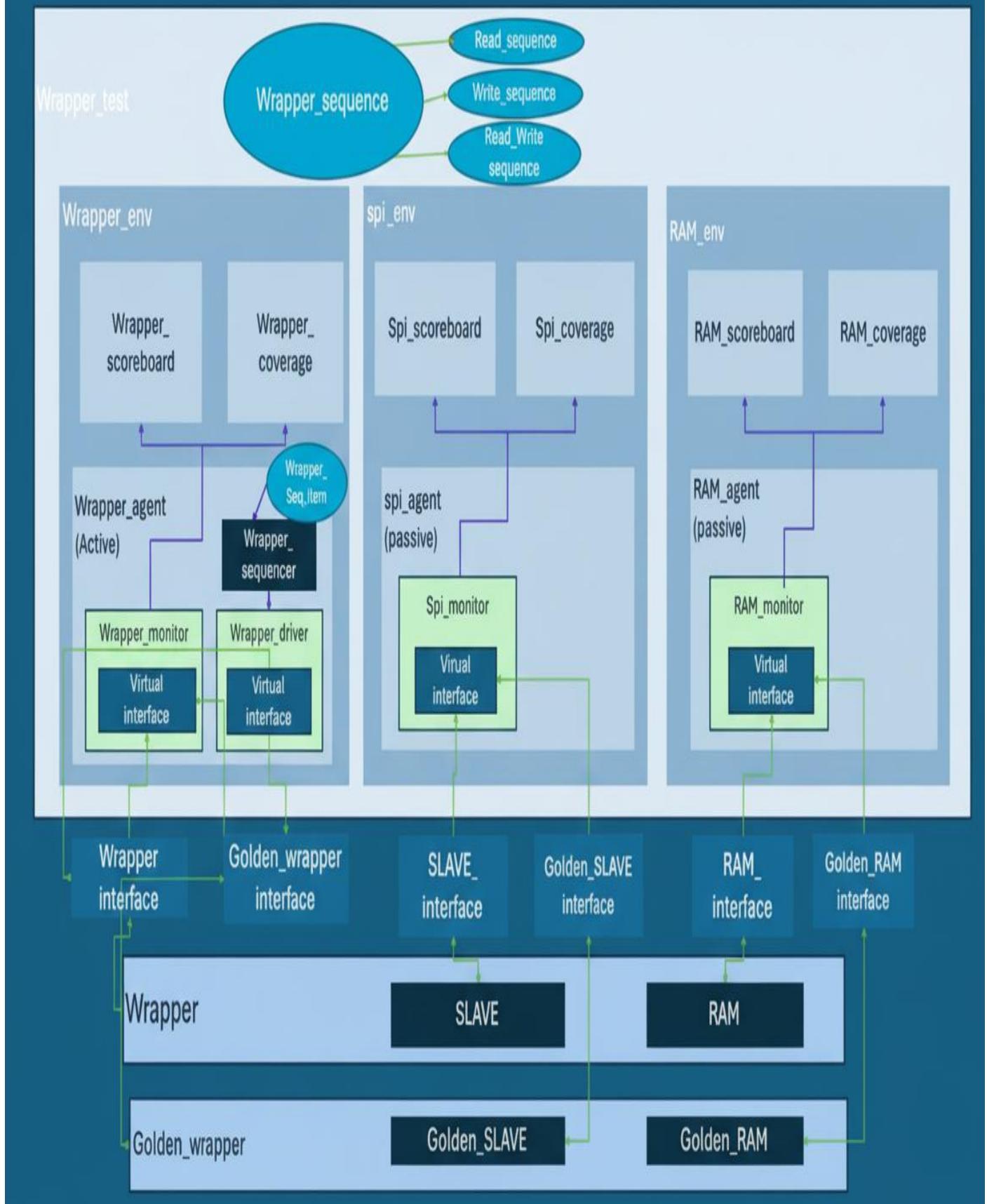
Project Title: SPI (Serial Peripheral Interface) slave with single-port RAM

1. UVM Test Structure
1.1 Overview
1.2 Wrapper Environment (Active Agent)
1.3 SPI and RAM Environments (Passive Agents)
1.4 Sequences and Transaction Flow
1.5 Scoreboards and Golden Models
1.6 Coverage and Assertions
1.7 Test Execution Flow
1.8 Summary
2. SLAVE Part
2.1 Design and Bug Analysis
Bug 1: Counter Reset Behavior
Bug 2: Command Validation Handling
Bug 3: Read Data Counter Adjustment
2.2 Complete Design and Assertions
Final Design (Post-Fix)
Assertions Implementation
2.3 Interface and Top-Level Files
SLAVE Interface
Top Module
2.4 UVM Components
SLAVE Test File
Sequence Item File
Sequencer File
Reset Sequence
Main Sequence
Configuration File
Environment File
Agent File
Driver File
Monitor File
Scoreboard File
Coverage File
2.5 Simulation Results
Simulation Transcript
Waveforms
Assertion Coverage
Functional Coverage
Code Coverage
2.6 Verification plan
3. RAM Part
3.1 Design and Bug Analysis
Bug: Read Pointer Address Handling
3.2 Complete Design and Assertions
Final RAM Design (Post-Fix)
RAM Golden Model
Assertion File
3.3 Interface and Top-Level Files
RAM Interface
Top Module
3.4 UVM Components
RAM Test File
Sequence Item File
Sequencer File
Reset Sequence
Write Sequence
Read Sequence
Read-Write Sequence
Configuration File

Environment File	
Agent File	
Driver File	
Monitor File	
Scoreboard File	
Coverage File	
3.5 Simulation Results	
Simulation Transcript	
Waveforms	
Full Waveform	
Reset Sequence Waveform	
Write-Only Sequence Waveform	
Read-Only Sequence Waveform	
Write-Read Sequence Waveform	
Assertion Coverage	
Functional Coverage	
Code Coverage	
3.6 Verification plan	
<hr/>	
4. Wrapper Part	
4.1 Integration Updates	
Modified Files in RAM and SLAVE Environments	
SLAVE Configuration Object	
RAM Configuration Object	
SLAVE Agent File	
RAM Agent File	
4.2 Wrapper Environment and Design	
Wrapper Interface File	
Wrapper Design and Assertion File	
Wrapper Golden Model	
Wrapper Top File	
Wrapper Test File	
Sequence Item File	
Configuration Object	
4.3 Wrapper Sequences	
Write-Only Sequence	
Read-Only Sequence	
Read-Write Sequence	
4.4 UVM Components	
Wrapper Sequencer	
Wrapper Environment	
Wrapper Agent	
Wrapper Driver	
Wrapper Monitor	
Wrapper Scoreboard	
Wrapper Coverage	
4.5 Simulation Results	
Simulation Transcript	
Complete Waveform	
RAM Behavior Waveform (180,000 ns – 200,000 ns Read-Only Period)	
Reset Sequence Waveform	
Write-Only Sequence Waveform	
Read-Only Sequence Waveform	
Read-Write Sequence Waveform	
Assertion Coverage	
Functional Coverage	
SLAVE Code Coverage	
RAM Code Coverage	
4.6 Verification plan	
5. Assertion Table	
5.1 RAM_sva Module Assertions	
5.2 FSM Transition Assertions (Conditional - ifdef SIM)	
5.3 Wrapper Interface Assertions	

UVM test structure

Wrapper_top



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

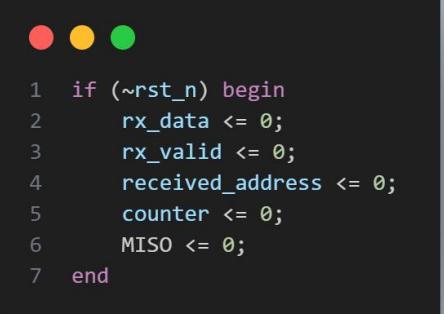
First: Snippets from codes:

Design and bugs found:

- Bug 1 :



```
1 if (~rst_n) begin
2   rx_data <= 0;
3   rx_valid <= 0;
4   received_address <= 0;
5   MISO <= 0;
6 end
```

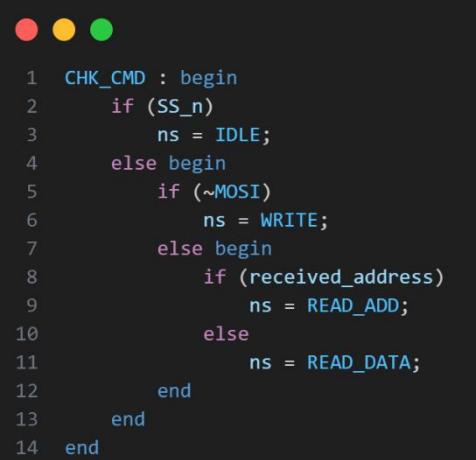


```
1 if (~rst_n) begin
2   rx_data <= 0;
3   rx_valid <= 0;
4   received_address <= 0;
5   counter <= 0;
6   MISO <= 0;
7 end
```

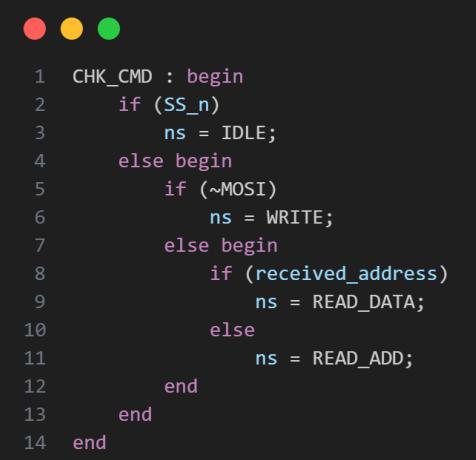
- Counter Reset Behavior:

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 :



```
1 CHK_CMD : begin
2   if (SS_n)
3     ns = IDLE;
4   else begin
5     if (~MOSI)
6       ns = WRITE;
7     else begin
8       if (received_address)
9         ns = READ_ADD;
10      else
11        ns = READ_DATA;
12    end
13  end
14 end
```



```
1 CHK_CMD : begin
2   if (SS_n)
3     ns = IDLE;
4   else begin
5     if (~MOSI)
6       ns = WRITE;
7     else begin
8       if (received_address)
9         ns = READ_DATA;
10      else
11        ns = READ_ADD;
12    end
13  end
14 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 :

```

1 READ_DATA : begin
2     if (tx_valid) begin
3         rx_valid <= 0;
4         if (counter > 0) begin
5             MISO <= tx_data[counter-1];
6             counter <= counter - 1;
7         end
8         else begin
9             received_address <= 0;
10        end
11    end
12    else begin
13        if (counter > 0) begin
14            rx_data[counter-1] <= MOSI;
15            counter <= counter - 1;
16        end
17        else begin
18            rx_valid <= 1;
19            counter <= 8;
20        end
21    end
22 end

```

```

1 READ_DATA : begin
2     if (tx_valid) begin
3         rx_valid <= 0;
4         if (counter > 0) begin
5             MISO <= tx_data[counter-1];
6             counter <= counter - 1;
7         end
8         else begin
9             received_address <= 0;
10        end
11    end
12    else begin
13        if (counter > 0) begin
14            rx_data[counter-1] <= MOSI;
15            counter <= counter - 1;
16        end
17        else begin
18            rx_valid <= 1;
19            counter <= 9;
20        end
21    end
22 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:

```

1 module SLAVE (MOSI,MISO,SS_n,clk,rst_n,rx_data,rx_valid,tx_data,tx_valid);
2
3 localparam IDLE      = 3'b000;
4 localparam WRITE     = 3'b001;
5 localparam CHK_CMD   = 3'b010;
6 localparam READ_ADD  = 3'b011;
7 localparam READ_DATA = 3'b100;
8
9 input      MOSI, clk, rst_n, SS_n, tx_valid;
10 input      [7:0] tx_data;
11 output reg [9:0] rx_data;
12 output reg      rx_valid;
13 output bit     MISO;
14
15 reg [3:0] counter;
16 reg      received_address;
17 reg [2:0] cs, ns;

```



```
1  lways @(posedge clk) begin
2      if (~rst_n) begin
3          cs <= IDLE;
4      end
5      else begin
6          cs <= ns;
7      end
8  end
9
10 always @(*) begin
11     case (cs)
12         IDLE : begin
13             if (SS_n)
14                 ns = IDLE;
15             else
16                 ns = CHK_CMD;
17         end
18         CHK_CMD : begin
19             if (SS_n)
20                 ns = IDLE;
21             else begin
22                 if (~MOSI)
23                     ns = WRITE;
24                 else begin
25                     if (received_address)
26                         ns = READ_DATA;
27                     else
28                         ns = READ_ADD;
29                 end
30             end
31         end
32         WRITE : begin
33             if (SS_n)
34                 ns = IDLE;
35             else
36                 ns = WRITE;
37         end
38         READ_ADD : begin
39             if (SS_n)
40                 ns = IDLE;
41             else
42                 ns = READ_ADD;
43         end
44         READ_DATA : begin
45             if (SS_n)
46                 ns = IDLE;
47             else
48                 ns = READ_DATA;
49         end
50     endcase
51 end
```



```
1  always @(posedge clk) begin
2      if (~rst_n) begin
3          rx_data <= 0;
4          rx_valid <= 0;
5          received_address <= 0;
6          counter <= 0;
7          MISO <= 0;
8      end
9      else begin
10         case (cs)
11             IDLE : begin
12                 rx_valid <= 0;
13             end
14             CHK_CMD : begin
15                 counter <= 10;
16             end
17             WRITE : begin
18                 if (counter > 0) begin
19                     rx_data[counter-1] <= MOSI;
20                     counter <= counter - 1;
21                 end
22                 else begin
23                     rx_valid <= 1;
24                 end
25             end
26             READ_ADD : begin
27                 if (counter > 0) begin
28                     rx_data[counter-1] <= MOSI;
29                     counter <= counter - 1;
30                 end
31                 else begin
32                     rx_valid <= 1;
33                     received_address <= 1;
34                 end
35             end
36             READ_DATA : begin
37                 if (tx_valid) begin
38                     rx_valid <= 0;
39                     if (counter > 0) begin
40                         MISO <= tx_data[counter-1];
41                         counter <= counter - 1;
42                     end
43                     else begin
44                         received_address <= 0;
45                     end
46                 end
47                 else begin
48                     if (counter > 0) begin
49                         rx_data[counter-1] <= MOSI;
50                         counter <= counter - 1;
51                     end
52                     else begin
53                         rx_valid <= 1;
54                         counter <= 9;
55                     end
56                 end
57             end
58         endcase
59     end
60 end
```

```

1  `ifdef SIM
2    // property check reset
3    property check_reset;
4      @(posedge clk)
5        (!rst_n) |=> (rx_valid == 0 && rx_data == 0 && MISO == 0);
6    endproperty
7
8    // Property: IDLE → CHK_CMD transition
9    property idle_to_chk_cmd;
10      @(posedge clk) disable iff (!rst_n)
11        (cs == IDLE && !SS_n) |=> (cs == CHK_CMD);
12    endproperty
13
14    // Property: IDLE → IDLE when SS_n is high
15    property idle_to_idle;
16      @(posedge clk) disable iff (!rst_n)
17        (cs == IDLE && SS_n) |=> (cs == IDLE);
18    endproperty
19
20    // Property: CHK_CMD → WRITE (when MOSI=0 and SS_n=0)
21    property chk_cmd_to_write;
22      @(posedge clk) disable iff (!rst_n)
23        (cs == CHK_CMD && !SS_n && !MOSI) |=> (cs == WRITE);
24    endproperty
25
26    // Property: CHK_CMD → READ_ADD (when MOSI=1, received_address=0, SS_n=0)
27    property chk_cmd_to_read_add;
28      @(posedge clk) disable iff (!rst_n)
29        (cs == CHK_CMD && !SS_n && MOSI && !received_address) |=> (cs == READ_ADD);
30    endproperty
31
32    // Property: CHK_CMD → READ_DATA (when MOSI=1, received_address=1, SS_n=0)
33    property chk_cmd_to_read_data;
34      @(posedge clk) disable iff (!rst_n)
35        (cs == CHK_CMD && !SS_n && MOSI && received_address) |=> (cs == READ_DATA);
36    endproperty
37
38    // Property: CHK_CMD → IDLE (when SS_n goes high)
39    property chk_cmd_to_idle;
40      @(posedge clk) disable iff (!rst_n)
41        (cs == CHK_CMD && SS_n) |=> (cs == IDLE);
42    endproperty
43
44    // Property: WRITE → IDLE (when SS_n goes high)
45    property write_to_idle;
46      @(posedge clk) disable iff (!rst_n)
47        (cs == WRITE && SS_n) |=> (cs == IDLE);
48    endproperty
49
50    // Property: WRITE → WRITE (when SS_n remains low)
51    property write_to_write;
52      @(posedge clk) disable iff (!rst_n)
53        (cs == WRITE && !SS_n) |=> (cs == WRITE);
54    endproperty
55
56    // Property: READ_ADD → IDLE (when SS_n goes high)
57    property read_add_to_idle;
58      @(posedge clk) disable iff (!rst_n)
59        (cs == READ_ADD && SS_n) |=> (cs == IDLE);
60    endproperty
61
62    // Property: READ_ADD → READ_ADD (when SS_n remains low)
63    property read_add_to_read_add;
64      @(posedge clk) disable iff (!rst_n)
65        (cs == READ_ADD && !SS_n) |=> (cs == READ_ADD);
66    endproperty
67
68    // Property: READ_DATA → IDLE (when SS_n goes high)
69    property read_data_to_idle;
70      @(posedge clk) disable iff (!rst_n)
71        (cs == READ_DATA && SS_n) |=> (cs == IDLE);
72    endproperty
73
74    // Property: READ_DATA → READ_DATA (when SS_n remains low)
75    property read_data_to_read_data;
76      @(posedge clk) disable iff (!rst_n)
77        (cs == READ_DATA && !SS_n) |=> (cs == READ_DATA);
78    endproperty
79
80    // Property: Valid states only
81    property valid_states_only;
82      @(posedge clk) disable iff (!rst_n)
83        cs inside {IDLE, WRITE, CHK_CMD, READ_ADD, READ_DATA};
84    endproperty

```



```
1 //=====
2 // Assertions
3 //=====
4 assert_idle_to_chk_cmd: assert property (idle_to_chk_cmd)
5     else $error("FSM Error @ %0t: IDLE did not transition to CHK_CMD when SS_n=0", $time);
6
7 assert_idle_to_idle: assert property (idle_to_idle)
8     else $error("FSM Error @ %0t: IDLE did not stay in IDLE when SS_n=1", $time);
9
10 assert_chk_cmd_to_write: assert property (chk_cmd_to_write)
11     else $error("FSM Error @ %0t: CHK_CMD did not transition to WRITE when MOSI=0", $time);
12
13 assert_chk_cmd_to_read_add: assert property (chk_cmd_to_read_add)
14     else $error("FSM Error @ %0t: CHK_CMD did not transition to READ_ADD", $time);
15
16 assert_chk_cmd_to_read_data: assert property (chk_cmd_to_read_data)
17     else $error("FSM Error @ %0t: CHK_CMD did not transition to READ_DATA", $time);
18
19 assert_chk_cmd_to_idle: assert property (chk_cmd_to_idle)
20     else $error("FSM Error @ %0t: CHK_CMD did not transition to IDLE when SS_n=1", $time);
21
22 assert_write_to_idle: assert property (write_to_idle)
23     else $error("FSM Error @ %0t: WRITE did not transition to IDLE when SS_n=1", $time);
24
25 assert_write_to_write: assert property (write_to_write)
26     else $error("FSM Error @ %0t: WRITE did not stay in WRITE when SS_n=0", $time);
27
28 assert_read_add_to_idle: assert property (read_add_to_idle)
29     else $error("FSM Error @ %0t: READ_ADD did not transition to IDLE when SS_n=1", $time);
30
31 assert_read_add_to_read_add: assert property (read_add_to_read_add)
32     else $error("FSM Error @ %0t: READ_ADD did not stay in READ_ADD when SS_n=0", $time);
33
34 assert_read_data_to_idle: assert property (read_data_to_idle)
35     else $error("FSM Error @ %0t: READ_DATA did not transition to IDLE when SS_n=1", $time);
36
37 assert_read_data_to_read_data: assert property (read_data_to_read_data)
38     else $error("FSM Error @ %0t: READ_DATA did not stay in READ_DATA when SS_n=0", $time);
39
40 assert_valid_states: assert property (valid_states_only)
41     else $error("FSM Error @ %0t: Invalid state detected: cs=%b", $time, cs);
42
43 a_check_reset: assert property (check_reset)
44     else $error("Assertion FAILED: Reset condition not satisfied – outputs not cleared properly.");
45
46 //=====
47 // Coverage for FSM transitions
48 //=====
49 cover_idle_to_chk_cmd: cover property (idle_to_chk_cmd);
50 cover_chk_cmd_to_write: cover property (chk_cmd_to_write);
51 cover_chk_cmd_to_read_add: cover property (chk_cmd_to_read_add);
52 cover_chk_cmd_to_read_data: cover property (chk_cmd_to_read_data);
53 cover_write_to_idle: cover property (write_to_idle);
54 cover_read_add_to_idle: cover property (read_add_to_idle);
55 cover_read_data_to_idle: cover property (read_data_to_idle);
56 c_check_reset: cover property (check_reset);
57
58 `endif
59
60 endmodule
```

Interface :

```
1 interface SLAVE_if (clk);
2     input clk ;
3     logic rst_n ;
4     logic MOSI ;
5     logic SS_n ;
6     logic tx_valid;
7     logic [7:0] tx_data;
8     logic [9:0] rx_data;
9     logic rx_valid ;
10    bit MISO;
11 endinterface
```

Top module:

```
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import SLAVE_test_pkg::*;
4
5 module SLAVE_top();
6     bit clk ,reset ;
7     initial begin
8         forever
9             #1 clk =~clk ;
10    end
11    SLAVE_if inst_if (clk) ;
12    SLAVE inst_DUT (inst_if.MOSI,inst_if.MISO,inst_if.SS_n,inst_if.clk,
13                    .rst_n,inst_if.rx_data,inst_if.rx_valid,inst_if.tx_data,
14                    inst_if.tx_valid);
15
16    initial begin
17        uvm_config_db #(virtual SLAVE_if)::set(null, "uvm_test_top" , "SLAVE_if" ,inst_if) ;
18        run_test("SLAVE_test");
19    end
20 endmodule
```

SLAVE test file:

```
● ● ●
1 package SLAVE_test_pkg;
2 import SLAVE_env_pkg::*;
3 import SLAVE_config_pkg::*;
4 import SLAVE_reset_seq_pkg::*;
5 import SLAVE_seq_pkg::*;
6 import uvm_pkg::*;
7 `include "uvm_macros.svh"
8
9 class SLAVE_test extends uvm_test;
10   `uvm_component_utils(SLAVE_test)
11   SLAVE_env env ;
12   SLAVE_config SLAVE_cfg ;
13   SLAVE_seq seq;
14   SLAVE_seq_reset rst_seq;
15
16   function new (string name = "SLAVE_test" , uvm_component parent = null);
17     super.new (name , parent) ;
18   endfunction
19
20   function void build_phase (uvm_phase phase);
21     super.build_phase(phase) ;
22     env = SLAVE_env::type_id::create("env",this);
23     SLAVE_cfg = SLAVE_config::type_id::create("SLAVE_cfg");
24     seq = SLAVE_seq::type_id::create("seq");
25     rst_seq = SLAVE_seq_reset::type_id::create("rst_seq");
26
27     if (!uvm_config_db #(virtual SLAVE_if)::get(this, "" , "SLAVE_if" , SLAVE_cfg.SLAVE_vif))
28       `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface");
29
30     uvm_config_db #($SLAVE_config)::set(this , "*" , "CFG" , SLAVE_cfg);
31   endfunction
32
33   task run_phase (uvm_phase phase);
34     super.run_phase(phase);
35     phase.raise_objection(this);
36     `uvm_info("run_phase" , "reset asserted" , UVM_MEDIUM)
37     rst_seq.start(env.agt.sqr);
38     `uvm_info("run_phase" , "reset deasserted" , UVM_MEDIUM)
39
40     `uvm_info("run_phase" , "seq stimulus generated started " , UVM_MEDIUM)
41     seq.start(env.agt.sqr);
42     `uvm_info("run_phase" , "seq stimulus generated ended" , UVM_MEDIUM)
43
44     phase.drop_objection(this);
45   endtask: run_phase
46 endclass
47
48 endpackage
```

SLAVE seq_item file:

```
1 package SLAVE_item_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5 class SLAVE_item extends uvm_sequence_item;
6   `uvm_object_utils(SLAVE_item)
7
8   // Randomized inputs
9   rand logic rst_n;
10  rand logic SS_n;
11  rand bit [10:0] MOSI_array;
12
13  // Non-randomized inputs (driven from MOSI_array or constraints)
14  logic MOSI;
15  rand logic tx_valid;
16  rand logic [7:0] tx_data;
17
18  // DUT outputs (monitored)
19  logic [9:0] rx_data;
20  logic rx_valid;
21  logic MISO;
22
23  // State tracking variables
24  logic [4:0] count;           // Counter for SS_n timing (0-22)
25  logic [3:0] bit_index;       // Track which bit of MOSI_array to send (0-10)
26  logic prev_SS_n;           // Previous value of SS_n
27  bit [10:0] stored_MOSI_array; // Store the randomized array
28  bit [2:0] current_cmd;      // Store current command being executed
29
30  function new(string name = "SLAVE_item");
31    super.new(name);
32    prev_SS_n = 1'b1;
33    count = 0;
34    bit_index = 0;
35    stored_MOSI_array = 11'b0000_0000_0000;
36    current_cmd = 3'b000;
37  endfunction
38
39  function string convert2string();
40    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",
41                     super.convert2string(),
42                     rst_n, MOSI, SS_n, tx_valid, tx_data, rx_data, rx_valid, MISO);
43  endfunction
44
45  function string convert2string_stimulus();
46    return $sformatf("rst_n=%b, MOSI=%b, SS_n=%b, tx_valid=%b, tx_data=%h, cmd=%b, count=%b,
47 , bit_idx=%b", rst_n, MOSI, SS_n, tx_valid, tx_data, current_cmd, count, bit_index);
48  endfunction
49
50  // Constraint 1: Reset signal deasserted most of the time
51  constraint reset_c {
52    rst_n dist {1 := 99, 0 := 1};
53  }
54
55  // Constraint 2: SS_n timing based on command type
56  // For WRITE (000) and READ_ADD (001): SS_n high every 13 cycles
57  // For READ_DATA (110, 111): SS_n high every 23 cycles
58  constraint SS_n_timing_c {
59    if (current_cmd inside {3'b000, 3'b001, 3'b110}) {
60      // Write or Read Address: 13 cycle period (12 low, 1 high)
61      SS_n == (count == 12);
62    }
63    else if (current_cmd inside { 3'b111}) {
64      // Read Data: 23 cycle period (22 low, 1 high)
65      SS_n == (count == 22);
66    }
67    else {
68      // Default/initial case
69      SS_n == (count == 12);
70    }
71  }
```

```

1          // Constraint 3: MOSI_array randomization
2          // Only randomize command bits [10:8] when SS_n transitions from 1 to 0
3          // Valid commands: 000 (write), 001 (write), 110 (read addr), 111 (read data)
4          constraint MOSI_array_c {
5              if (prev_SS_n && !SS_n) {
6                  // Falling edge of SS_n - randomize with valid commands
7                  MOSI_array[10:8] inside {3'b000, 3'b001, 3'b110, 3'b111};
8                  // Data bits [7:0] can be any value
9              }
10             else {
11                 // Not a falling edge - keep stored value
12                 MOSI_array == stored_MOSI_array;
13             }
14         }
15
16         // Constraint 4: tx_valid signal
17         // High only for READ_DATA command (111)
18         constraint tx_valid_c {
19             if (current_cmd == 3'b111) {
20                 tx_valid == 1;
21             }
22             else {
23                 tx_valid == 0;
24             }
25         }
26
27         // Constraint: tx_data can be any value
28         // constraint tx_data_c {
29             //     tx_data inside {[8'h00:8'hFF]};
30         //}
31
32         // Post-randomize to handle bit-by-bit MOSI assignment and state updates
33         function void post_randomize();
34             logic is_falling_edge;
35
36             is_falling_edge = (prev_SS_n && !SS_n);
37
38             // Handle SS_n falling edge - start new transaction
39             if (is_falling_edge) begin
40                 stored_MOSI_array = MOSI_array;
41                 current_cmd = MOSI_array[10:8];
42                 bit_index = 0;
43                 count = 0;
44                 MOSI = stored_MOSI_array[10]; // Start with MSB
45
46                 `uvm_info("SLAVE_ITEM", $sformatf(
47                     "New Transaction: cmd=%b, MOSI_array=%b, count=%d",
48                     current_cmd, stored_MOSI_array, count
49                 ), UVM_HIGH)
50             end
51             // Handle SS_n active (low) - continue shifting bits
52             else if (!SS_n) begin
53                 if (bit_index < 11) begin
54                     MOSI = stored_MOSI_array[10 - bit_index];
55                     bit_index++;
56                 end
57                 else begin
58                     MOSI = 1'b0; // Default after all bits sent
59                 end
60             end
61             // Handle SS_n inactive (high) - idle
62             else begin
63                 MOSI = 1'b0;
64                 bit_index = 0;
65             end
66
67             // Update counter for SS_n timing
68             if (current_cmd inside {3'b000, 3'b001, 3'b110}) begin
69                 // 13 cycle period
70                 if (count >= 12) count = 0;
71                 else count++;
72             end
73             else if (current_cmd inside {3'b111}) begin
74                 // 23 cycle period
75                 if (count >= 22) count = 0;
76                 else count++;
77             end
78             else begin
79                 // Default: 13 cycle period
80                 if (count >= 12) count = 0;
81                 else count++;
82             end
83
84             // Update prev_SS_n for next cycle
85             prev_SS_n = SS_n;
86         endfunction
87
88     endclass
89 endpackage

```

SLAVE sequencer file :

```
● ● ●  
1 package SLAVE_sequencer_pkg;  
2     import uvm_pkg::*;  
3     `include "uvm_macros.svh"  
4     import SLAVE_item_pkg::*;  
5     class SLAVE_sequencer extends uvm_sequencer #(SLAVE_item);  
6         `uvm_component_utils(SLAVE_sequencer)  
7  
8         function new(string name = "SLAVE_sequencer" , uvm_component parent = null );  
9             super.new(name, parent);  
10            endfunction  
11        endclass  
12    endpackage
```

SLAVE reset_seq file :

```
● ● ●  
1 package SLAVE_reset_seq_pkg;  
2     import uvm_pkg::*;  
3     `include "uvm_macros.svh"  
4     import SLAVE_item_pkg::*;  
5     class SLAVE_seq_reset extends uvm_sequence #(SLAVE_item);  
6         `uvm_object_utils(SLAVE_seq_reset)  
7         SLAVE_item seq_item ;  
8  
9         function new(string name = "SLAVE_seq_reset");  
10            super.new(name);  
11            endfunction  
12  
13            task body;  
14                seq_item = SLAVE_item ::type_id::create("seq_item");  
15                start_item(seq_item);  
16                seq_item.rst_n = 0 ;  
17                finish_item(seq_item);  
18            endtask  
19        endclass  
20    endpackage
```

SLAVE main_seq file :

```
1 package SLAVE_seq_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_item_pkg::*;
5     class SLAVE_seq extends uvm_sequence #(SLAVE_item);
6         `uvm_object_utils(SLAVE_seq)
7         SLAVE_item seq_item ;
8
9         function new(string name = "SLAVE_seq");
10            super.new(name);
11        endfunction
12
13        task body;
14            seq_item = SLAVE_item ::type_id::create("seq_item");
15            repeat (10000) begin
16                start_item(seq_item);
17                assert(seq_item.randomize);
18                finish_item(seq_item);
19            end
20        endtask
21    endclass
22 endpackage
```

SLAVE configuration file:

```
1 package SLAVE_config_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     class SLAVE_config extends uvm_object;
5         `uvm_object_utils(SLAVE_config)
6         virtual SLAVE_if SLAVE_vif ;
7         function new(string name = "SLAVE_config");
8             super.new(name) ;
9         endfunction
10    endclass
11 endpackage
```

SLAVE env file :

```
1 package SLAVE_env_pkg;
2   import SLAVE_scoreboard_pkg::*;
3   import SLAVE_agent_pkg::*;
4   import SLAVE_coverage_pkg::*;
5   import uvm_pkg::*;
6   `include "uvm_macros.svh"
7
8 class SLAVE_env extends uvm_env;
9
10  `uvm_component_utils(SLAVE_env)
11    SLAVE_agent agt ;
12    SLAVE_coverage cov ;
13    SLAVE_scoreboard sb ;
14
15  function new (string name = "SLAVE_env" , uvm_component parent = null);
16    super.new(name , parent) ;
17  endfunction
18
19  function void build_phase (uvm_phase phase);
20    super.build_phase(phase) ;
21    agt = SLAVE_agent::type_id::create("agt",this);
22    cov = SLAVE_coverage::type_id::create("cov",this);
23    sb = SLAVE_scoreboard::type_id::create("sb",this);
24  endfunction
25
26  function void connect_phase (uvm_phase phase);
27    super.connect_phase(phase) ;
28    agt.agt_ap.connect(sb.sb_export);
29    agt.agt_ap.connect(cov.cov_export);
30  endfunction
31 endclass
32 endpackage
```

SLAVE agent file :

```
1 package SLAVE_agent_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_sequencer_pkg::*;
5     import SLAVE_driver_pkg::*;
6     import SLAVE_config_pkg::*;
7     import SLAVE_monitor_pkg::*;
8     import SLAVE_item_pkg::*;
9
10    class SLAVE_agent extends uvm_agent;
11        `uvm_component_utils(SLAVE_agent)
12        SLAVE_sequencer sqr ;
13        SLAVE_driver drv ;
14        SLAVE_monitor mon;
15        SLAVE_config SLAVE_cfg;
16        uvm_analysis_port #(SLAVE_item) agt_ap;
17
18        function new(string name = "SLAVE_agent" , uvm_component parent = null );
19            super.new(name, parent);
20        endfunction
21
22        function void build_phase (uvm_phase phase);
23            super.build_phase(phase);
24            if (!uvm_config_db #(SLAVE_config)::get(this, "", "CFG", SLAVE_cfg)) begin
25                `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
26            end
27            sqr = SLAVE_sequencer::type_id::create("sqr",this);
28            drv = SLAVE_driver::type_id::create("drv",this);
29            mon = SLAVE_monitor::type_id::create("mon",this);
30            agt_ap = new("agt_ap",this);
31        endfunction
32
33        function void connect_phase(uvm_phase phase);
34            super.connect_phase(phase);
35            drv.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
36            mon.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
37            drv.seq_item_port.connect(sqr.seq_item_export);
38            mon.mon_ap.connect(agt_ap);
39        endfunction
40    endclass
41 endpackage
```

SLAVE driver file :

```
1 package SLAVE_driver_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_item_pkg::*;
5
6     class SLAVE_driver extends uvm_driver #(SLAVE_item);
7         `uvm_component_utils(SLAVE_driver)
8         virtual SLAVE_if SLAVE_vif;
9         SLAVE_item stim_seq_item;
10
11     function new (string name = "SLAVE_driver", uvm_component parent = null);
12         super.new(name, parent);
13     endfunction
14
15     task run_phase(uvm_phase phase);
16         super.run_phase(phase);
17         forever begin
18             stim_seq_item = SLAVE_item::type_id::create("stim_seq_item");
19             seq_item_port.get_next_item(stim_seq_item);
20             SLAVE_vif.rst_n = stim_seq_item.rst_n ;
21             SLAVE_vif.MOSI = stim_seq_item.MOSI;
22             SLAVE_vif.SS_n = stim_seq_item.SS_n;
23             SLAVE_vif.tx_valid = stim_seq_item.tx_valid;
24             SLAVE_vif.tx_data = stim_seq_item.tx_data;
25             @(negedge SLAVE_vif.clk);
26             seq_item_port.item_done();
27         end
28     endtask
29 endclass
30 endpackage
```

SLAVE monitor file :

```
● ○ ●
1 package SLAVE_monitor_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_item_pkg::*;
5
6     class SLAVE_monitor extends uvm_monitor;
7         `uvm_component_utils(SLAVE_monitor)
8         virtual SLAVE_if SLAVE_vif;
9         SLAVE_item rsp_seq_item;
10        uvm_analysis_port #(SLAVE_item) mon_ap;
11
12        function new (string name = "SLAVE_monitor", uvm_component parent = null);
13            super.new(name, parent);
14        endfunction
15
16        function void build_phase(uvm_phase phase);
17            super.build_phase(phase);
18            mon_ap = new("mon_ap",this);
19        endfunction
20
21        task run_phase(uvm_phase phase);
22            super.run_phase(phase);
23            forever begin
24                rsp_seq_item = SLAVE_item::type_id::create("rsp_seq_item");
25                rsp_seq_item.rst_n = SLAVE_vif.rst_n ;
26                rsp_seq_item.SS_n = SLAVE_vif.SS_n ;
27                rsp_seq_item.MOSI = SLAVE_vif.MOSI ;
28                rsp_seq_item.tx_valid = SLAVE_vif.tx_valid ;
29                rsp_seq_item.tx_data = SLAVE_vif.tx_data ;
30                rsp_seq_item.rx_data = SLAVE_vif.rx_data ;
31                rsp_seq_item.rx_valid = SLAVE_vif.rx_valid ;
32                rsp_seq_item.MISO = SLAVE_vif.MISO ;
33                @(negedge SLAVE_vif.clk);
34                mon_ap.write(rsp_seq_item);
35            end
36        endtask
37    endclass
38 endpackage
```

SLAVE scoreboard file :

```
1 package SLAVE_scoreboard_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_item_pkg::*;
5
6     class SLAVE_scoreboard extends uvm_scoreboard;
7         `uvm_component_utils(SLAVE_scoreboard)
8         uvm_analysis_export #(SLAVE_item) sb_export;
9         uvm_tlm_analysis_fifo #(SLAVE_item) sb_fifo;
10        SLAVE_item seq_item_sb;
11        logic [9:0] rx_data_ref;
12        logic rx_valid_ref ;
13        bit MISO_ref ;
14
15        localparam IDLE      = 3'b000;
16        localparam WRITE     = 3'b001;
17        localparam CHK_CMD   = 3'b010;
18        localparam READ_ADD  = 3'b011;
19        localparam READ_DATA = 3'b100;
20        logic [2:0] cs, ns;
21        logic received_address ;
22        logic [3:0] counter;
23
24        int error_count    = 0 ;
25        int correct_count = 0 ;
26
27        function new (string name = "SLAVE_scoreboard" , uvm_component parent = null);
28            super.new(name , parent) ;
29        endfunction
30
31        function void build_phase (uvm_phase phase);
32            super.build_phase(phase);
33            sb_export = new("sb_export",this);
34            sb_fifo   = new("sb_fifo",this);
35        endfunction
36
37        function void connect_phase(uvm_phase phase);
38            super.connect_phase(phase);
39            sb_export.connect(sb_fifo.analysis_export);
40        endfunction
41
42        function void report_phase(uvm_phase phase);
43            super.report_phase(phase);
44            `uvm_info("SCOREBOARD", $sformatf(
45                "Simulation Summary: correct_count=%d
46 , error_count), UVM_NONE)
47        endfunction
48
49        task run_phase (uvm_phase phase);
50            super.run_phase(phase);
51            forever begin
52                sb_fifo.get(seq_item_sb);
53                ref_seq_item_chk_model(seq_item_sb);
54                if (rx_data_ref === seq_item_sb.rx_data && rx_valid_ref === seq_item_sb.rx_valid && seq_item_sb.MISO === MISO_ref) begin
55                    correct_count++;
56                end
57                else begin
58                    `uvm_error("SCOREBOARD", $sformatf("Mismatch: DUT_data=%0h REF_data=%0h ||
59                               DUT_rx_valid=%b REF_rx_valid=%b ||
60                               DUT_MISO=%b REF_MISO=%b"
61                               , seq_item_sb.rx_data, rx_data_ref , seq_item_sb.rx_valid ,
62                               rx_valid_ref , seq_item_sb.MISO , MISO_ref));
63                    error_count++;
64                end
65            end
66        endtask : run_phase
```

```

1      task ref_seq_item_chk_model(SLAVE_item seq_item_chk);
2          if (!seq_item_chk.rst_n) begin
3              cs = IDLE ;
4              rx_data_ref = 0 ;
5              rx_valid_ref = 0 ;
6              MISO_ref = 0 ;
7              counter = 0 ;
8              received_address = 0 ;
9          end
10         else begin
11             case (cs)
12                 IDLE: begin
13                     rx_valid_ref = 0 ;
14                     if (seq_item_chk.ss_n) ns = IDLE ;
15                     else ns = CHK_CMD ;
16                 end
17                 CHK_CMD : begin
18                     if (seq_item_chk.ss_n) ns = IDLE ;
19                     else begin
20                         if (!seq_item_chk.MOSI) ns = WRITE ;
21                         else begin
22                             if (received_address) ns = READ_DATA ;
23                             else ns = READ_ADD ;
24                         end
25                     end
26                     counter = 10 ;
27                 end
28                 WRITE : begin
29                     if (seq_item_chk.ss_n) ns = IDLE ;
30                     else ns = WRITE ;
31                     if (counter > 0) begin
32                         rx_data_ref[counter-1] = seq_item_chk.MOSI;
33                         counter = counter - 1 ;
34                     end
35                     else rx_valid_ref = 1 ;
36                 end
37                 READ_ADD : begin
38                     if (seq_item_chk.ss_n) ns = IDLE ;
39                     else ns = READ_ADD;
40                     if (counter > 0) begin
41                         rx_data_ref[counter-1] = seq_item_chk.MOSI;
42                         counter = counter - 1 ;
43                     end
44                     else begin
45                         rx_valid_ref = 1 ;
46                         received_address = 1 ;
47                     end
48                 end
49                 READ_DATA : begin
50                     if (seq_item_chk.ss_n) ns = IDLE ;
51                     else ns = READ_DATA ;
52                     if (seq_item_chk.tx_valid) begin
53                         rx_valid_ref = 0 ;
54                         if (counter > 0) begin
55                             MISO_ref = seq_item_chk.tx_data[counter-1];
56                             counter = counter - 1 ;
57                         end
58                         else received_address = 0 ;
59                     end
60                     else begin
61                         if (counter > 0) begin
62                             rx_data_ref[counter-1] = seq_item_chk.MOSI;
63                             counter = counter - 1 ;
64                         end
65                         else begin
66                             rx_valid_ref = 1 ;
67                             counter = 8 ;
68                         end
69                     end
70                 end
71             endcase
72             cs = ns;
73         end
74     endtask
75 endclass
76
77 endpackage

```

SLAVE coverage file :

```
1 package SLAVE_coverage_pkg;
2   import uvm_pkg::*;
3   import SLAVE_item_pkg::*;
4   `include "uvm_macros.svh"
5
6   class SLAVE_coverage extends uvm_component;
7     `uvm_component_utils(SLAVE_coverage)
8     uvm_analysis_export #(SLAVE_item) cov_export;
9     uvm_tlm_analysis_fifo #(SLAVE_item) cov_fifo;
10    SLAVE_item seq_item_cov;
11
12   covergroup covcode;
13     A_cp : coverpoint seq_item_cov.rx_data[9:8] {
14       bins rx_00 = {00};
15       bins rx_01 = {01};
16       bins rx_10 = {10};
17       bins rx_11 = {11};
18     }
19
20   SS_n: coverpoint seq_item_cov.SS_n {
21     bins normal_transaction = (1 => 0 [*12] => 1);
22     bins extended_transaction = (1 => 0 [*22] => 1);
23     bins communication_on = {0} ;
24   }
25
26   MOSI: coverpoint seq_item_cov.MOSI {
27     bins write_address = (0 => 0 => 0);
28     bins write_DATA = (0 => 0 => 1);
29     bins read_ADDRESS = (1 => 1 => 0);
30     bins read_DATA = (1 => 1 => 1);
31   }
32
33   cross SS_n , MOSI {
34     option.cross_auto_bin_max = 0;
35     bins write_add_normal = binsof (MOSI.write_address) && binsof (SS_n.communication_on);
36     bins write_data_normal = binsof (MOSI.write_DATA) && binsof (SS_n.communication_on);
37     bins read_add_normal = binsof (MOSI.read_ADDRESS) && binsof (SS_n.communication_on);
38     bins read_data_extended = binsof (MOSI.read_DATA) && binsof (SS_n.communication_on);
39   }
40 endgroup
41
42 function new (string name = "SLAVE_coverage" , uvm_component parent = null);
43   super.new(name, parent);
44   covcode = new();
45 endfunction
46
47 function void build_phase (uvm_phase phase);
48   super.build_phase(phase);
49   cov_export = new("cov_export",this);
50   cov_fifo = new("cov_fifo",this);
51 endfunction
52
53 function void connect_phase(uvm_phase phase);
54   super.connect_phase(phase);
55   cov_export.connect(cov_fifo.analysis_export);
56 endfunction
57
58 task run_phase (uvm_phase phase);
59   super.run_phase(phase);
60   forever begin
61     cov_fifo.get(seq_item_cov);
62     covcode.sample();
63   end
64 endtask
65 endclass
66 endpackage
```

Do file:

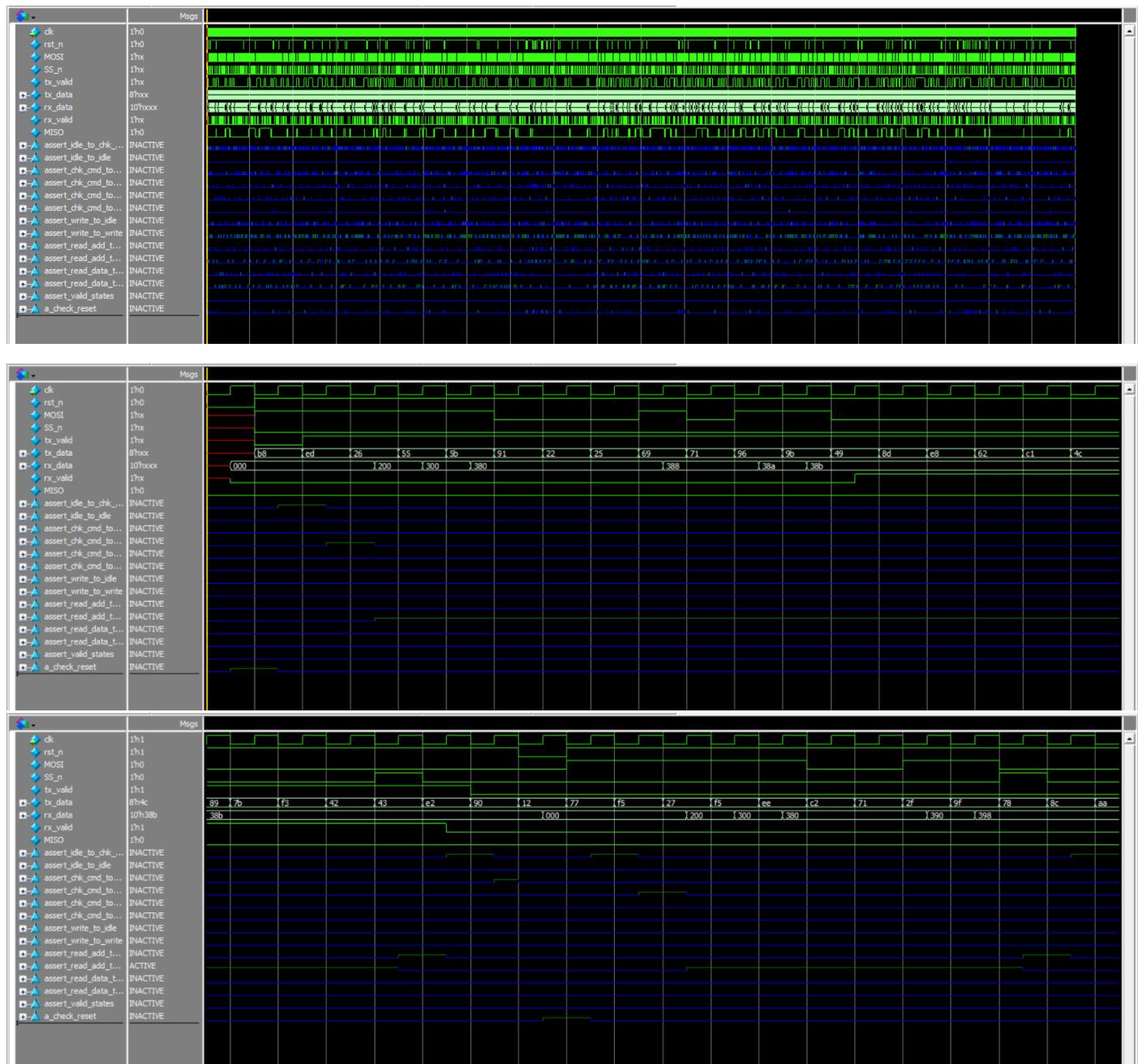
```
1 vlib work
2 vlog +define+SIM +cover *.sv
3 vsim -voptargs+=acc work.SLAVE_top -classdebug -uvmcontrol=all
4 add wave -position insertpoint sim:/SLAVE_top/inst_if/*
5 add wave /SLAVE_top/inst_DUT/assert_idle_to_chk_cmd /
6 SLAVE_top/inst_DUT/assert_idle_to_idle /SLAVE_top/
7 inst_DUT/assert_chk_cmd_to_write /SLAVE_top/inst_DUT/
8 assert_chk_cmd_to_read_add /SLAVE_top/inst_DUT/
9 assert_chk_cmd_to_read_data /SLAVE_top/inst_DUT/
10 assert_chk_cmd_to_idle /SLAVE_top/inst_DUT/assert_write_to_idle /SLAVE_top
11 /inst_DUT/assert_write_to_write /SLAVE_top/inst_DUT/assert_read_add_to_idle
12 /SLAVE_top/inst_DUT/assert_read_add_to_read_add /SLAVE_top/inst_DUT/
13 assert_read_data_to_idle /SLAVE_top/inst_DUT/assert_read_data_to_read_data
14 /SLAVE_top/inst_DUT/assert_valid_states /SLAVE_top/inst_DUT/a_check_reset
15 coverage save -instance /SLAVE_top/inst_DUT SLAVE.ucdb -onexit
16 run -all
```

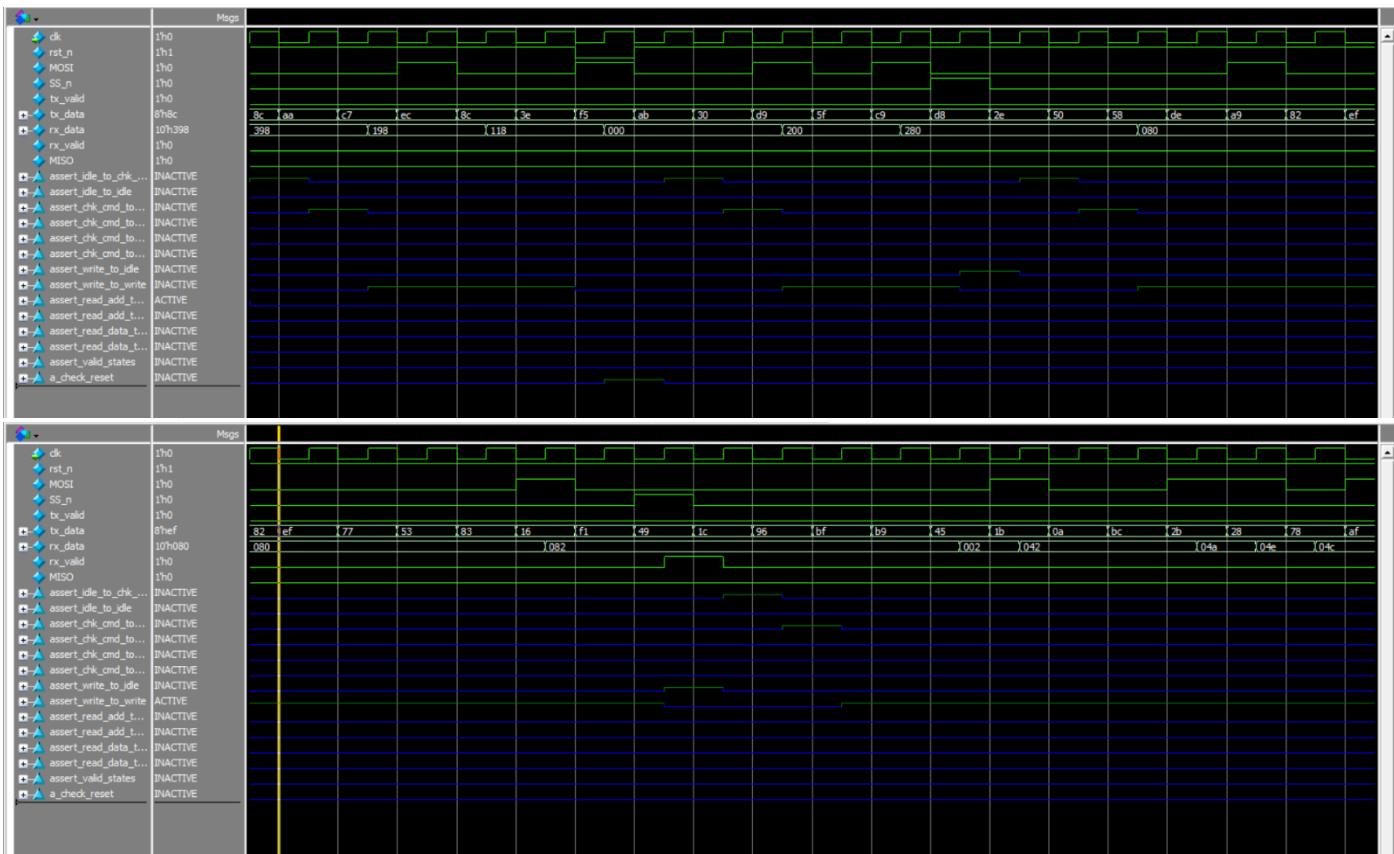
Second: Snippets from Simulation results:

Transcript:

```
# UVM_INFO Verilog_SVC/uvm-1.1d/SVC/base/uvm_objection.svh(126) @ 20002: reported [TEST_DONE] run phase is ready to proceed to the extract phase
# UVM_INFO SLAVE_scoreboard.sv(44) @ 20002: uvm_test_top.env.sv [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	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_chk_cmd_to_write	SVA	✓	Off	381	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_chk_cmd_to_read_add	SVA	✓	Off	179	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_chk_cmd_to_read_data	SVA	✓	Off	154	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_write_to_idle	SVA	✓	Off	341	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_read_add_to_idle	SVA	✓	Off	144	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/cover_read_data_to_idle	SVA	✓	Off	134	1	Unl...	1	100%		✓	0	0	0 ns	0
+/SLAVE_top/inst_DUT/c_check_reset	SVA	✓	Off	114	1	Unl...	1	100%		✓	0	0	0 ns	0
<hr/>														
+/SLAVE_seq_pkg::SLAVE_seq::tbody#/ublk#29747843#15/immmed_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)=	✓		

```

# Assertion Coverage:
#   Assertions          14      14      0  100.00%
# -----
#   Name        File(Line)      Failure Count      Pass 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%	100	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 VSLAVE_coverage_pkg::SLAVE_coverage::c...			100.00%	100	100.00...		✓				0
CVP A_cp			100.00%	100	100.00...		✓				
B bin auto[0]			3065	1	100.00...		✓				
B bin auto[1]			2586	1	100.00...		✓				
B bin auto[2]			2530	1	100.00...		✓				
B bin auto[3]			1819	1	100.00...		✓				
CVP SS_n			100.00%	100	100.00...		✓				
B bin normal_transaction			482	1	100.00...		✓				
B bin extended_transaction			161	1	100.00...		✓				
B bin communication_on			9355	1	100.00...		✓				
CVP MOSI			100.00%	100	100.00...		✓				
B bin write_address			3395	1	100.00...		✓				
B bin write_DATA			1024	1	100.00...		✓				
B bin read_ADDRESS			962	1	100.00...		✓				
B bin read_DATA			1140	1	100.00...		✓				
D CROSS #cross_0#			100.00%	100	100.00...		✓				
B bin write_addr_normal			3125	1	100.00...		✓				
B bin write_data_normal			1024	1	100.00...		✓				
B bin read_addr_normal			829	1	100.00...		✓				
B 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:

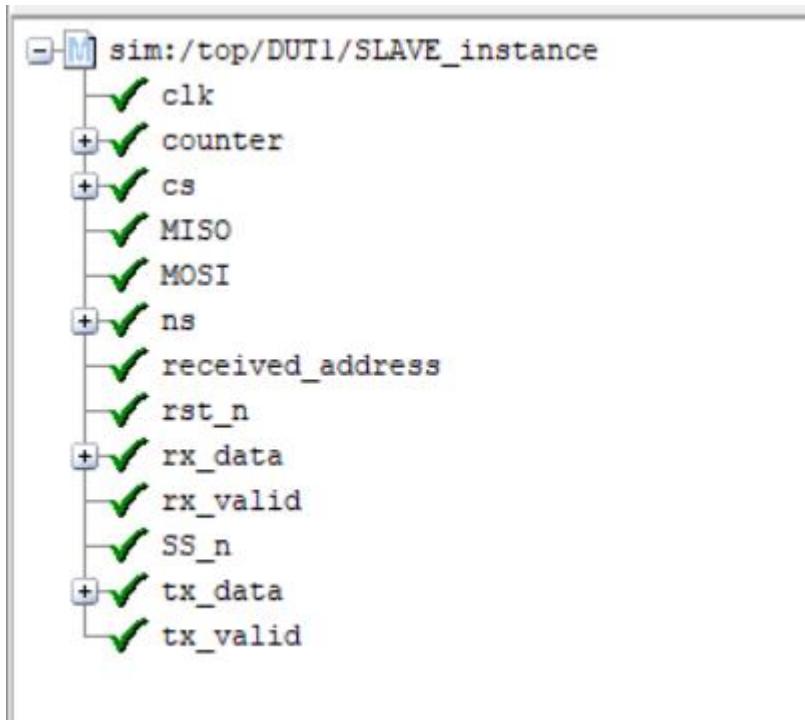
The screenshot shows a code editor window with the file "SPI_slave.sv" open. The code is a SystemVerilog module for a SPI slave. The entire code is covered with green checkmarks, indicating 100% statement coverage. The code includes logic for handling CS, NS, and MISO/MOSI signals, as well as a counter for receiving data.

```
20 always @ (posedge clk) begin
22   cs <= IDLE;
25   cs <= ns;
29   always @ (*) begin
33     ns = IDLE;
35     ns = CHK_CMD;
42     ns = WRITE;
45     ns = READ_DATA;
47     ns = READ_ADD;
53     ns = IDLE;
55     ns = WRITE;
59     ns = IDLE;
61     ns = READ_ADD;
65     ns = IDLE;
67     ns = READ_DATA;
72   always @ (posedge clk) begin
74     rx_data <= 0;
75     rx_valid <= 0;
76     received_address <= 0;
77     MISO <= 0;
78     counter <= 0 ;
83     rx_valid <= 0;
86     counter <= 10;
90     rx_data[counter-1] <= MOSI;
91     counter <= counter - 1;
94     rx_valid <= 1;
99     rx_data[counter-1] <= MOSI;
100    counter <= counter - 1;
103    rx_valid <= 1;
104    received_address <= 1;
109    rx_valid <= 0;
111    MISO <= tx_data[counter-1];
112    counter <= counter - 1;
115    received_address <= 0;
120    rx_data[counter-1] <= MOSI;
121    counter <= counter - 1;
122    rx_valid <= 0 ;
125    rx_valid <= 1;
126    counter <= 9;
```

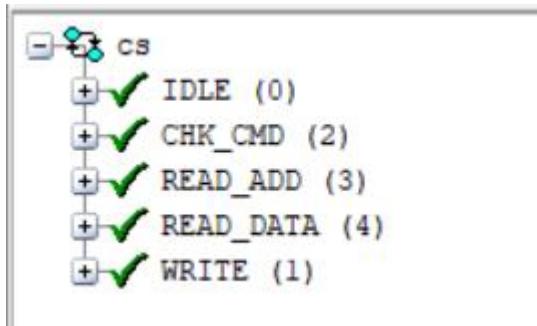
-Branches:

```
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

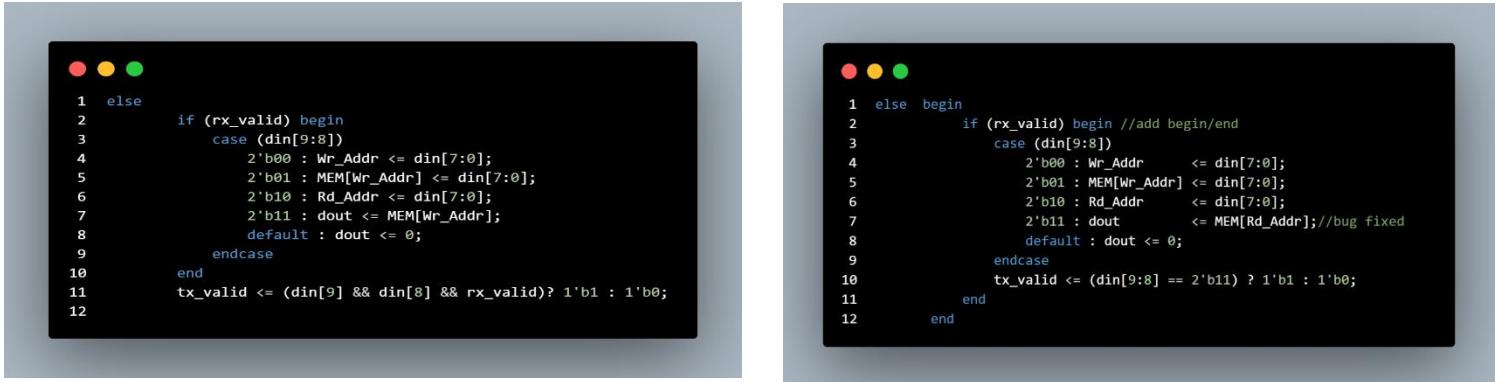
Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
SLAVE_0	Reset Behavior: When the active-low reset (rst_n) is asserted, All output signals reset to 0.	Randomized with constraint that drive the reset to be off most of the simulation time.	-	A checker in the scoreboard to make sure the output is correct and concurrent 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 constraints 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 reference 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 reference 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 reference model in the scoreboard and SVA inside SVA file to check internal signals

RAM part

First: Snippets from codes:

Design and bugs found:

- Bug



```
1 else begin
2     if (rx_valid) begin
3         case (din[9:8])
4             2'b00 : Wr_Addr <= din[7:0];
5             2'b01 : MEM[Wr_Addr] <= din[7:0];
6             2'b10 : Rd_Addr <= din[7:0];
7             2'b11 : dout <= MEM[Rd_Addr];
8             default : dout <= 0;
9         endcase
10    end
11    tx_valid <= (din[9] && din[8] && rx_valid)? 1'b1 : 1'b0;
12
```

```
1 else begin
2     if (rx_valid) begin //add begin/end
3         case (din[9:8])
4             2'b00 : Wr_Addr      <= din[7:0];
5             2'b01 : MEM[Wr_Addr] <= din[7:0];
6             2'b10 : Rd_Addr      <= din[7:0];
7             2'b11 : dout        <= MEM[Rd_Addr];//bug fixed
8             default : dout <= 0;
9         endcase
10        tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;
11    end
12
```

- **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 :



```
1 module RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3     input [9:0] din;
4     input clk, rst_n, rx_valid;
5
6     output reg [7:0] dout;
7     output reg tx_valid;
8
9     reg [7:0] MEM [255:0];
10    reg [7:0] Rd_Addr, Wr_Addr;
11
12    integer i ;
13    always @(posedge clk) begin
14        if (~rst_n) begin
15            dout      <= 0;
16            tx_valid <= 0;
17            Rd_Addr  <= 0;
18            Wr_Addr  <= 0;
19        end
20        else begin
21            if (rx_valid) begin //add begin/end
22                case (din[9:8])
23                    2'b00 : Wr_Addr      <= din[7:0];
24                    2'b01 : MEM[Wr_Addr] <= din[7:0];
25                    2'b10 : Rd_Addr      <= din[7:0];
26                    2'b11 : dout        <= MEM[Rd_Addr];//bug fixed
27                    default : dout <= 0;
28                endcase
29                tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;
30            end
31        end
32    end
33 endmodule
```

RAM golden model :

```
● ● ●
1  module RAM_golden (din,clk,rst_n,rx_valid,dout_ref,tx_valid_ref);
2
3      parameter MEM_DEPTH = 256;
4      parameter ADDR_SIZE = 8;
5
6      input  [9:0] din;
7      input  clk, rst_n, rx_valid;
8
9      output reg [7:0] dout_ref;
10     output reg tx_valid_ref;
11
12    reg [ADDR_SIZE-1 : 0] addr_rd, addr_wr;
13    reg [7:0] mem [MEM_DEPTH-1 : 0];
14
15    integer i;
16    always @(posedge clk ) begin
17        if (!rst_n) begin
18            dout_ref     <= 0 ;
19            tx_valid_ref <= 0 ;
20            addr_rd    <= 0 ;
21            addr_wr    <= 0 ;
22        end
23        else begin
24            if (rx_valid) begin
25                case (din[9:8])
26                    2'b00 : begin
27                        addr_wr      <= din[7:0] ;
28                        tx_valid_ref <= 1'b0 ;
29                    end
30                    2'b01 : begin
31                        mem[addr_wr]  <= din[7:0] ;
32                        tx_valid_ref <= 1'b0 ;
33                    end
34                    2'b10 : begin
35                        addr_rd      <= din[7:0] ;
36                        tx_valid_ref <= 1'b0 ;
37                    end
38                    2'b11 : begin
39                        dout_ref     <= mem[addr_rd] ;
40                        tx_valid_ref <= 1'b1 ;
41                    end
42                    default: dout_ref <= 0;
43                endcase
44            end
45        end
46    end
47 endmodule
```

RAM assertion file :

```
 1 module RAM_sva(
 2     input [9:0] din,
 3     input clk, rst_n, rx_valid,
 4     input [7:0] dout,
 5     input tx_valid
 6 );
 7
 8     // 1) reset
 9     property p_reset;
10         @ (posedge clk)
11             (!rst_n) |=> (dout == 0 && tx_valid == 0);
12     endproperty
13     a_reset: assert property (p_reset)
14         else $error("Assertion 1 failed");
15     c_reset: cover property (p_reset);
16
17     // 2) tx_valid is low
18     property p_tx_low;
19         @ (posedge clk) disable iff (!rst_n)
20             ( (!din[9]) && (!din[8]) && rx_valid) |=> (tx_valid == 0);
21     endproperty
22     a_tx_low: assert property (p_tx_low)
23         else $error("Assertion 2 failed");
24     c_tx_low: cover property (p_tx_low);
25
26     // 3) tx_valid rises then falls
27     property p_tx_high;
28         @ (posedge clk) disable iff (!rst_n)
29             ((din[8] && din[9]) && rx_valid) |=> (tx_valid == 1) |=> $fell(tx_valid)[->1] ;
30     endproperty
31     a_tx_high: assert property (p_tx_high)
32     else $error("Assertion 3 failed");
33     c_tx_high: cover property (p_tx_high);
34
35     // 4) write address => write data
36     property p_write;
37         @ (posedge clk) disable iff (!rst_n)
38             (rx_valid && din[9:8] == 2'b00) |=>##1 (din[9:8] inside {2'b00,2'b01});
39     endproperty
40     a_write: assert property (p_write) else $error("Assertion write failed");
41     c_write: cover property (p_write);
42
43     // 5) read address => read data
44     property p_read;
45         @ (posedge clk) disable iff (!rst_n)
46             (din[9:8] == 2'b10) |=>##[0:2] ( din[9:8] inside {2'b10,2'b11});
47     endproperty
48     a_read: assert property (p_read) else $error("Assertion read failed ");
49     c_read: cover property (p_read);
50
51 endmodule
```

RAM Interface :

```
● ● ●  
1 interface RAM_intf (input logic clk);  
2     parameter MEM_DEPTH = 256;  
3     parameter ADDR_SIZE = 8;  
4  
5     // Inputs  
6     logic rst_n;  
7     logic [9:0] din;  
8     logic rx_valid;  
9  
10    // Outputs DUT  
11    logic tx_valid;  
12    logic [7:0] dout;  
13  
14    // Outputs Golden Model  
15    logic tx_valid_ref;  
16    logic [7:0] dout_ref;  
17  
18    // Modports  
19    modport SVA (  
20        input clk, rst_n, din, rx_valid,  
21        input dout, tx_valid  
22    );  
23    endinterface : RAM_intf  
24
```

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_if 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_if)::set(null,"","RAM_IF", RAM_if) ;
42     run_test ("RAM_test") ;
43   end
44 endmodule
45
```

RAM test file:

```
1 package RAM_test_pkg;
2 import RAM_env_pkg::*;
3 import seq_item_pkg::*;
4 import RAM_sequencer_pkg::*;
5 import scoreboard_pkg::*;
6 import RAM_read_only_seq_pkg::*;
7 import RAM_write_only_seq_pkg::*;
8 import RAM_read_write_seq_pkg::*;
9 import RAM_reset_seq_pkg::*;
10 import pkg_cfg::*;
11 import uvm_pkg::*;
12 `include "uvm_macros.svh"
13
14 class RAM_test extends uvm_test;
15   `uvm_component_utils(RAM_test)
16
17   RAM_config_obj RAM_cfg ;
18   virtual RAM_if RAM_vif ;
19   RAM_env env ;
20   RAM_read_only_seq read_only_seq ;
21   RAM_write_only_seq write_only_seq ;
22   RAM_read_write_seq read_write_seq ;
23   RAM_reset_seq reset_seq ;
24
25   function new (string name = "RAM_test", uvm_component parent = null);
26     super.new(name, parent) ;
27   endfunction
28
29   function void build_phase(uvm_phase phase);
30     super.build_phase(phase);
31     env          = RAM_env          :: type_id :: create ("env",this) ;
32     RAM_cfg      = RAM_config_obj   :: type_id :: create("RAM_cfg");
33     reset_seq    = RAM_reset_seq    :: type_id :: create("reset_seq",this);
34     read_only_seq = RAM_read_only_seq :: type_id :: create("read_only_seq",this);
35     write_only_seq = RAM_write_only_seq :: type_id :: create("write_only_seq",this);
36     read_write_seq = RAM_read_write_seq :: type_id :: create("read_only_seq",this);
37
38     if (!uvm_config_db #(virtual RAM_if)::get(this, "", "RAM_IF", RAM_cfg.RAM_config_vif))begin
39       `uvm_fatal("build_phase", "test -unable to get the virtual interface");
40     end
41     uvm_config_db #(RAM_config_obj)::set(this,"*", "CFG", RAM_cfg);
42   endfunction
43
44   task run_phase(uvm_phase phase);
45     super.run_phase(phase) ;
46     phase.raise_objection(this) ;
47
48     //reset sequence
49     `uvm_info("run_phase", "reset_asserted",UVM_LOW)
50     reset_seq.start(env.agt.sqr);
51     `uvm_info("run_phase", "reset_deasserted",UVM_LOW)
52
53     // create and start write only sequence
54     `uvm_info("run_phase", "Starting write only sequence...", UVM_MEDIUM);
55     write_only_seq.start(env.agt.sqr);
56     `uvm_info("run_phase", "write only sequence finished.", UVM_MEDIUM);
57
58     // create and start read/write sequence
59     `uvm_info("run_phase", "Starting read/write sequence...", UVM_MEDIUM);
60     read_write_seq.start(env.agt.sqr);
61     `uvm_info("run_phase", "read/write sequence finished.", UVM_MEDIUM);
62
63     // create and start read only sequence
64     `uvm_info("run_phase", "Starting read only sequence...", UVM_MEDIUM);
65     read_only_seq.start(env.agt.sqr);
66     `uvm_info("run_phase", "read only sequence finished.", UVM_MEDIUM);
67
68     phase.drop_objection(this);
69   endtask : run_phase
70 endclass: RAM_test
71 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 :

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

RAM reset_seq file :

```
● ● ●  
1 package RAM_reset_seq_pkg ;  
2     import seq_item_pkg::*;  
3     import uvm_pkg::*;  
4     `include "uvm_macros.svh"  
5  
6     class RAM_reset_seq extends uvm_sequence #(RAM_seq_item) ;  
7         `uvm_object_utils(RAM_reset_seq)  
8  
9         RAM_seq_item seq_item ;  
10  
11         function new (string name = "RAM_reset_seq");  
12             super.new(name);  
13         endfunction  
14  
15         task body();  
16             seq_item = RAM_seq_item::type_id::create("seq_item");  
17             start_item(seq_item);  
18  
19             seq_item.constraint_mode(0);  
20             assert(seq_item.randomize() with { seq_item.rst_n == 0; seq_item.din[9:8] == 2'b00;});  
21  
22             finish_item(seq_item);  
23         endtask  
24     endclass  
25 endpackage
```

RAM write_seq file :

```
1 package RAM_write_only_seq_pkg ;
2     import seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class RAM_write_only_seq extends uvm_sequence #(RAM_seq_item) ;
7         `uvm_object_utils(RAM_write_only_seq)
8
9         RAM_seq_item seq_item ;
10
11        function new (string name = "RAM_write_only_seq");
12            super.new(name);
13        endfunction
14
15        task body();
16            seq_item = RAM_seq_item::type_id::create("seq_item");
17            repeat(10000) begin
18                start_item(seq_item);
19
20                seq_item.constraint_mode(0);
21                seq_item.c_rx.constraint_mode(1);
22                seq_item.c_RST.constraint_mode(1);
23                seq_item.c_WRITE_ONLY.constraint_mode(1);
24                assert (seq_item.randomize());
25
26                finish_item(seq_item);
27            end
28        endtask
29    endclass
30 endpackage
```

RAM read_seq file :

```
1 package RAM_read_only_seq_pkg ;
2     import seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class RAM_read_only_seq extends uvm_sequence #(RAM_seq_item) ;
7     `uvm_object_utils(RAM_read_only_seq)
8
9     RAM_seq_item seq_item ;
10
11    function new (string name = "RAM_read_only_seq");
12        super.new(name);
13    endfunction
14
15    task body();
16        seq_item = RAM_seq_item::type_id::create("seq_item");
17        repeat(1000) begin
18            start_item(seq_item);
19            seq_item.constraint_mode(0);
20            seq_item.c_rx.constraint_mode(1);
21            seq_item.c_RST.constraint_mode(1);
22            seq_item.c_READ_ONLY.constraint_mode(1);
23            assert (seq_item.randomize());
24            finish_item(seq_item);
25        end
26    endtask
27 endclass
28 endpackage
```

RAM read_write_seq file :

```
1 package RAM_read_write_seq_pkg ;
2     import seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class RAM_read_write_seq extends uvm_sequence #(RAM_seq_item) ;
7     `uvm_object_utils(RAM_read_write_seq)
8
9     RAM_seq_item seq_item ;
10
11    function new (string name = "RAM_read_write_seq");
12        super.new(name);
13    endfunction
14
15    task body();
16        seq_item = RAM_seq_item::type_id::create("seq_item");
17        repeat(1000) begin
18            start_item(seq_item);
19            seq_item.constraint_mode(0);
20            seq_item.c_rx.constraint_mode(1);
21            seq_item.c_rst.constraint_mode(1);
22            seq_item.c_read_write.constraint_mode(1);
23            assert (seq_item.randomize());
24            finish_item(seq_item);
25        end
26    endtask
27 endclass
28 endpackage
```

RAM configuration file:

```
1 package pkg_cfg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4
5     class RAM_config_obj extends uvm_object;
6     `uvm_object_utils(RAM_config_obj)
7
8     virtual RAM_ifc RAM_config_vif ;
9
10    function new (string name = "RAM_config_obj");
11        super.new(name);
12    endfunction
13 endclass
14 endpackage
```

RAM env file :

```
1 package RAM_env_pkg;
2   import scoreboard_pkg::*;
3   import agent_pkg::*;
4   import RAM_coverage_pkg::*;
5   import RAM_sequencer_pkg::*;
6   import uvm_pkg::*;
7   `include "uvm_macros.svh"
8
9 class RAM_env extends uvm_env;
10   `uvm_component_utils(RAM_env)
11
12   RAM_agent agt ;
13   RAM_coverage cov ;
14   RAM_scoreboard sb ;
15
16   function new (string name = "RAM_env", uvm_component parent = null);
17     super.new(name, parent) ;
18   endfunction
19
20   function void build_phase(uvm_phase phase);
21     super.build_phase(phase);
22     agt = RAM_agent    :: type_id :: create("agt", this);
23     cov = RAM_coverage :: type_id :: create("cov", this);
24     sb  = RAM_scoreboard :: type_id :: create("sb", this);
25   endfunction : build_phase
26
27   function void connect_phase (uvm_phase phase);
28     super.connect_phase(phase) ;
29     agt.agt_ap.connect(sb.sb_export);
30     agt.agt_ap.connect(cov.cov_export);
31   endfunction
32 endclass
33 endpackage
```

Do file :

```
1 vlib work
2 vlog -f src_files.list +cover +covercells +coveropt
3 vsim -voptargs+=acc work.top -classdebug -uvmcontrol=all -cover
4 vsim -sv_seed 1906057213 work.top
5 add wave -r sim:/top/RAM_if/*
6 coverage save RAM.ucdb -onexit
7 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 :

```
1 package agent_pkg;
2     import pkg_cfg::*;
3     import seq_item_pkg::*;
4     import RAM_sequencer_pkg::*;
5     import RAM_monitor_pkg::*;
6     import pkg_driver::*;
7     import uvm_pkg::*;
8     `include "uvm_macros.svh"
9
10    class RAM_agent extends uvm_agent;
11        `uvm_component_utils(RAM_agent)
12
13        RAM_driver drv ;
14        RAM_monitor mon ;
15        RAM_config_obj RAM_cfg ;
16        RAM_sequencer sqr ;
17        uvm_analysis_port #(RAM_seq_item) agt_ap ;
18
19        function new (string name = "RAM_agent", uvm_component parent = null);
20            super.new(name, parent) ;
21        endfunction
22
23        function void build_phase(uvm_phase phase);
24            super.build_phase(phase);
25            if (!uvm_config_db #(RAM_config_obj)::get(this, "", "CFG", RAM_cfg))
26                `uvm_fatal("build_phase", " unable to get configuration object");
27
28            sqr = RAM_sequencer :: type_id :: create("sqr", this) ;
29            drv = RAM_driver    :: type_id :: create("drv", this) ;
30            mon = RAM_monitor   :: type_id :: create("mon", this) ;
31            agt_ap = new("agt_ap", this);
32        endfunction
33
34        function void connect_phase(uvm_phase phase);
35            super.connect_phase(phase);
36            drv.RAM_vif = RAM_cfg.RAM_config_vif ;
37            mon.RAM_vif = RAM_cfg.RAM_config_vif ;
38            drv.seq_item_port.connect(sqr.seq_item_export);
39            mon.mon_ap.connect(agt_ap);
40        endfunction
41    endclass
42 endpackage
43
```

RAM driver file :

```
1 package pkg_driver;
2     import pkg_cfg::*;
3     import seq_item_pkg::*;
4     import RAM_read_only_seq_pkg::*;
5     import RAM_write_only_seq_pkg::*;
6     import RAM_read_write_seq_pkg::*;
7     import RAM_reset_seq_pkg::*;
8     import uvm_pkg::*;
9     `include "uvm_macros.svh"
10
11    class RAM_driver extends uvm_driver #(RAM_seq_item);
12        `uvm_component_utils(RAM_driver)
13
14        virtual RAM_intf RAM_vif ;
15        RAM_seq_item stim_seq_item ;
16
17        function new (string name = "RAM_driver", uvm_component parent = null);
18            super.new(name, parent) ;
19        endfunction
20
21        function void build_phase(uvm_phase phase);
22            super.build_phase(phase);
23        endfunction
24
25        task run_phase(uvm_phase phase);
26            super.run_phase(phase);
27            forever begin
28                stim_seq_item = RAM_seq_item :: type_id ::create("stim_seq_item") ;
29                seq_item_port.get_next_item(stim_seq_item);
30
31                @(negedge RAM_vif.clk);
32                RAM_vif.rst_n      = stim_seq_item.rst_n ;
33                RAM_vif.din       = stim_seq_item.din ;
34                RAM_vif.rx_valid  = stim_seq_item.rx_valid ;
35
36                seq_item_port.item_done();
37                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
38            end
39        endtask
40
41    endclass
42 endpackage
```

RAM monitor file :

```
● ○ ●
1 package RAM_monitor_pkg ;
2     import seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6 class RAM_monitor extends uvm_monitor ;
7     `uvm_component_utils(RAM_monitor)
8
9     RAM_seq_item rsp_seq_item ;
10    virtual RAM_intf RAM_vif ;
11    uvm_analysis_port #(RAM_seq_item) mon_ap ;
12
13    function new (string name = "RAM_monitor", uvm_component parent = null);
14        super.new(name, parent);
15    endfunction
16
17    function void build_phase(uvm_phase phase);
18        super.build_phase(phase);
19        mon_ap = new("mon_ap", this);
20    endfunction
21
22    task run_phase(uvm_phase phase);
23        super.run_phase(phase);
24        forever begin
25            rsp_seq_item = RAM_seq_item::type_id::create ("rsp_seq_item");
26
27            rsp_seq_item.rst_n      = RAM_vif.rst_n ;
28            rsp_seq_item.din       = RAM_vif.din ;
29            rsp_seq_item.rx_valid = RAM_vif.rx_valid ;
30            rsp_seq_item.dout     = RAM_vif.dout ;
31            rsp_seq_item.tx_valid = RAM_vif.tx_valid ;
32            rsp_seq_item.dout_ref = RAM_vif.dout_ref ;
33            rsp_seq_item.tx_valid_ref = RAM_vif.tx_valid_ref ;
34
35            @(negedge RAM_vif.clk);
36            mon_ap.write(rsp_seq_item);
37            `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
38        end
39    endtask
40 endclass
41 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

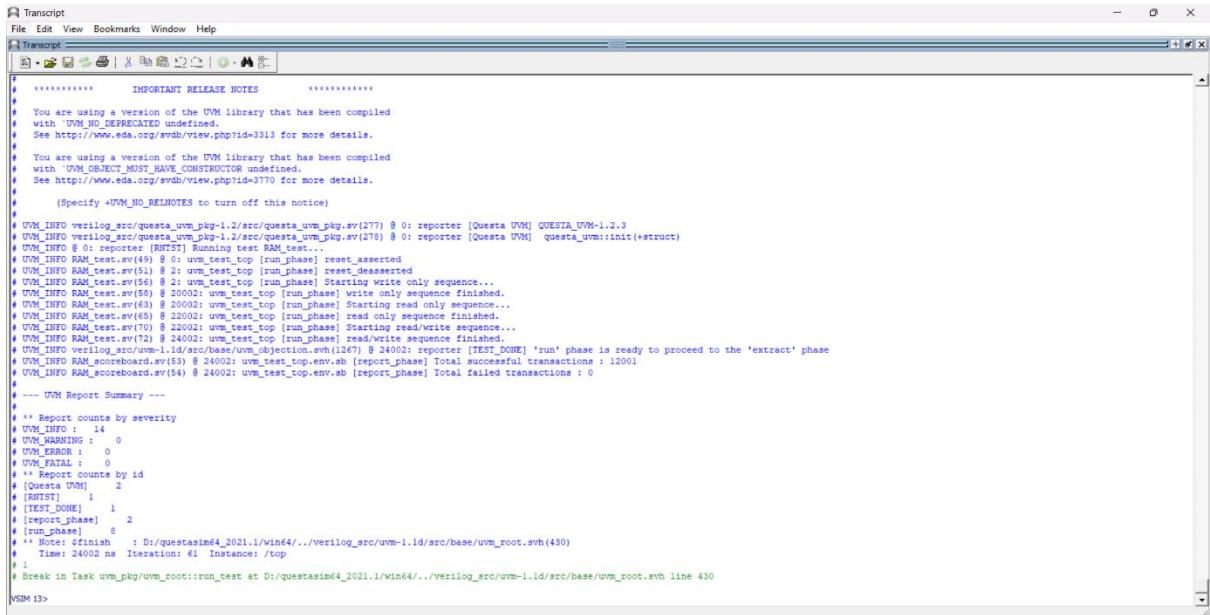
  function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info("report_phase",$sformatf("Total successful transactions : %0d",
correct_count),UVM_MEDIUM)
    `uvm_info("report_phase",$sformatf("Total failed transactions : %0d",
error_count),UVM_MEDIUM)
  endfunction
endclass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

RAM coverage file :

```
1 package RAM_coverage_pkg;
2     import seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class RAM_coverage extends uvm_component;
7         `uvm_component_utils(RAM_coverage)
8
9             uvm_analysis_export #(RAM_seq_item) cov_export;
10            uvm_tlm_analysis_fifo #(RAM_seq_item) cov_fifo;
11            RAM_seq_item seq_item_cov;
12
13            covergroup cg ;
14                //Coverpoint to check transaction ordering for din[9:8]
15                cp_din : coverpoint seq_item_cov.din[9:8] {
16                    bins din_00 = {2'b00};
17                    bins din_01 = {2'b01} ;
18                    bins din_10 = {2'b10} ;
19                    bins din_11 = {2'b11} ;
20                    bins write_address_data = (2'b00 => 2'b01) ;
21                    bins read_address_data = (2'b10 => 2'b11) ;
22                    bins write_read_data = (2'b00 => 2'b01 => 2'b10 => 2'b11) ;
23                }
24
25                //Coverpoint rx_valid
26                cp_rx : coverpoint seq_item_cov.rx_valid {bins rx_valid_1 = {1};}
27
28                //Coverpoint tx_valid
29                cp_tx : coverpoint seq_item_cov.tx_valid {bins tx_valid_1 = {1};}
30
31                //-----cross coverage-----
32                //Between all bins of din[9:8] and rx_valid signal when it is high
33                cross_rx_din : cross cp_din,cp_rx {
34                    bins rx_din_00 = binsof(cp_din.din_00) && binsof(cp_rx.rx_valid_1);
35                    bins rx_din_01 = binsof(cp_din.din_01) && binsof(cp_rx.rx_valid_1);
36                    bins rx_din_10 = binsof(cp_din.din_10) && binsof(cp_rx.rx_valid_1);
37                    bins rx_din_11 = binsof(cp_din.din_11) && binsof(cp_rx.rx_valid_1);
38                    option.cross_auto_bin_max = 0;
39                }
40
41                //Between din[9:8] when it equals read data and tx_valid when it is high
42                cross_tx_din : cross cp_din,cp_tx {
43                    bins tx_din_11 = binsof(cp_din.din_11) && binsof(cp_tx.tx_valid_1);
44                    option.cross_auto_bin_max = 0;
45                }
46            endgroup
47
48            function new (string name = "RAM_coverage" , uvm_component parent = null);
49                super.new(name , parent) ;
50                cg = new() ;
51            endfunction
52
53            function void build_phase (uvm_phase phase);
54                super.build_phase(phase);
55                cov_export = new("cov_export",this);
56                cov_fifo    = new("cov_fifo",this);
57            endfunction
58
59            function void connect_phase(uvm_phase phase);
60                super.connect_phase(phase);
61                cov_export.connect(cov_fifo.analysis_export);
62            endfunction
63
64            task run_phase (uvm_phase phase);
65                super.run_phase(phase);
66                forever begin
67                    cov_fifo.get(seq_item_cov);
68                    cg.sample();
69                end
70            endtask: run_phase
71        endclass
72    endpackage
```

Second: Snippets from Simulation results:

Transcript:



```
File Edit View Bookmarks Window Help
File Transcript
F ***** IMPORTANT RELEASE NOTES *****
You are using a version of the UVM library that has been compiled
with UVM_NO_DEPRECATED undefined.
See http://www.eda.org/svlib/view.php?id=3313 for more details.

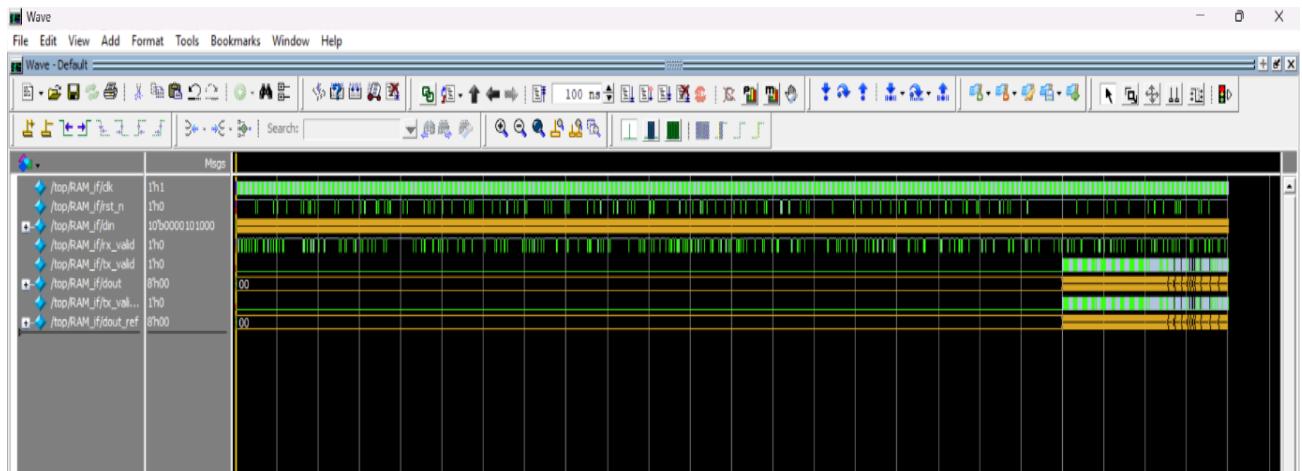
You are using a version of the UVM library that has been compiled
with UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
See http://www.eda.org/svlib/view.php?id=3770 for more details.

(Specify +UVM_NO_RELNOTES to turn off this notice)

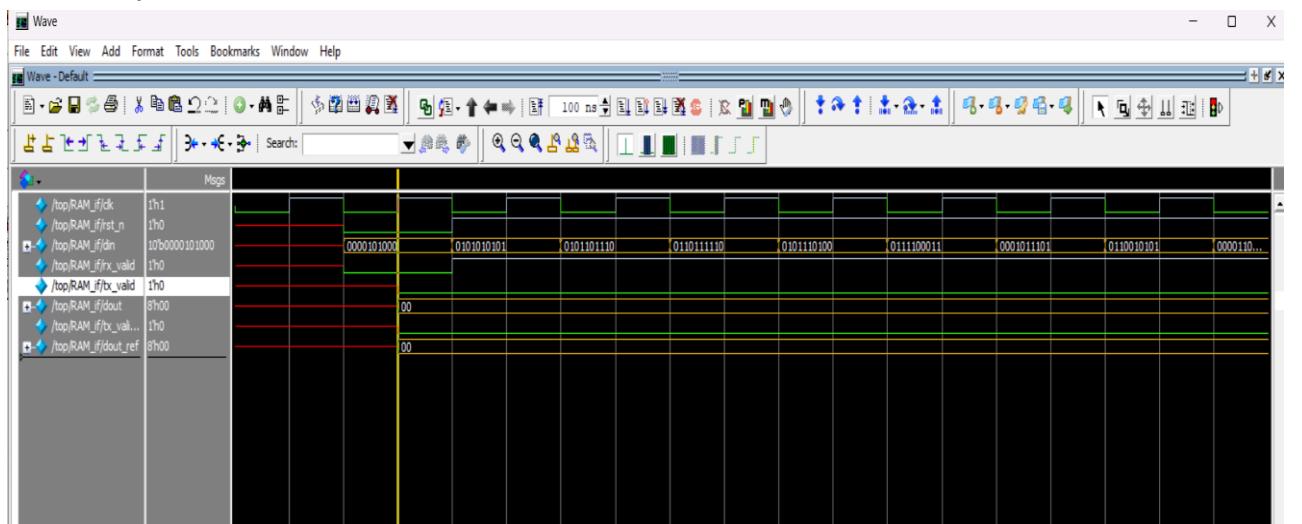
UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) 0: reporter [Questa UVM] questa_uvm::init(+struct)
UVM_TEST 0: 0 reporter [TEST] Running test RAM_test
UVM_INFO RAM_test.sv(49) 0: 2i uvm_phase [run_phase] reset_asserted
UVM_INFO RAM_test.sv(51) 0: 2i uvm_test_top [run_phase] reset_deasserted
UVM_INFO RAM_test.sv(56) 0: 2i uvm_test_top [run_phase] Starting write only sequence...
UVM_INFO RAM_test.sv(63) 0: 20002i uvm_test_top [run_phase] write only sequence finished.
UVM_INFO RAM_test.sv(66) 0: 20002i uvm_test_top [run_phase] Starting read only sequence...
UVM_INFO RAM_test.sv(70) 0: 22002i uvm_test_top [run_phase] read only sequence finished.
UVM_INFO RAM_test.sv(72) 0: 24002i uvm_test_top [run_phase] read/write sequence finished.
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objectection.svh(1267) 0: 24002i reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO RAM_scoreboard.sv(53) 0: 24002i uvm_test_top.env.sv [report_phase] Total successful transactions : 12001
UVM_INFO RAM_scoreboard.sv(54) 0: 24002i uvm_test_top.env.sv [report_phase] Total failed transactions : 0
--- UVM Report Summary ---
** Report counts by severity
UVM_FATAL : 14
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FAULT : 0
** Report counts by id
[Questa UVM] 2
[TEST] 1
[TEST_DONE] 1
[report_phase] 2
[run_phase] 8
** Note: $finish : D:/questasim64_2021.1/win64//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
1
Time: 24002 ns Iteration: 61 Instance: /top
1
Break in Task uvm_pkg/uvm_root::run_test at D:/questasim64_2021.1/win64//verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
VSM 1>
```

Waveforms:

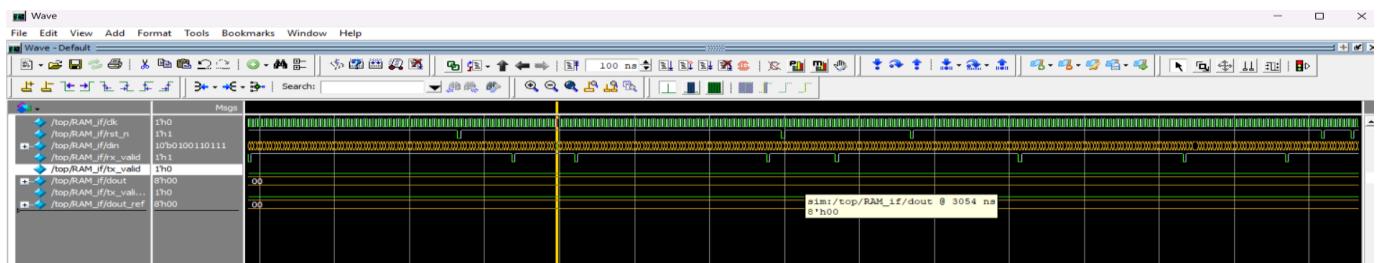
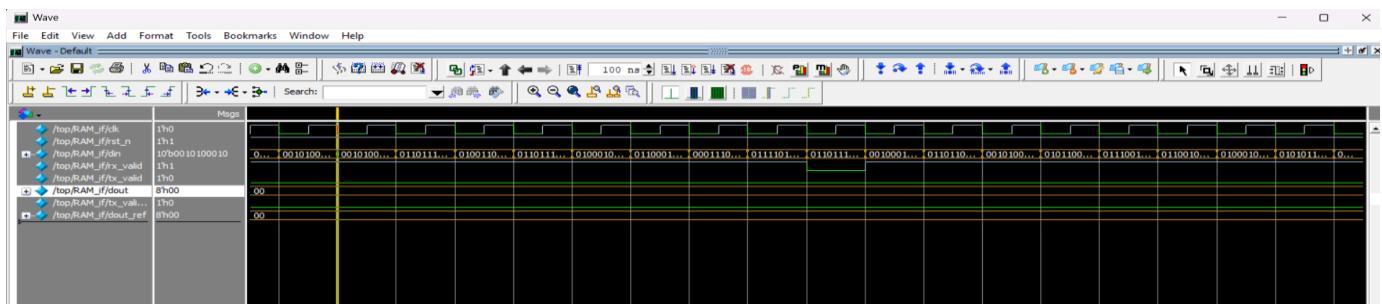
➤ Full wave:



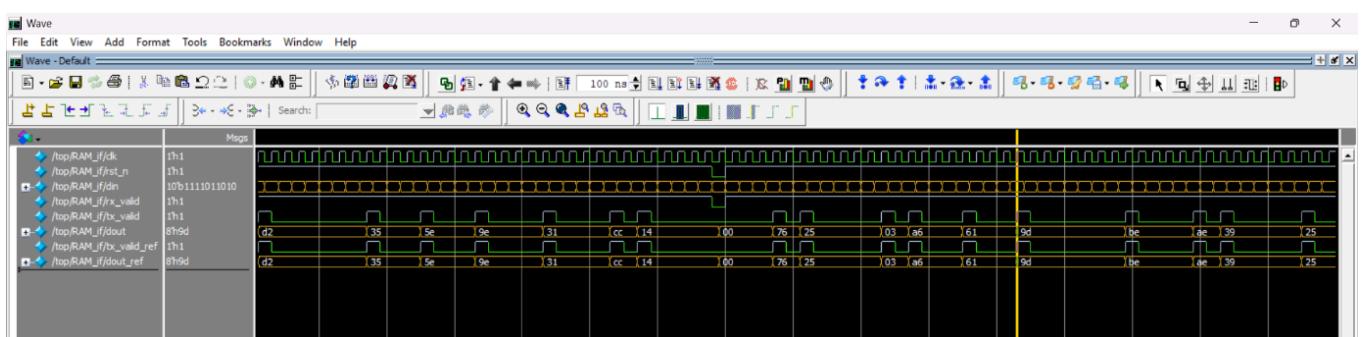
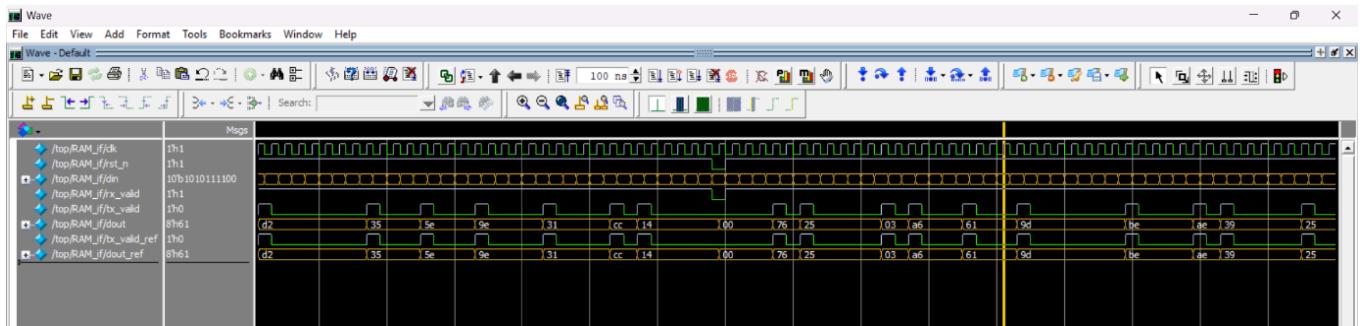
➤ Reset seq:



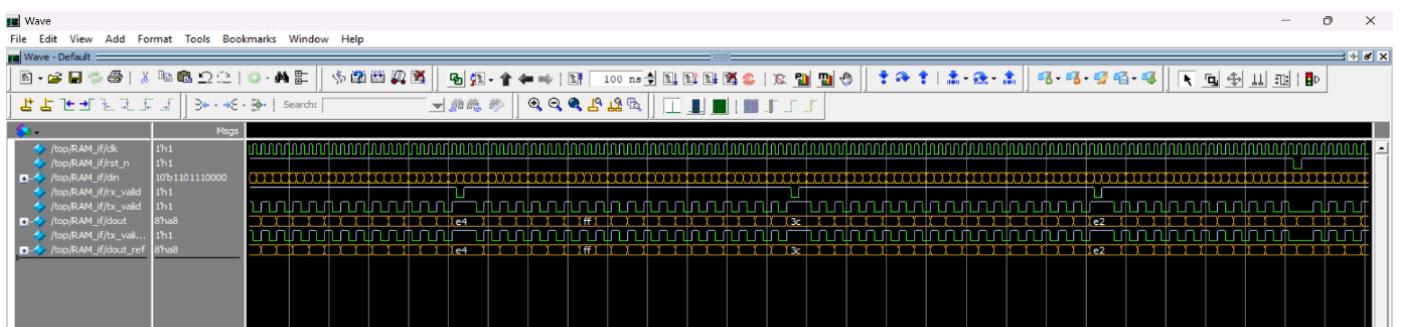
➤ Write only seq:

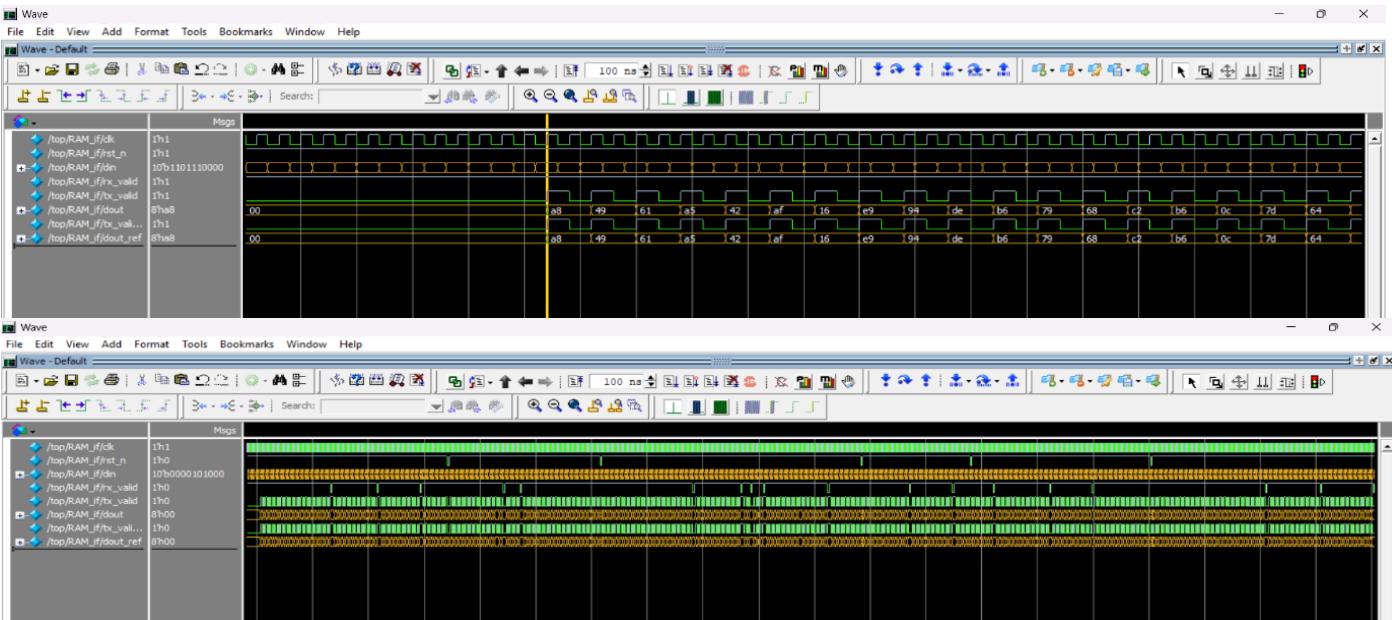


➤ READ ONLY SEQ:



➤ WRITE-READ SEQ:





Assertions coverage:

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Express
lvm_pigzum_reg_map:do_write#ublk#215181159#1731/lmmed_1735	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert (\$cast(sea,
lvm_pigzum_reg_map:idc_read#ublk#215181159#1771/lmmed_1775	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert (\$cast(sea,
RAM_reset_seq_pkp:RAM_reset_seq_pkp:lmmed_20	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (randomize(
RAM_read_write_seq_pkp:RAM_read_write_seq_pkp:#ublk#31059495#17/lmmed_23	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (randomize(
RAM_write_only_seq_pkp:RAM_write_only_seq_pkp:#ublk#21903535#17/lmmed_24	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (randomize(
RAM_read_only_seq_pkp:RAM_read_only_seq_pkp:#ublk#244104311#17/lmmed_23	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert (randomize(
top/DUT/sva_inst/c_reset	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
top/DUT/sva_inst/c_tx_low	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
top/DUT/sva_inst/c_tx_high	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
top/DUT/sva_inst/c_write	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-
top/DUT/sva_inst/c_read	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	-

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/sva_inst/c_reset	SVA	✓	Off	134	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/top/DUT/sva_inst/c_tx_low	SVA	✓	Off	5136	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/top/DUT/sva_inst/c_tx_high	SVA	✓	Off	630	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/top/DUT/sva_inst/c_write	SVA	✓	Off	5238	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
/top/DUT/sva_inst/c_read	SVA	✓	Off	900	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0

Function coverage:

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
RAM_coverage_pkp:RAM_coverage		100.00%							
TTFB_cg		100.00%	100	100.00%	✓	✓			auto(0)
CVP cp:cp_din		100.00%	100	100.00%	✓	✓			
CVP cp:cp_rx		100.00%	100	100.00%	✓	✓			
CVP cp:cp_tx		100.00%	100	100.00%	✓	✓			
CROSS cp:cross_rx_din		100.00%	100	100.00%	✓	✓			
CROSS cp:cross_tx_din		100.00%	100	100.00%	✓	✓			
INST [RAM_coverage_pkp:RAM_coverage::cg		100.00%	100	100.00%	✓	✓			0
CVP cp_din		100.00%	100	100.00%	✓	✓			
bin din_0		5373	1	100.00%	✓	✓			
bin din_01		5159	1	100.00%	✓	✓			
bin din_10		813	1	100.00%	✓	✓			
bin din_11		656	1	100.00%	✓	✓			
bin write_address_data		2688	1	100.00%	✓	✓			
bin read_address_data		656	1	100.00%	✓	✓			
bin write_read_data		55	1	100.00%	✓	✓			
CVP cp_rx		100.00%	100	100.00%	✓	✓			
bin rx_valid_1		11752	1	100.00%	✓	✓			
CVP cp_tx		100.00%	100	100.00%	✓	✓			
bin tx_valid_1		653	1	100.00%	✓	✓			
CROSS cross_rx_din		100.00%	100	100.00%	✓	✓			
bin rx_din_00		5267	1	100.00%	✓	✓			
bin rx_din_01		5048	1	100.00%	✓	✓			
bin rx_din_10		793	1	100.00%	✓	✓			
bin rx_din_11		644	1	100.00%	✓	✓			
CROSS cross_tx_din		100.00%	100	100.00%	✓	✓			
bin tx_din_11		10	1	100.00%	✓	✓			

Code coverage:

-Statements:

```

RAM.v
13 always @(posedge clk) begin
15 dout      <= 0;
16 tx_valid <= 0;
17 Rd_Addr  <= 0;
18 Wr_Addr  <= 0;
23 2'b00 : Wr_Addr      <= din[7:0];
24 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 2'b10 : Rd_Addr      <= din[7:0];
26 2'b11 : dout         <= MEM[Rd_Addr];//bug fixed
29 tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;

```

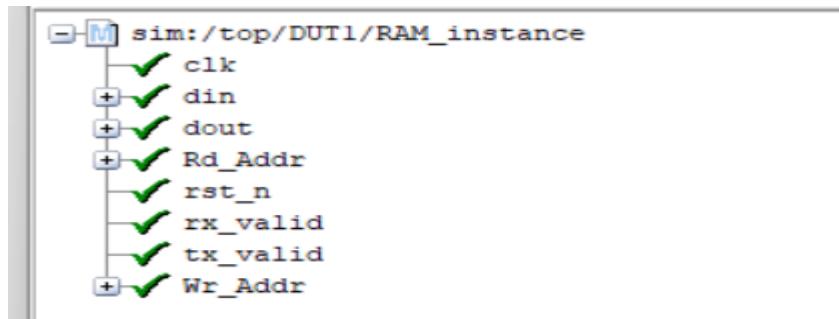
-Branches:

```

RAM.v
14 if (~rst_n) begin
20 else begin
21 if (rx_valid) begin //add begin/end
23 2'b00 : Wr_Addr      <= din[7:0];
24 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 2'b10 : Rd_Addr      <= din[7:0];
26 2'b11 : dout         <= MEM[Rd_Addr];//bug fixed

```

-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 concurrent 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 reference model in the score board 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 reference model in the score board and SVA inside SVA file to check internal signals
RAM_3	<ul style="list-style-type: none"> 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 	Randomized with constraint that drive the din[9:8] to match required specifications <ul style="list-style-type: none"> 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 reference model in the score board and SVA inside SVA file to check internal signals

Wrapper part

First : Changed files in RAM and SLAVE env.

SLAVE configuration object edited :

```
1 package SLAVE_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class SLAVE_config extends uvm_object;
5     `uvm_object_utils(SLAVE_config)
6     virtual SLAVE_if SLAVE_vif ;
7     uvm_active_passive_enum is_active;
8     function new(string name = "SLAVE_config");
9       super.new(name) ;
10      endfunction
11    endclass
12 endpackage
```

RAM configuration object edited :

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

SLAVE agent file_edited :

```
1 package SLAVE_agent_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import SLAVE_sequencer_pkg::*;
5     import SLAVE_driver_pkg::*;
6     import SLAVE_config_pkg::*;
7     import SLAVE_monitor_pkg::*;
8     import SLAVE_item_pkg::*;
9
10    class SLAVE_agent extends uvm_agent;
11        `uvm_component_utils(SLAVE_agent)
12        SLAVE_sequencer sqr ;
13        SLAVE_driver drv ;
14        SLAVE_monitor mon;
15        SLAVE_config SLAVE_cfg;
16        uvm_analysis_port #(SLAVE_item) agt_ap;
17
18        function new(string name = "SLAVE_agent" , uvm_component parent = null );
19            super.new(name, parent);
20        endfunction
21
22        function void build_phase (uvm_phase phase);
23            super.build_phase(phase);
24            if (!uvm_config_db #(SLAVE_config)::get(this, "", "CFG", SLAVE_cfg)) begin
25                `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
26            end
27            if(SLAVE_cfg.is_active== UVM_ACTIVE) begin
28                sqr = SLAVE_sequencer:::type_id:::create("sqr",this);
29                drv = SLAVE_driver:::type_id:::create("drv",this);
30            end
31            mon = SLAVE_monitor:::type_id:::create("mon",this);
32            agt_ap = new("agt_ap",this);
33        endfunction
34
35        function void connect_phase(uvm_phase phase);
36            super.connect_phase(phase);
37            if(SLAVE_cfg.is_active== UVM_ACTIVE) begin
38                drv.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
39                drv.seq_item_port.connect(sqr.seq_item_export);
40            end
41            mon.SLAVE_vif = SLAVE_cfg.SLAVE_vif;
42            mon.mon_ap.connect(agt_ap);
43        endfunction
44    endclass
45 endpackage
```

RAM agent file_edited :

```
1 package agent_pkg;
2     import pkg_cfg::*;
3     import seq_item_pkg::*;
4     import RAM_sequencer_pkg::*;
5     import RAM_monitor_pkg::*;
6     import pkg_driver::*;
7     import uvm_pkg::*;
8     `include "uvm_macros.svh"
9
10    class RAM_agent extends uvm_agent;
11        `uvm_component_utils(RAM_agent)
12
13        RAM_driver drv ;
14        RAM_monitor mon ;
15        RAM_config_obj RAM_cfg ;
16        RAM_sequencer sqr ;
17        uvm_analysis_port #(RAM_seq_item) agt_ap ;
18
19        function new (string name = "RAM_agent", uvm_component parent = null);
20            super.new(name, parent) ;
21        endfunction
22
23        function void build_phase(uvm_phase phase);
24            super.build_phase(phase);
25            if (!uvm_config_db #(RAM_config_obj)::get(this, "", "CFG", RAM_cfg))
26                `uvm_fatal("build_phase", " unable to get configuration object");
27
28            if(RAM_cfg.is_active== UVM_ACTIVE) begin
29                sqr = RAM_sequencer::type_id::create("sqr",this);
30                drv = RAM_driver::type_id::create("drv",this);
31            end
32            mon = RAM_monitor::type_id::create("mon",this);
33            agt_ap = new("agt_ap",this);
34        endfunction
35
36        function void connect_phase(uvm_phase phase);
37            super.connect_phase(phase);
38            if(RAM_cfg.is_active== UVM_ACTIVE) begin
39                drv.RAM_vif = RAM_cfg.RAM_config_vif;
40                drv.seq_item_port.connect(sqr.seq_item_export);
41            end
42            mon.RAM_vif = RAM_cfg.RAM_config_vif ;
43            mon.mon_ap.connect(agt_ap);
44        endfunction
45    endclass
46 endpackage
47
```

Second : Wrapper env. Files

Wrapper interface file :

```
● ○ ●
1  interface wrapper_if(clk);
2    input logic clk;
3    logic MOSI,MISO_ref,SS_n,rst_n;
4    bit MISO;
5    logic tx_valid;
6    logic [7:0] tx_data;
7    logic [9:0] rx_data;
8    logic rx_valid;
9    modport wrp_sva (input rx_data, rx_valid, clk, MOSI, MISO, SS_n, rst_n, tx_data, tx_valid);
10   endinterface
```

Wrapper design & assertion file :

```
● ○ ●
1  module wrapper (wrapper_if wrapperif);
2    logic [9:0] rx_data;
3    logic      rx_valid;
4    logic      tx_valid;
5    logic [7:0] tx_data;
6
7    RAM    RAM_instance (rx_data,wrapperif.clk,wrapperif.rst_n,rx_valid,
8                      tx_data,tx_valid);
9
10   SLAVE SLAVE_instance (wrapperif.MOSI,wrapperif.MISO,wrapperif.SS_n,wrapperif.clk,
11                      wrapperif.rst_n,rx_data,rx_valid,tx_data,tx_valid);
12
13   assign wrapperif.rx_data = rx_data;
14   assign wrapperif.rx_valid = rx_valid;
15   assign wrapperif.tx_data = tx_data;
16   assign wrapperif.tx_valid = tx_valid;
17
18 // assertions section
19 // assertion 1 : An assertion ensures that whenever reset is asserted, the output (MISO) is inactive.
20   property p_reset_outputs_inactive;
21   @(posedge wrapperif.clk) !wrapperif.rst_n |=> (wrapperif.MISO == 0 &&
22                                         wrapperif.rx_valid == 0 &&
23                                         wrapperif.rx_data == 0);
24   endproperty
25
26   assert property (p_reset_outputs_inactive);
27   cover property (p_reset_outputs_inactive);
28 // assertion 2 : An assertion to make sure that the MISO remains with a stable value eventually as long
29 //                 as it is not a read data operation
30   property stable_miso;
31   @(wrapperif.clk) disable iff (!wrapperif.rst_n) (wrapperif.rx_data[9:8] != 2'b11) |=> $stable(wrapperif.MISO);
32   endproperty
33
34   assert property (stable_miso);
35   cover property (stable_miso);
36
37 endmodule
```

Wrapper golden model :

```
1  module golden_model_SLAVE (
2      MOSI, MISO, SS_n, clk, rst_n, rx_data, rx_valid, tx_data, tx_valid
3  );
4
5  localparam S_IDLE      = 3'b000;
6  localparam S_WRITE     = 3'b001;
7  localparam S_CMDCHK   = 3'b010;
8  localparam S_RDADDR   = 3'b011;
9  localparam S_RDDATA   = 3'b100;
10
11 input      MOSI, clk, rst_n, SS_n, tx_valid;
12 input      [7:0] tx_data;
13 output reg [9:0] rx_data;
14 output reg      rx_valid;
15 output bit      MISO;
16
17 // renamed internal signals
18 reg [3:0] bit_counter;
19 reg      addr_received;
20
21 reg [2:0] state_curr, state_next;
22
23 // State register
24 always @(posedge clk) begin
25     if (~rst_n)
26         state_curr <= S_IDLE;
27     else
28         state_curr <= state_next;
29 end
30
31 // Next state logic
32 always @(*) begin
33     case (state_curr)
34         S_IDLE: begin
35             if (SS_n)
36                 state_next = S_IDLE;
37             else
38                 state_next = S_CMDCHK;
39         end
40
41         S_CMDCHK: begin
42             if (SS_n)
43                 state_next = S_IDLE;
44             else begin
45                 if (~MOSI)
46                     state_next = S_WRITE;
47                 else begin
48                     if (addr_received)
49                         state_next = S_RDDATA;
50                     else
51                         state_next = S_RDADDR;
52                 end
53             end
54         end
55
56         S_WRITE: begin
57             if (SS_n)
58                 state_next = S_IDLE;
59             else
60                 state_next = S_WRITE;
61         end
62
63         S_RDADDR: begin
64             if (SS_n)
65                 state_next = S_IDLE;
66             else
67                 state_next = S_RDADDR;
68         end
69
70         S_RDDATA: begin
71             if (SS_n)
72                 state_next = S_IDLE;
73             else
74                 state_next = S_RDDATA;
75         end
76     endcase
77 end
```

```

1 // Output and data handling
2 always @(posedge clk) begin
3     if (~rst_n) begin
4         rx_data      <= 0;
5         rx_valid    <= 0;
6         addr_received <= 0;
7         MISO        <= 0;
8         bit_counter <= 0;
9     end
10    else begin
11        case (state_curr)
12            S_IDLE: begin
13                rx_valid <= 0;
14            end
15
16            S_CMDCHK: begin
17                bit_counter <= 10;
18            end
19
20            S_WRITE: begin
21                if (bit_counter > 0) begin
22                    rx_data[bit_counter-1] <= MOSI;
23                    bit_counter <= bit_counter - 1;
24                end
25                else begin
26                    rx_valid <= 1;
27                end
28            end
29
30            S_RDADDR: begin
31                if (bit_counter > 0) begin
32                    rx_data[bit_counter-1] <= MOSI;
33                    bit_counter <= bit_counter - 1;
34                end
35                else begin
36                    rx_valid      <= 1;
37                    addr_received <= 1;
38                end
39            end
40
41            S_RDDATA: begin
42                if (tx_valid) begin
43                    rx_valid <= 0;
44                    if (bit_counter > 0) begin
45                        MISO <= tx_data[bit_counter-1];
46                        bit_counter <= bit_counter - 1;
47                    end
48                    else begin
49                        addr_received <= 0;
50                    end
51                end
52                else begin
53                    if (bit_counter > 0) begin
54                        rx_data[bit_counter-1] <= MOSI;
55                        bit_counter <= bit_counter - 1;
56                        rx_valid <= 0;
57                    end
58                    else begin
59                        rx_valid <= 1;
60                        bit_counter <= 9;
61                    end
62                end
63            end
64        endcase
65    end
66 end
67
68 endmodule

```

```

1  module golden_model_RAM (din,clk,rst_n,rx_valid,dout,tx_valid);
2
3     parameter MEM_DEPTH = 256;
4     parameter ADDR_SIZE = 8;
5
6     input [9:0] din;
7     input clk, rst_n, rx_valid;
8
9     output reg [7:0] dout;
10    output reg tx_valid;
11
12    reg [ADDR_SIZE-1 : 0] addr_rd, addr_wr;
13    reg [7:0] mem [MEM_DEPTH-1 : 0];
14
15    integer i;
16    always @(posedge clk) begin
17        if (!rst_n) begin
18            dout <= 0 ;
19            tx_valid <= 0 ;
20            addr_rd <= 0 ;
21            addr_wr <= 0 ;
22        end
23        else begin
24            if (rx_valid) begin
25                case (din[9:8])
26                    2'b00 : begin
27                        addr_wr <= din[7:0] ;
28                        tx_valid <= 1'b0 ;
29                    end
30                    2'b01 : begin
31                        mem[addr_wr] <= din[7:0] ;
32                        tx_valid <= 1'b0 ;
33                    end
34                    2'b10 : begin
35                        addr_rd <= din[7:0] ;
36                        tx_valid <= 1'b0 ;
37                    end
38                    2'b11 : begin
39                        dout <= mem[addr_rd] ;
40                        tx_valid <= 1'b1 ;
41                    end
42                    default: dout <= 0;
43                endcase
44            end
45        end
46    end
47 endmodule
48
49
50 module golden_model_SPI_wrapper (
51     input logic MOSI_ref,
52     output logic MISO_ref,
53     input logic SS_n_ref,
54     input logic clk,
55     input logic rst_n_ref
56 );
57
58 // Internal signals connecting RAM and SLAVE
59 logic [9:0] rx_data_din_ref;
60 logic rx_valid_ref;
61 logic [7:0] tx_data_dout_ref;
62 logic tx_valid_ref;
63
64 // Instantiate the golden model RAM and SLAVE (with identical logic)
65 golden_model_RAM RAM_ref (
66     .din(rx_data_din_ref),
67     .clk(clk),
68     .rst_n(rst_n_ref),
69     .rx_valid(rx_valid_ref),
70     .dout(tx_data_dout_ref),
71     .tx_valid(tx_valid_ref)
72 );
73
74 golden_model_SLAVE SLAVE_ref (
75     .MOSI(MOSI_ref),
76     .MISO(MISO_ref),
77     .SS_n(SS_n_ref),
78     .clk(clk),
79     .rst_n(rst_n_ref),
80     .rx_data(rx_data_din_ref),
81     .rx_valid(rx_valid_ref),
82     .tx_data(tx_data_dout_ref),
83     .tx_valid(tx_valid_ref)
84 );
85
86 endmodule

```

Wrapper top file :

```
● ● ●
1 import uvm_pkg::*;
2 `include "uvm_macros.svh"
3 import wrapper_test_pkg::*;
4 import SLAVE_test_pkg::*;
5 import RAM_test_pkg::*;
6
7 module top();
8     bit clk;
9
10    initial begin
11        forever
12            #1 clk =~clk ;
13    end
14
15    wrapper_if wrapperif (clk);
16    SLAVE_if slaveif (clk);
17    RAM_if ramif (clk);
18
19    // Instantiate the wrapper (which contains both RAM and SLAVE internally)
20    wrapper DUT1 (wrapperif);
21
22    // Golden model for wrapper
23    golden_model_SPI_wrapper wrapper_ins (
24        .MOSI_ref(wrapperif.MOSI),
25        .MISO_ref(wrapperif.MISO_ref),
26        .SS_n_ref(wrapperif.SS_n),
27        .clk(wrapperif.clk),
28        .rst_n_ref(wrapperif.rst_n)
29    );
30
31    bind DUT1.RAM_instance RAM_sva RAM_assertion (
32        .clk(wrapperif.clk),
33        .rst_n(wrapperif.rst_n),
34        .tx_valid(wrapperif.tx_valid),
35        .dout(wrapperif.tx_data),
36        .din(wrapperif.rx_data),
37        .rx_valid(wrapperif.rx_valid)
38    );
39
40    // Connect SLAVE interface to wrapper's internal SLAVE signals for monitoring
41    assign slaveif.rst_n = wrapperif.rst_n;
42    assign slaveif.SS_n = wrapperif.SS_n;
43    assign slaveif.MOSI = wrapperif.MOSI;
44    assign slaveif.MISO = wrapperif.MISO;
45    assign slaveif.rx_data = DUT1.rx_data;
46    assign slaveif.rx_valid = DUT1.rx_valid;
47    assign slaveif.tx_data = DUT1.tx_data;
48    assign slaveif.tx_valid = DUT1.tx_valid;
49
50    // Connect RAM interface to wrapper's internal RAM signals for monitoring
51    assign ramif.rst_n = wrapperif.rst_n;
52    assign ramif.din = DUT1.rx_data;
53    assign ramif.rx_valid = DUT1.rx_valid;
54    assign ramif.dout = DUT1.tx_data;
55    assign ramif.tx_valid = DUT1.tx_valid;
56
57    // Connect RAM golden interface to wrapper's internal RAM signals for monitoring
58    assign ramif.dout_ref = DUT1.tx_data;
59    assign ramif.tx_valid_ref = DUT1.tx_valid;
60
61    initial begin
62        uvm_config_db #(virtual wrapper_if)::set(null, "uvm_test_top", "WRAPPER_IF", wrapperif);
63        uvm_config_db #(virtual SLAVE_if)::set(null, "uvm_test_top", "SLAVE_IF", slaveif);
64        uvm_config_db #(virtual RAM_if)::set(null, "uvm_test_top", "RAM_IF", ramif);
65        run_test("wrapper_test");
66    end
67
68 endmodule
```

Wrapper test file :

```
1 package wrapper_test_pkg;
2 import SLAVE_env_pkg::*;
3 import SLAVE_config_pkg::*;
4 import RAM_env_pkg::*;
5 import pkg_cfg::*;
6 import wrapper_env_pkg::*;
7 import wrapper_config_pkg::*;
8 import wrapper_reset_seq_pkg::*;
9 import uvm_pkg::*;
10 import wrapper_read_only_seq_pkg::*;
11 import wrapper_write_only_seq_pkg::*;
12 import wrapper_read_write_seq_pkg::*;
13 `include "uvm_macros.svh"
14
15 class wrapper_test extends uvm_test;
16   `uvm_component_utils(wrapper_test)
17   wrapper_env env ;
18   wrapper_config wrapper_cfg ;
19   wrapper_reset_seq rst_seq;
20   RAM_env env_ram ;
21   SLAVE_env env_slave ;
22   SLAVE_config slave_cfg ;
23   RAM_config_obj ram_cfg ;
24   wrapper_read_only_seq ro_seq;
25   wrapper_write_only_seq wo_seq;
26   wrapper_read_write_seq rw_seq;
27
28   function new (string name = "wrapper_test" , uvm_component parent = null);
29     super.new (name , parent) ;
30   endfunction
31
32   function void build_phase (uvm_phase phase);
33     super.build_phase(phase) ;
34     env = wrapper_env::type_id::create("env",this);
35     env_ram = RAM_env::type_id::create("env_ram",this);
36     env_slave = SLAVE_env::type_id::create("env_slave",this);
37     wrapper_cfg = wrapper_config::type_id::create("wrapper_cfg");
38     ram_cfg = RAM_config_obj::type_id::create("ram_cfg");
39     slave_cfg = SLAVE_config::type_id::create("slave_cfg");
40     rst_seq = wrapper_reset_seq::type_id::create("rst_seq");
41     ro_seq = wrapper_read_only_seq::type_id::create("ro_seq");
42     wo_seq = wrapper_write_only_seq::type_id::create("wo_seq");
43     rw_seq = wrapper_read_write_seq::type_id::create("rw_seq");
44
45     if (!uvm_config_db #(virtual wrapper_if)::get(this, "" , "WRAPPER_IF" , wrapper_cfg.wrapper_vif))
46       `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface");
47
48     if(!uvm_config_db #(virtual SLAVE_if)::get(this, "" , "SLAVE_IF" , slave_cfg.SLAVE_vif))
49       `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface of SLAVE");
50
51     if(!uvm_config_db #(virtual RAM_if)::get(this, "" , "RAM_IF" , ram_cfg.RAM_config_vif))
52       `uvm_fatal("build_phase" , "TEST - unable to get the virtual interface of RAM");
53
54     ram_cfg.is_active = UVM_PASSIVE;
55     slave_cfg.is_active = UVM_PASSIVE;
56     wrapper_cfg.is_active = UVM_ACTIVE;
57
58     // Set config objects with correct keys that agents expect
59     uvm_config_db #(RAM_config_obj)::set(this , "*" , "CFG" , ram_cfg);
60     uvm_config_db #(SLAVE_config)::set(this , "*" , "CFG" , slave_cfg);
61     uvm_config_db #(wrapper_config)::set(this , "*" , "CFG" , wrapper_cfg);
62   endfunction
63
64   task run_phase (uvm_phase phase);
65     super.run_phase(phase);
66     phase.raise_objection(this);
67
68     `uvm_info("run_phase", "reset asserted" , UVM_MEDIUM)
69     rst_seq.start(env.agt.sqr);
70     `uvm_info("run_phase", "reset deasserted" , UVM_MEDIUM)
71
72     `uvm_info("run_phase", "write only sequence started" , UVM_MEDIUM)
73     wo_seq.start(env.agt.sqr);
74     `uvm_info("run_phase", "write only sequence finished" , UVM_MEDIUM)
75
76     `uvm_info("run_phase", "read only sequence started" , UVM_MEDIUM)
77     ro_seq.start(env.agt.sqr);
78     `uvm_info("run_phase", "read only sequence finished" , UVM_MEDIUM)
79
80     `uvm_info("run_phase", "read/write sequence started" , UVM_MEDIUM)
81     rw_seq.start(env.agt.sqr);
82     `uvm_info("run_phase", "read/write sequence finished" , UVM_MEDIUM)
83
84     phase.drop_objection(this);
85   endtask
86 endclass
87
88 endpackage
```

Wrapper seq_item file :

```
1 package wrapper_seq_item_pkg;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 class wrapper_seq_item extends uvm_sequence_item;
5   `uvm_object_utils(wrapper_seq_item)
6
7   // Randomized inputs
8   rand logic rst_n;
9   rand logic SS_n;
10  rand bit [10:0] MOSI_array;
11
12  // Non-randomized inputs (driven from MOSI_array or constraints)
13  logic MOSI;
14
15  // DUT outputs (monitored)
16  bit MISO, MISO_ref;
17  bit [2:0] prev_op = 3'b000;
18
19  // State tracking variables
20  logic [4:0] count;           // Counter for SS_n timing (0-22)
21  logic [3:0] bit_index;       // Track which bit of MOSI_array to send (0-10)
22  logic prev_SS_n;           // Previous value of SS_n
23  bit [10:0] stored_MOSI_array; // Store the randomized array
24  bit [2:0] prev_cmd;         // store the previous command
25
26  function new(string name = "wrapper_seq_item");
27    super.new(name);
28    prev_SS_n = 1'b1;
29    count = 0;
30    bit_index = 0;
31    stored_MOSI_array = 11'b000_0000_0000;
32  endfunction
33
34  function string convert2string();
35    return $sformatf("%s rst_n=%b, SS_n=%b, MOSI=%b",
36                      super.convert2string(), rst_n, SS_n, MOSI);
37  endfunction
38
39  function string convert2string_stimuls();
40    return $sformatf("rst_n=%b, SS_n=%b, MOSI=%b, MISO=%b, MISO_ref=%b",
41                      rst_n, SS_n, MOSI, MISO, MISO_ref);
42  endfunction
43
44  // constraint section
45  // constraint 1 : reset constraint
46  constraint reset_cons {
47    rst_n dist {1 := 99, 0 := 1};
48  }
49
50  // Constraint 2: SS_n timing based on command type
51  // For WRITE (000) and READ_ADD (001): SS_n high every 13 cycles
52  // For READ_DATA (110, 111): SS_n high every 23 cycles
53  constraint SS_n_timing_c {
54    if (MOSI_array[10:8] inside {3'b000, 3'b001, 3'b110}) {
55      // Write or Read Address: 13 cycle period (12 low, 1 high)
56      SS_n == (count == 12);
57    }
58    else if (MOSI_array[10:8] inside {3'b111}) {
59      // Read Data: 23 cycle period (22 low, 1 high)
60      SS_n == (count == 22);
61    }
62    else {
63      // Default/initial case
64      SS_n == (count == 12);
65    }
66  }
67
68  constraint addr_change {
69    MOSI_array[7:0] dist { [8'h00:8'hFF] := 100 };
70  }
71
72  // For a write-only sequence, every Write Address operation shall always be followed by either
73  // Write Address or Write Data operation.
74  constraint c_write_only {
75    if (prev_cmd == 3'b000) {
76      MOSI_array[10:8] == 3'b001;
77    }
78    else if (prev_cmd == 3'b001) {
79      MOSI_array[10:8] == 3'b000 ;
80    }
81    else {
82      MOSI_array[10:8] inside {3'b000 , 3'b001};
83    }
84  };
85
86  // For a read-only sequence, only Read Address or Read Data operations are allowed.
87  constraint c_read_only {
88    if (prev_cmd == 3'b110) {
89      MOSI_array[10:8] == 3'b111;
90    }
91    else if (prev_cmd == 3'b111) {
92      MOSI_array[10:8] == 3'b110;
93    }
94    else {
95      MOSI_array[10:8] inside {3'b110 , 3'b111 };
96    }
97  };

```

```

1      // For a read-only sequence, only Read Address or Read Data operations are allowed.
2      constraint c_read_write {
3          if (prev_cmd == 3'b000){
4              MOSI_array[10:8] inside {3'b000 , 3'b001};
5          }
6          else if (prev_cmd == 3'b001) {
7              MOSI_array[10:8] dist {3'b000:=40 , 3'b110:=60};
8          }
9          else if (prev_cmd == 3'b110) {
10             MOSI_array[10:8] == 3'b111;
11         }
12         else if (prev_cmd == 3'b111) {
13             MOSI_array[10:8] dist {3'b000:=60 , 3'b110:=40};
14         }
15         else {
16             MOSI_array[10:8] inside {3'b000 , 3'b001 , 3'b110 , 3'b111};
17         }
18     };
19
20     // Post-randomize to handle bit-by-bit MOSI assignment and state updates
21     function void post_randomize();
22         logic is_falling_edge;
23
24         is_falling_edge = (prev_SS_n && !SS_n);
25
26         if (!rst_n) begin
27             count      = 0;
28             bit_index  = 0;
29             prev_SS_n  = 1'b1;
30             stored_MOSI_array = 11'b0;
31             MOSI_array[10:8]  = 3'b000;
32             prev_cmd   = 0 ;
33             MOSI       = 1'b0;
34             `uvm_info("RESET", "Reset occurred: internal state reinitialized", UVM_HIGH)
35             return; // Skip normal transaction logic this cycle
36         end
37
38         // Handle SS_n falling edge - start new transaction
39         if (is_falling_edge) begin
40             stored_MOSI_array = MOSI_array;
41             prev_cmd = MOSI_array[10:8]; // Store previous before updating
42             bit_index = 0;
43             count = 0;
44             MOSI = stored_MOSI_array[10]; // Start with MSB
45
46             `uvm_info("WRAPPER_ITEM", $sformatf(
47                 "New Transaction: cmd=%b, MOSI_array=%b, count=%b",
48                 prev_cmd, stored_MOSI_array, count
49             ), UVM_HIGH)
50         end
51         // Handle SS_n active (low) - continue shifting bits
52         else if (!SS_n) begin
53             if (bit_index < 11) begin
54                 MOSI = stored_MOSI_array[10 - bit_index];
55                 bit_index++;
56             end
57             else begin
58                 MOSI = 1'b0; // Default after all bits sent
59             end
60         end
61         // Handle SS_n inactive (high) - idle
62         else begin
63             MOSI = 1'b0;
64             bit_index = 0;
65         end
66
67         // Update counter for SS_n timing
68         if (MOSI_array[10:8] inside {3'b000, 3'b001 ,3'b110}) begin
69             // 13 cycle period
70             if (count >= 12) begin
71                 count = 0;
72             end
73             else count++;
74         end
75         else if (MOSI_array[10:8] inside { 3'b111}) begin
76             // 23 cycle period
77             if (count >= 22) begin
78                 count = 0;
79             end
80             else count++;
81         end
82         else begin
83             // Default: 13 cycle period
84             if (count >= 12) begin
85                 count = 0;
86             end
87             else count++;
88         end
89         // Update prev_SS_n for next cycle
90         prev_SS_n = SS_n;
91     endfunction
92 endclass
93 endpackage

```

Wrapper configuration object:

```
1 package wrapper_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class wrapper_config extends uvm_object;
5     `uvm_object_utils(wrapper_config)
6     virtual wrapper_if wrapper_vif ;
7     uvm_active_passive_enum is_active;
8     function new(string name = "wrapper_config");
9       super.new(name) ;
10    endfunction
11  endclass
12 endpackage
```

Wrapper write_only seq:

```
1 package wrapper_write_only_seq_pkg ;
2   import wrapper_seq_item_pkg::*;
3   import uvm_pkg::*;
4   `include "uvm_macros.svh"
5
6   class wrapper_write_only_seq extends uvm_sequence #(wrapper_seq_item) ;
7     `uvm_object_utils(wrapper_write_only_seq)
8
9     wrapper_seq_item seq_item ;
10
11    function new (string name = "wrapper_write_only_seq");
12      super.new(name);
13    endfunction
14
15    task body();
16      seq_item = wrapper_seq_item::type_id::create("seq_item");
17      repeat(90000) begin
18        start_item(seq_item);
19        seq_item.c_read_only.constraint_mode(0);
20        seq_item.c_read_write.constraint_mode(0);
21        seq_item.c_write_only.constraint_mode(1);
22        assert (seq_item.randomize());
23        finish_item(seq_item);
24      end
25    endtask
26  endclass
27 endpackage
```

Wrapper raed_only seq:

```
1 package wrapper_read_only_seq_pkg;
2     import wrapper_seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class wrapper_read_only_seq extends uvm_sequence #(wrapper_seq_item);
7         `uvm_object_utils(wrapper_read_only_seq)
8         wrapper_seq_item seq_item;
9
10    function new(string name = "wrapper_read_only_seq");
11        super.new(name);
12    endfunction
13
14    task body();
15        seq_item = wrapper_seq_item::type_id::create("seq_item");
16        repeat(10000) begin
17            start_item(seq_item);
18            seq_item.c_write_only.constraint_mode(0);
19            seq_item.c_read_write.constraint_mode(0);
20            seq_item.c_read_only.constraint_mode(1);
21            assert(seq_item.randomize());
22            finish_item(seq_item);
23        end
24    endtask
25 endclass
26 endpackage
```

Wrapper raed_write seq:

```
● ● ●

1 package wrapper_read_write_seq_pkg ;
2     import wrapper_seq_item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class wrapper_read_write_seq extends uvm_sequence #(wrapper_seq_item) ;
7         `uvm_object_utils(wrapper_read_write_seq)
8         wrapper_seq_item seq_item ;
9
10    function new (string name = "wrapper_read_write_seq");
11        super.new(name);
12    endfunction
13
14    task body();
15        seq_item = wrapper_seq_item::type_id::create("seq_item");
16        repeat(10000) begin
17            start_item(seq_item);
18            seq_item.c_read_only.constraint_mode(0);
19            seq_item.c_write_only.constraint_mode(0);
20            seq_item.c_read_write.constraint_mode(1);
21            assert (seq_item.randomize());
22            finish_item(seq_item);
23        end
24    endtask
25 endclass
26 endpackage
```

Wrapper sequencer file :

```
● ● ●

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

Wrapper env file :

```
1 package wrapper_env_pkg;
2 import wrapper_scoreboard_pkg::*;
3 import wrapper_agent_pkg::*;
4 import wrapper_coverage_pkg::*;
5 import uvm_pkg::*;
6 `include "uvm_macros.svh"
7
8 class wrapper_env extends uvm_env;
9
10   `uvm_component_utils(wrapper_env)
11
12   wrapper_agent agt ;
13   wrapper_coverage cov ;
14   wrapper_scoreboard sb ;
15
16   function new (string name = "wrapper_env" , uvm_component parent = null);
17     super.new(name , parent) ;
18   endfunction
19
20   function void build_phase (uvm_phase phase);
21     super.build_phase(phase) ;
22     agt = wrapper_agent::type_id::create("agt",this);
23     cov = wrapper_coverage::type_id::create("cov",this);
24     sb = wrapper_scoreboard::type_id::create("sb",this);
25   endfunction
26
27   function void connect_phase (uvm_phase phase);
28     super.connect_phase(phase) ;
29     agt.agt_ap.connect(sb.sb_export);
30     agt.agt_ap.connect(cov.cov_export);
31   endfunction
32
33 endclass
34 endpackage
```

Wrapper agent file :

```
● ● ●
1 package wrapper_agent_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_sequencer_pkg::*;
5   import wrapper_driver_pkg::*;
6   import wrapper_config_pkg::*;
7   import wrapper_monitor_pkg::*;
8   import wrapper_seq_item_pkg::*;

9
10  class wrapper_agent extends uvm_agent;
11    `uvm_component_utils(wrapper_agent)
12    wrapper_sequencer sqr ;
13    wrapper_driver drv ;
14    wrapper_monitor mon;
15    wrapper_config wrapper_cfg;
16    uvm_analysis_port #(wrapper_seq_item) agt_ap;
17
18    function new(string name = "wrapper_agent" , uvm_component parent = null );
19      super.new(name, parent);
20    endfunction
21
22    function void build_phase (uvm_phase phase);
23      super.build_phase(phase);
24      if (!uvm_config_db #(wrapper_config)::get(this, "", "CFG", wrapper_cfg)) begin
25        `uvm_fatal("build_phase", "AGENT - unable to get configuration object");
26      end
27      sqr = wrapper_sequencer::type_id::create("sqr",this);
28      drv = wrapper_driver::type_id::create("drv",this);
29      mon = wrapper_monitor::type_id::create("mon",this);
30      agt_ap = new("agt_ap",this);
31    endfunction
32
33    function void connect_phase(uvm_phase phase);
34      super.connect_phase(phase);
35      drv.wrapper_vif = wrapper_cfg.wrapper_vif;
36      mon.wrapper_vif = wrapper_cfg.wrapper_vif;
37      drv.seq_item_port.connect(sqr.seq_item_export);
38      mon.mon_ap.connect(agt_ap);
39    endfunction
40
41
42  endclass
43 endpackage
```

Wrapper driver file :

```
1 package wrapper_driver_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import wrapper_seq_item_pkg::*;
5
6   class wrapper_driver extends uvm_driver #(wrapper_seq_item);
7     `uvm_component_utils(wrapper_driver)
8     virtual wrapper_if wrapper_vif;
9     wrapper_seq_item stim_seq_item;
10
11   function new (string name = "wrapper_driver", uvm_component parent = null);
12     super.new(name, parent);
13   endfunction
14
15   task run_phase(uvm_phase phase);
16     super.run_phase(phase);
17     forever begin
18       stim_seq_item = wrapper_seq_item::type_id::create("stim_seq_item");
19       seq_item_port.get_next_item(stim_seq_item);
20       wrapper_vif.rst_n=stim_seq_item.rst_n;
21       wrapper_vif.SS_n=stim_seq_item.SS_n;
22       wrapper_vif.MOSI=stim_seq_item.MOSI;
23       @(negedge wrapper_vif.clk);
24       seq_item_port.item_done();
25       `uvm_info("run_phase", stim_seq_item.convert2string_stimuls(), UVM_HIGH);
26     end
27   endtask
28 endclass
29 endpackage
```

Wrapper monitor file :

```
● ○ ●
1 package wrapper_monitor_pkg;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import wrapper_seq_item_pkg::*;
5
6     class wrapper_monitor extends uvm_monitor;
7         `uvm_component_utils(wrapper_monitor)
8         virtual wrapper_if wrapper_vif;
9         wrapper_seq_item rsp_seq_item;
10        uvm_analysis_port #(wrapper_seq_item) mon_ap;
11
12        function new (string name = "wrapper_monitor", uvm_component parent = null);
13            super.new(name, parent);
14        endfunction
15
16        function void build_phase(uvm_phase phase);
17            super.build_phase(phase);
18            mon_ap = new("mon_ap",this);
19        endfunction
20
21        task run_phase(uvm_phase phase);
22            super.run_phase(phase);
23            forever begin
24                rsp_seq_item = wrapper_seq_item::type_id::create("rsp_seq_item");
25                rsp_seq_item.rst_n=wrapper_vif.rst_n;
26                rsp_seq_item.SS_n=wrapper_vif.SS_n;
27                rsp_seq_item.MOSI=wrapper_vif.MOSI;
28                rsp_seq_item.MISO=wrapper_vif.MISO;
29                rsp_seq_item.MISO_ref=wrapper_vif.MISO_ref;
30                @(negedge wrapper_vif.clk);
31                mon_ap.write(rsp_seq_item);
32            end
33        endtask
34    endclass
35 endpackage
```

Wrapper scoreboard:

```
● ● ●  
1 package wrapper_scoreboard_pkg;  
2     import uvm_pkg::*;  
3     `include "uvm_macros.svh"  
4         import wrapper_seq_item_pkg::*;  
5  
6     class wrapper_scoreboard extends uvm_scoreboard;  
7         `uvm_component_utils(wrapper_scoreboard)  
8             uvm_analysis_export #(wrapper_seq_item) sb_export;  
9             uvm_tlm_analysis_fifo #(wrapper_seq_item) sb_fifo;  
10            wrapper_seq_item seq_item_sb;  
11  
12            int error_count = 0;  
13            int correct_count = 0;  
14  
15            logic MISO_ref;  
16  
17            function new (string name = "wrapper_scoreboard" , uvm_component parent = null);  
18                super.new(name , parent) ;  
19            endfunction  
20  
21            function void report_phase(uvm_phase phase);  
22                super.report_phase(phase);  
23                `uvm_info("SCOREBOARD", $sformatf("Simulation Summary:  
24                    correct_count=  
25 , error_endfunction  
26 ",correct_count, error_count), UVM_NONE)  
27                function void build_phase (uvm_phase phase);  
28                    super.build_phase(phase);  
29                    sb_export = new("sb_export",this);  
30                    sb_fifo  = new("sb_fifo",this);  
31                endfunction  
32  
33                function void connect_phase(uvm_phase phase);  
34                    super.connect_phase(phase);  
35                    sb_export.connect(sb_fifo.analysis_export);  
36                endfunction  
37  
38                task run_phase (uvm_phase phase);  
39                    super.run_phase(phase);  
40                    forever begin  
41                        sb_fifo.get(seq_item_sb);  
42                        if (seq_item_sb.MISO != seq_item_sb.MISO_ref) begin  
43                            `uvm_error("run_phase" , "comaprison fail");  
44                            error_count++;  
45                        end  
46                        else correct_count++;  
47                    end  
48                endtask  
49            endclass  
50  
51        endpackage
```

Wrapper coverage file :

```
1 package wrapper_coverage_pkg;
2     import uvm_pkg::*;
3     import wrapper_seq_item_pkg::*;
4     `include "uvm_macros.svh"
5
6     class wrapper_coverage extends uvm_component;
7         `uvm_component_utils(wrapper_coverage)
8         uvm_analysis_export #(wrapper_seq_item) cov_export;
9         uvm_tlm_analysis_fifo #(wrapper_seq_item) cov_fifo;
10        wrapper_seq_item seq_item_cov;
11
12        function new (string name = "wrapper_coverage" , uvm_component parent = null);
13            super.new(name , parent) ;
14        endfunction
15
16        function void build_phase (uvm_phase phase);
17            super.build_phase(phase);
18            cov_export = new("cov_export",this);
19            cov_fifo   = new("cov_fifo",this);
20        endfunction
21
22        function void connect_phase(uvm_phase phase);
23            super.connect_phase(phase);
24            cov_export.connect(cov_fifo.analysis_export);
25        endfunction
26
27        task run_phase (uvm_phase phase);
28            super.run_phase(phase);
29            forever begin
30                cov_fifo.get(seq_item_cov);
31            end
32        endtask
33    endclass
34 endpackage
```

Do file :

```
1 vlib work
2 vlog -f src_files.list +define+SIM +cover +covercells +coveropt
3 vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
4
5 add wave /top/wrapperif/*
6 add wave -r /top/DUT1/SLAVE_instance/cs
7 add wave -r /top/DUT1/SLAVE_instance/ns
8 add wave -r /top/DUT1/RAM_instance/MEM
9 add wave /top/DUT1/assert__p_reset_outputs_inactive
10 add wave /top/DUT1/assert__stable_miso
11
12
13 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
```

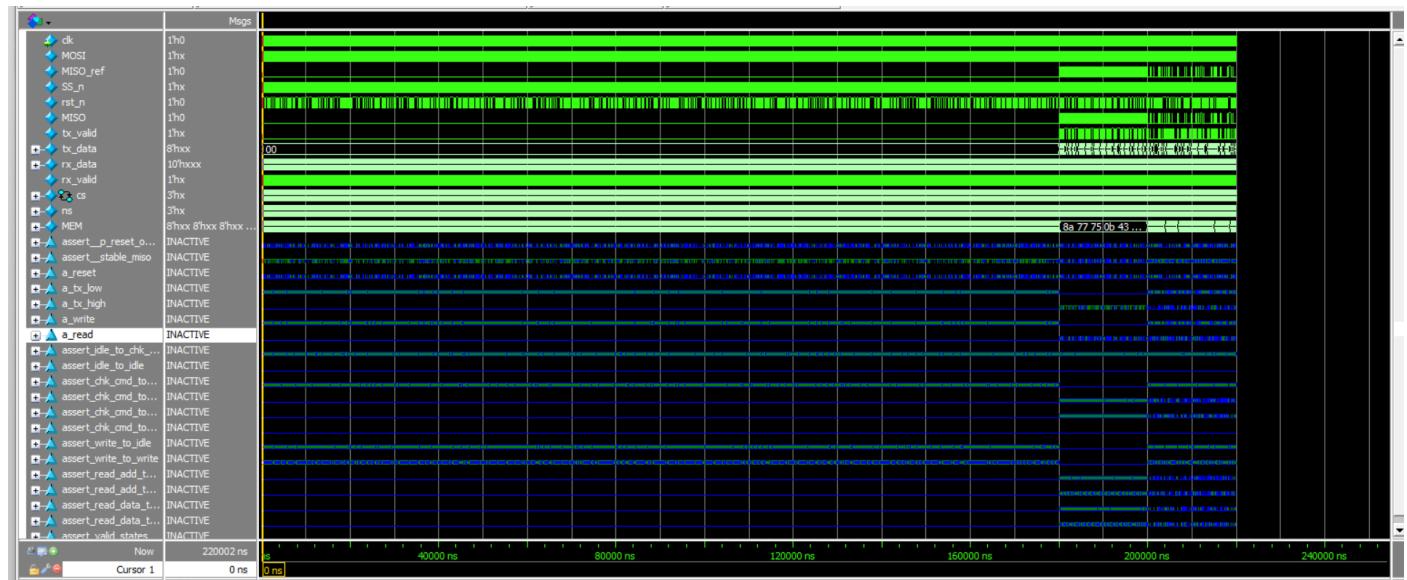
Third:Snippets from Simulation results:

Transcript:

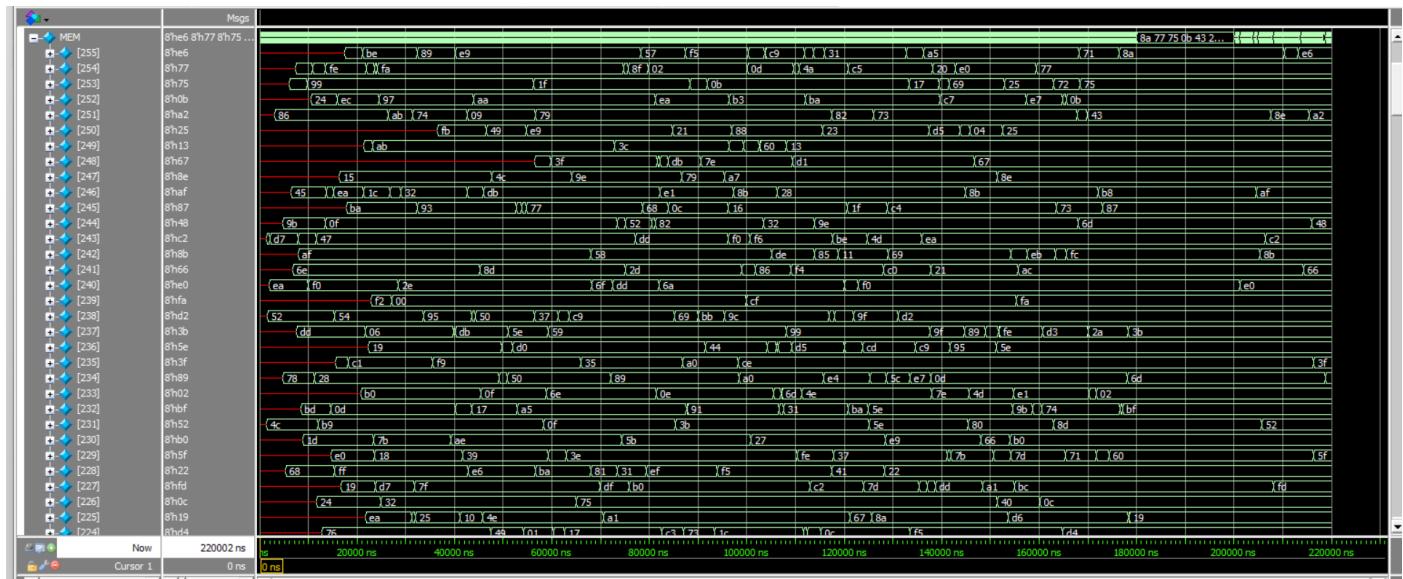
```
# **** UVM Report Summary ****
#   correct_count=110001, error_count=0
# UVM_INFO RAM_scoreboard.sv(53) @ 220002: uvm_test_top.env.ram_sb [report_phase] Total successful transactions : 110001
# UVM_INFO RAM_scoreboard.sv(54) @ 220002: uvm_test_top.env.ram_sb [report_phase] Total failed transactions : 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
# [Questasim UVM] 2
# [RNTIST] 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:

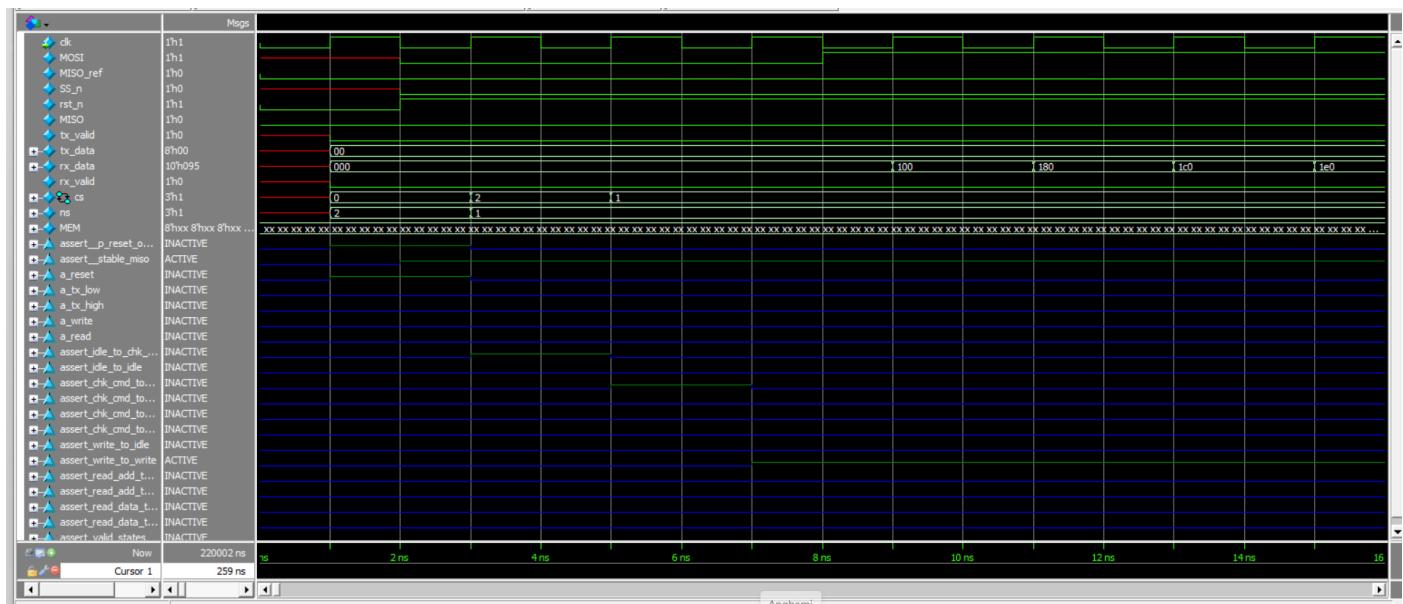


➤ RAM behavior

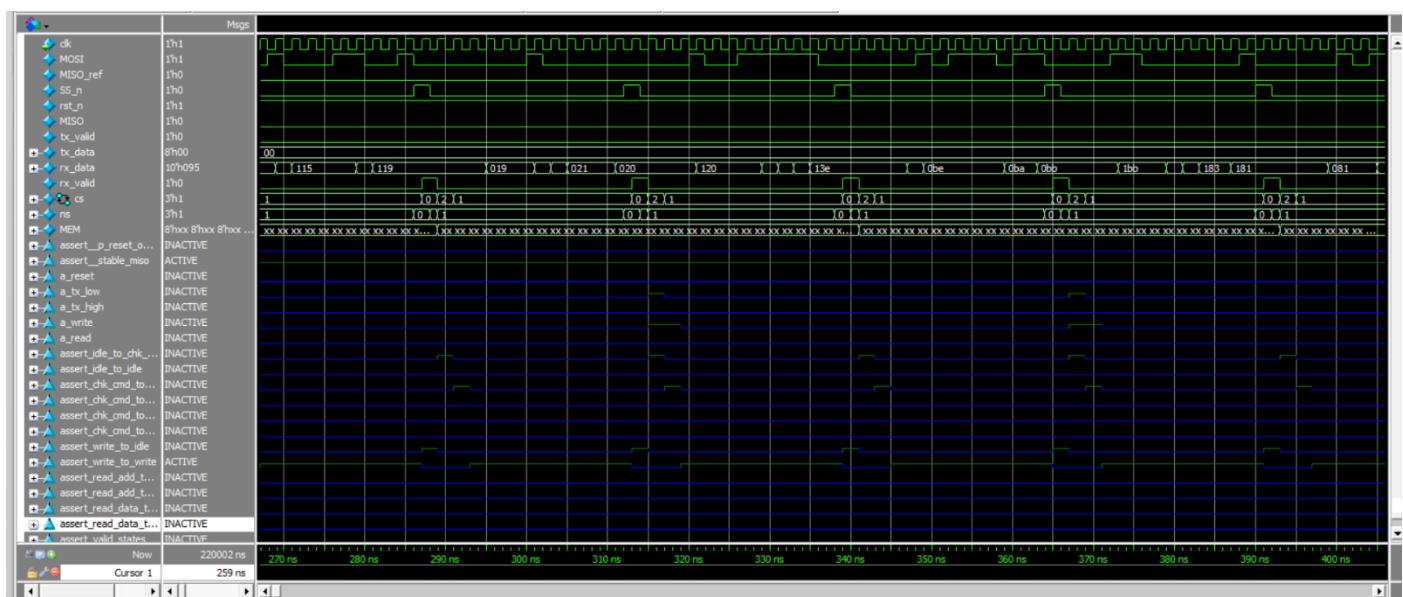
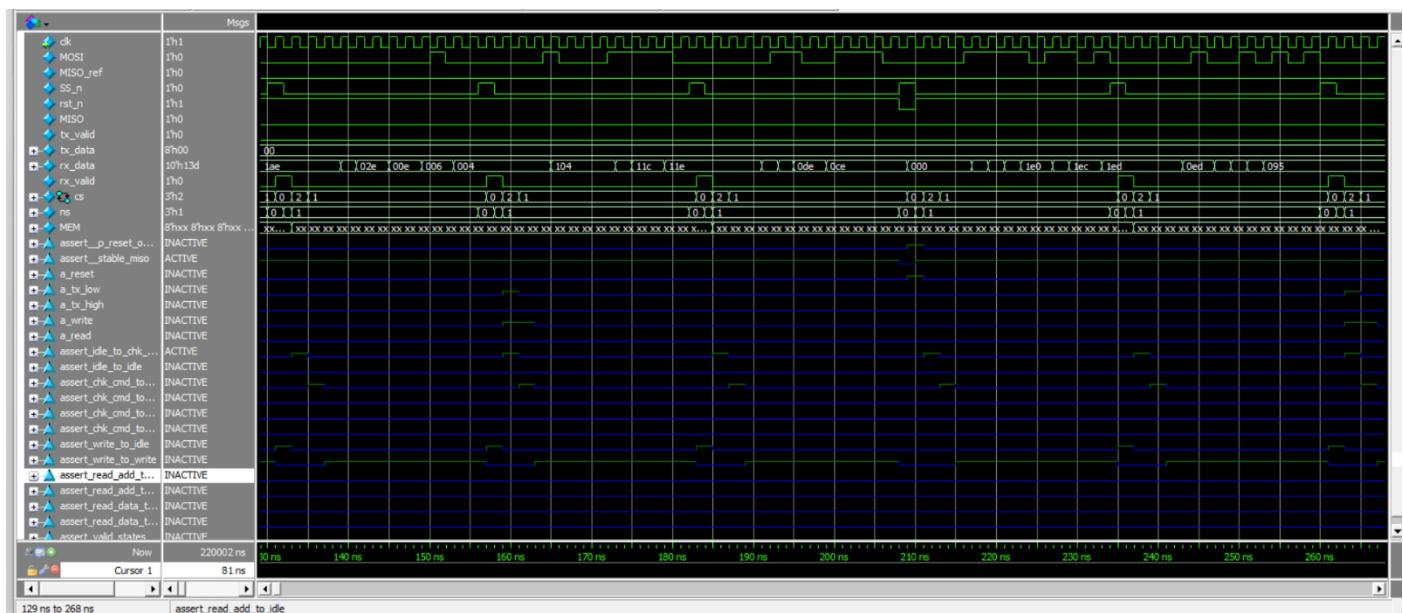
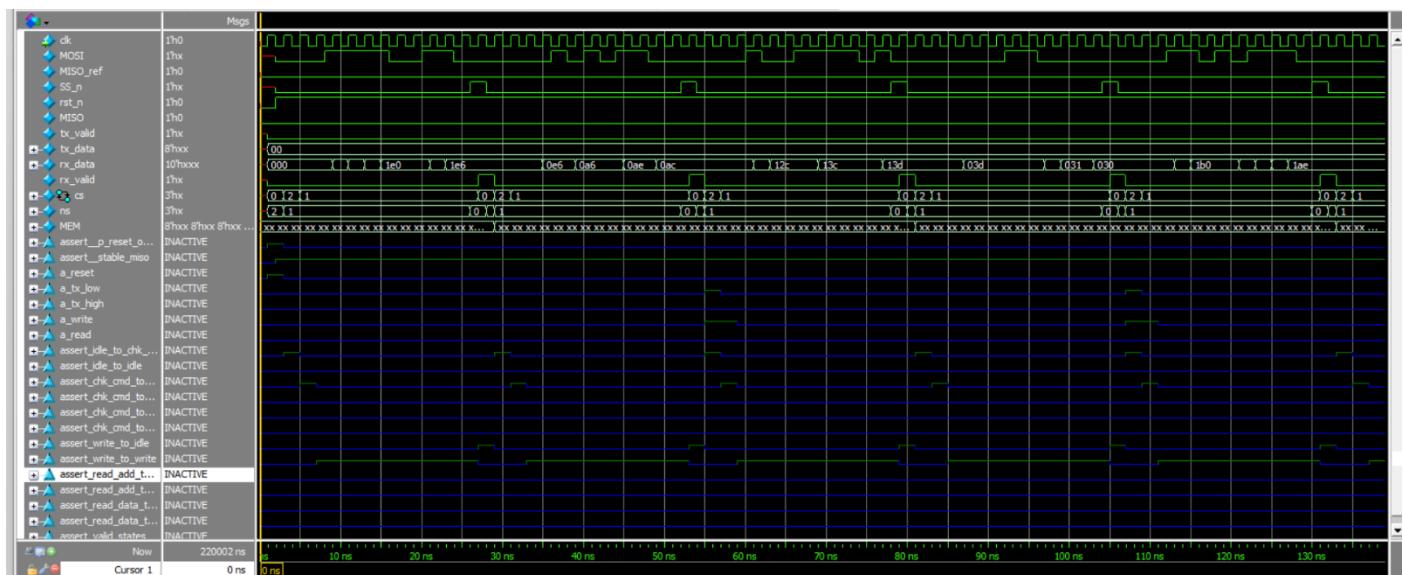


During the time interval from **180,000 ns to 200,000 ns**, the design operates exclusively in a **read-only sequence**, resulting in **no modifications to the RAM content**. This confirms that memory integrity is preserved during read operations, with data being accessed but not altered.

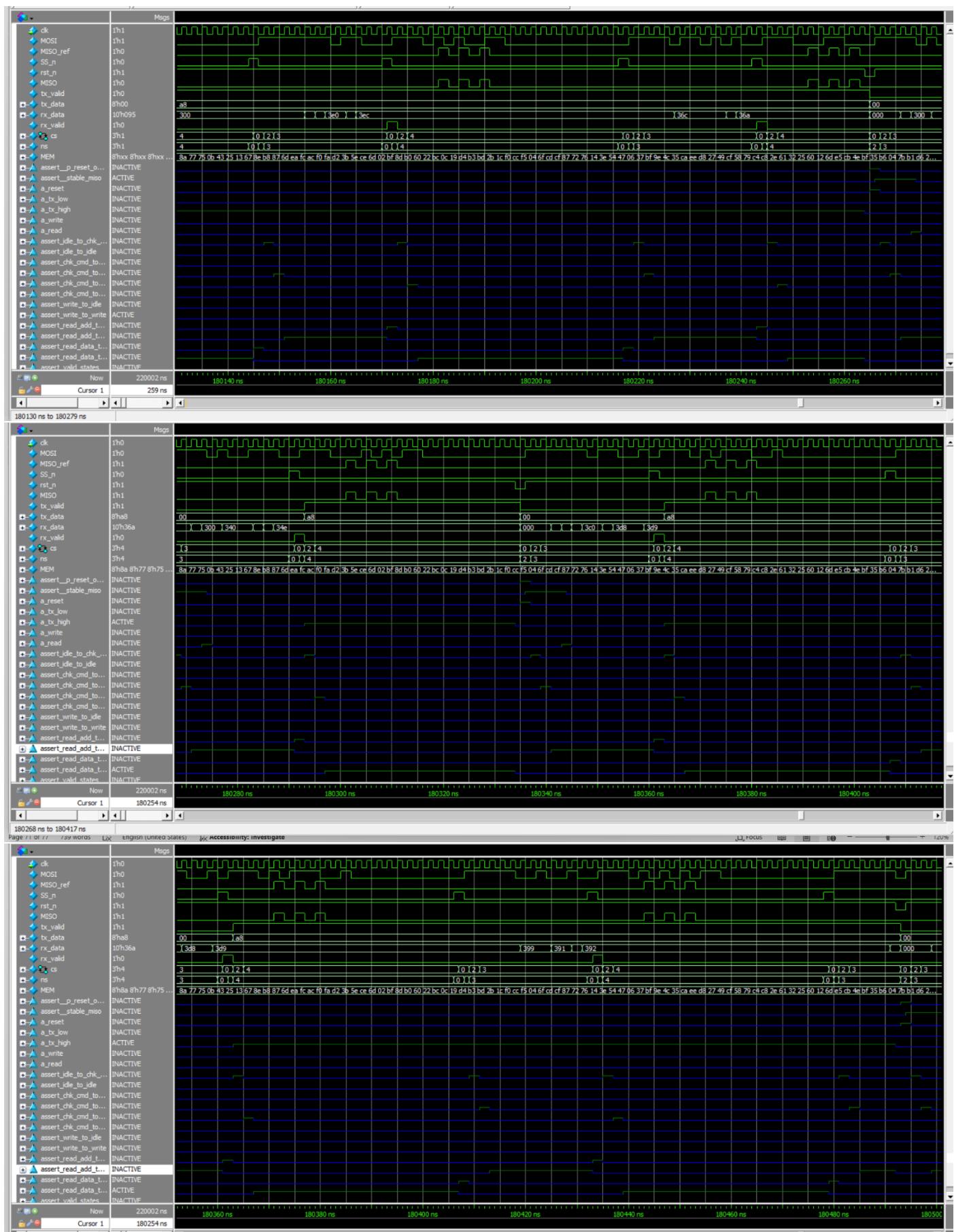
➤ Rest sequence:



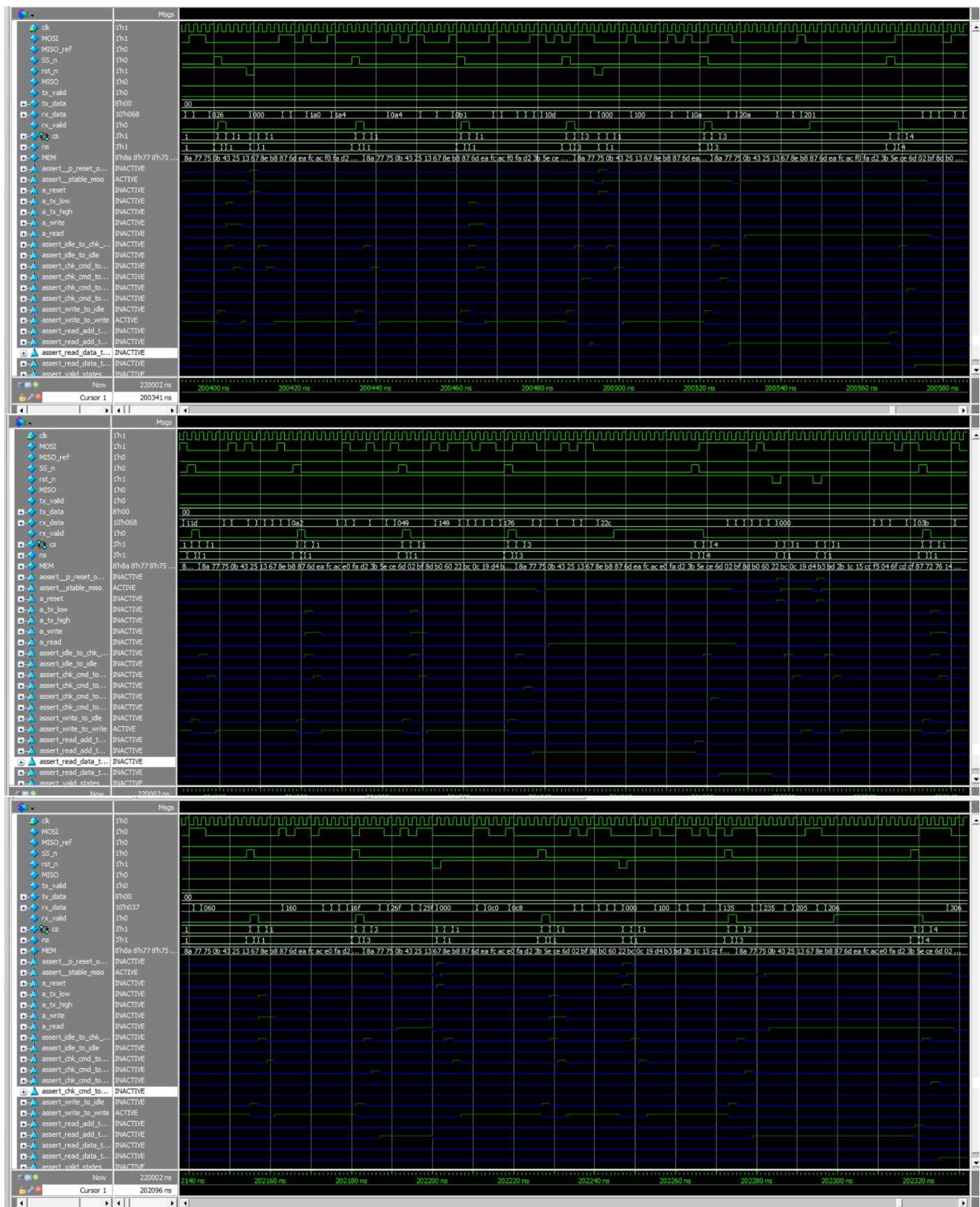
➤ Write only seq :



➤ 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	59525	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/cover_p_reset_outputs_inactive	SVA	✓	Off	1179	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/RAM_instance/RAM_assertion/c_reset	SVA	✓	Off	1179	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/RAM_instance/RAM_assertion/c_tx_low	SVA	✓	Off	3181	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/RAM_instance/RAM_assertion/c_tx_high	SVA	✓	Off	55	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/RAM_instance/RAM_assertion/c_wrlt	SVA	✓	Off	3148	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/RAM_instance/c_read	SVA	✓	Off	3052	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_idk_to_chk_cmd	SVA	✓	Off	8530	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_chk_cmd_to_write	SVA	✓	Off	7611	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_addr	SVA	✓	Off	399	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_chk_cmd_to_read_data	SVA	✓	Off	423	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_write_to_idle	SVA	✓	Off	6782	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_read_add_to_idle	SVA	✓	Off	336	1	Unl...	1	100%	✓	0	0	0	0 ns	0
▲ /top/DUT1/SLAVE_instance/cover_read_data_to_idle	SVA	✓	Off	343	1	Unl...	1	100%	✓	0	0	0	0 ns	0

<code>\top(DUT)\assert_p_reset_outputs_inactive</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge wrapper.dki ... ✓
<code>\top(DUT)\assert_stable_miso</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@wrapendif.dki disable ... ✓
<code>\top(DUT)\RAM_instance/RAM.assertion/a_reset</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk !rst_n! ... ✓
<code>\top(DUT)\RAM_instance/RAM.assertion/a_tx_low</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\RAM_instance/RAM.assertion/a_tx_high</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\RAM_instance/RAM.assertion/a_write</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\RAM_instance/RAM.assertion/a_read</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_idle_to_chk_cmd</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_idle_to_idle</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_chk_cmd_to_write</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_chk_cmd_to_read_add</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_chk_cmd_to_read_data</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_chk_cmd_to_idle</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_write_to_idle</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_write_to_write</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_read_add_to_idle</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SAVE_instance/assert_read_add_to_read_add</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SLAVE_instance/assert_read_data_to_idle</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SLAVE_instance/assert_read_data_to_read_data</code>	Concurrent	SVA	on	0	-	0B	0B	0 ns	0 off	assert(@posedge clk disable iff ... ✓
<code>\top(DUT)\SLAVE_instance/assert_valid_states</code>	Concurrent	SVA	on	0	-	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 Unit	Design UnitType	Lang	File (Line)	Hits	Status
/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:
-----
```

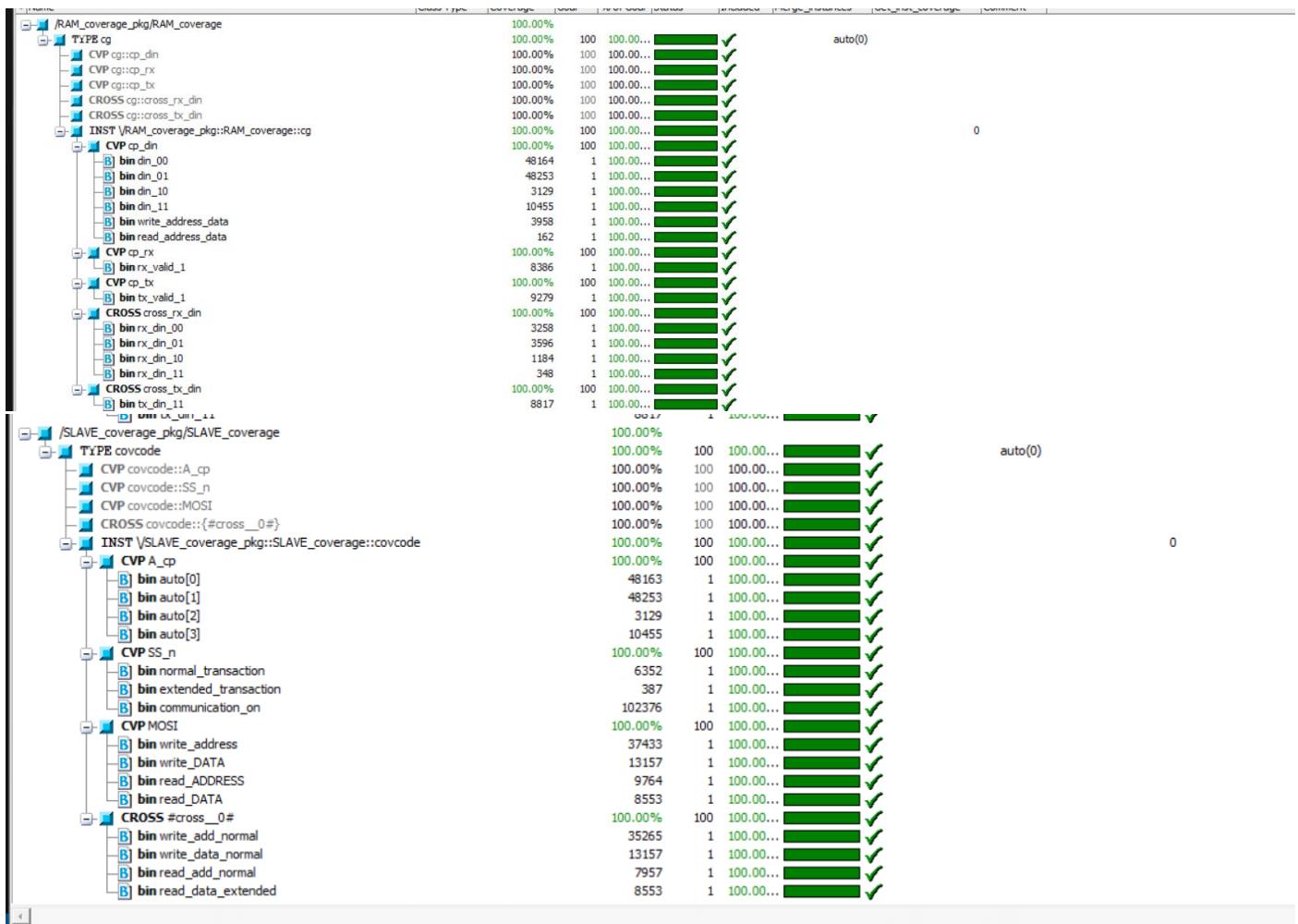
Name	Design Unit	Design UnitType	Lang	File(Line)	Hits	Status
/top/DUT1/RAM_instance/RAM_assertion/c_reset	RAM_sva	Verilog	SVA	RAM_assertions.sv(15)	1179	Covered
/top/DUT1/RAM_instance/RAM_assertion/c_tx_low	RAM_sva	Verilog	SVA	RAM_assertions.sv(24)	3181	Covered
/top/DUT1/RAM_instance/RAM_assertion/c_tx_high	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 Unit	Design UnitType	Lang	File(Line)	Hits	Status
/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:

The screenshot shows a code editor window with the file "SPI_slave.sv" open. The code is a Verilog module for a SPI slave. The entire code is covered, indicated by numerous green checkmarks placed vertically along the left margin of the code lines. The code itself is as follows:

```
20 always @ (posedge clk) begin
22 cs <= IDLE;
25 cs <= ns;
29 always @ (*) begin
33 ns = IDLE;
35 ns = CHK_CMD;
42 ns = WRITE;
45 ns = READ_DATA;
47 ns = READ_ADD;
53 ns = IDLE;
55 ns = WRITE;
59 ns = IDLE;
61 ns = READ_ADD;
65 ns = IDLE;
67 ns = READ_DATA;
72 always @ (posedge clk) begin
74 rx_data <= 0;
75 rx_valid <= 0;
76 received_address <= 0;
77 MISO <= 0;
78 counter <= 0 ;
83 rx_valid <= 0;
86 counter <= 10;
90 rx_data[counter-1] <= MOSI;
91 counter <= counter - 1;
94 rx_valid <= 1;
99 rx_data[counter-1] <= MOSI;
100 counter <= counter - 1;
103 rx_valid <= 1;
104 received_address <= 1;
109 rx_valid <= 0;
111 MISO <= tx_data[counter-1];
112 counter <= counter - 1;
115 received_address <= 0;
120 rx_data[counter-1] <= MOSI;
121 counter <= counter - 1;
122 rx_valid <= 0 ;
125 rx_valid <= 1;
126 counter <= 9;
```

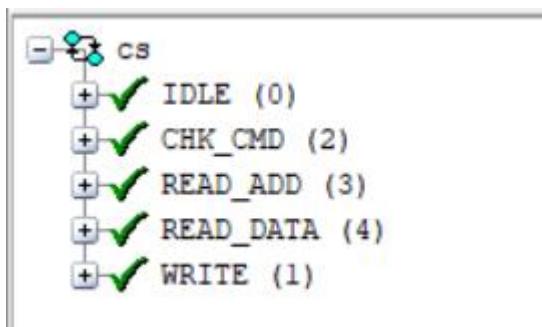
-Slave Branches:

```
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
```

-Slave Toggles:



-Slave FSMs:



-RAM Statements:

```
RAM.v
13 always @ (posedge clk) begin
15   dout      <= 0;
16   tx_valid <= 0;
17   Rd_Addr  <= 0;
18   Wr_Addr  <= 0;
23   2'b00 : Wr_Addr      <= din[7:0];
24   2'b01 : MEM[Wr_Addr] <= din[7:0];
25   2'b10 : Rd_Addr      <= din[7:0];
26   2'b11 : dout          <= MEM[Rd_Addr];//bug fixed
29   tx_valid <= (din[9:8] == 2'b11) ? 1'b1 : 1'b0;
```

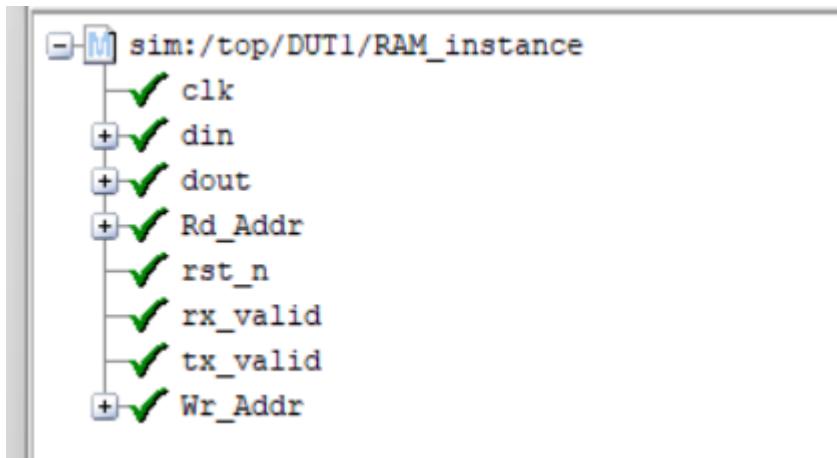
-RAM Branches:

```

14 if (~rst_n) begin
20 else begin
21 if (rx_valid) begin //add begin/end
23 2'b00 : Wr_Addr      <= din[7:0];
24 2'b01 : MEM[Wr_Addr] <= din[7:0];
25 2'b10 : Rd_Addr      <= din[7:0];
26 2'b11 : dout          <= MEM[Rd_Addr];//bug fixed

```

-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 with <code>rc</code> _write_only constraint enabled: If <code>prev_cmd</code> = 000 → next cmd = 001 If <code>prev_cmd</code> = 001 → next cmd = 000	Cross coverage between consecutive commands Verify only 000 → 001 and 001 → 000 transitions occur Count sequence length and verify 90,000 iterations	Scoreboard checks MISO vs MISO_refSVA verifies no read operations occur Monitor validates command sequence ordering
WRAPPER_1	Read-Only Sequence: Only Read Address or Read Data operations are allowed	Constrained randomization with <code>rc</code> _read_only constraint enabled: If <code>prev_cmd</code> = 110 → next cmd = 111 If <code>prev_cmd</code> = 111 → next cmd = 110	Cross coverage between consecutive commands Verify only 110 → 111 and 111 → 110 transitions occur Count sequence length and verify 10,000 iterations Monitor <code>tx_valid</code> assertion for read data	Scoreboard validates MISO data integrity SVA verifies no write operations occur Check MISO stability during non-read operations
WRAPPER_2	Read-Write Mixed Sequence: Random mix of all operation types with realistic transitions	Constrained randomization with <code>rc</code> _read_write constraint: If <code>prev</code> = 000 → next in {000, 001} If <code>prev</code> = 001 → 000(40%) or 110(60%) If <code>prev</code> = 110 → next = 111 If <code>prev</code> = 111 → 000(60%) or 110(40%)	Cross coverage of all valid command transitions Cover address range distribution (0x00-0xFF) Track write-then-read to same address Monitor transition distribution matches constraints Verify 10,000 mixed operations completed	Scoreboard validates all MISO outputs Check data integrity: write then read same address SVA verifies timing for both 13 and 23 cycle periods Validate RAM state consistency
WRAPPER_3	Address Range Coverage: All memory addresses (0x00 to 0xFF) should be accessed	Address randomization with distribution constraint: <code>MOSI_array[7:0]</code> dist {[8'h00:8'hFF] :~ 100}	Coverpoint for all 256 address values Separate bins for boundary addresses: 0x00, 0xFF Track addresses accessed during write vs read	Verify coverage report shows 100% address coverage Check memory initialization and read-back
WRAPPER_4	Reset During Transaction: Reset assertion during active transaction should safely abort operation	Reset constraint with occasional assertion: <code>rst_n</code> dist {1 := 99, 0 := 1}	Cover reset during each state (IDLE, WRITE, CMDCHK, RDADDR, RDDATA) Monitor recovery after reset de-assertion	SVA: <code>!rst_n</code> => (<code>MISO</code> == 0 && <code>rx_valid</code> == 0 && <code>tx_valid</code> == 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 data Cover MISO stable during write operations	SVA: (<code>rx_data[9:8]</code> != 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