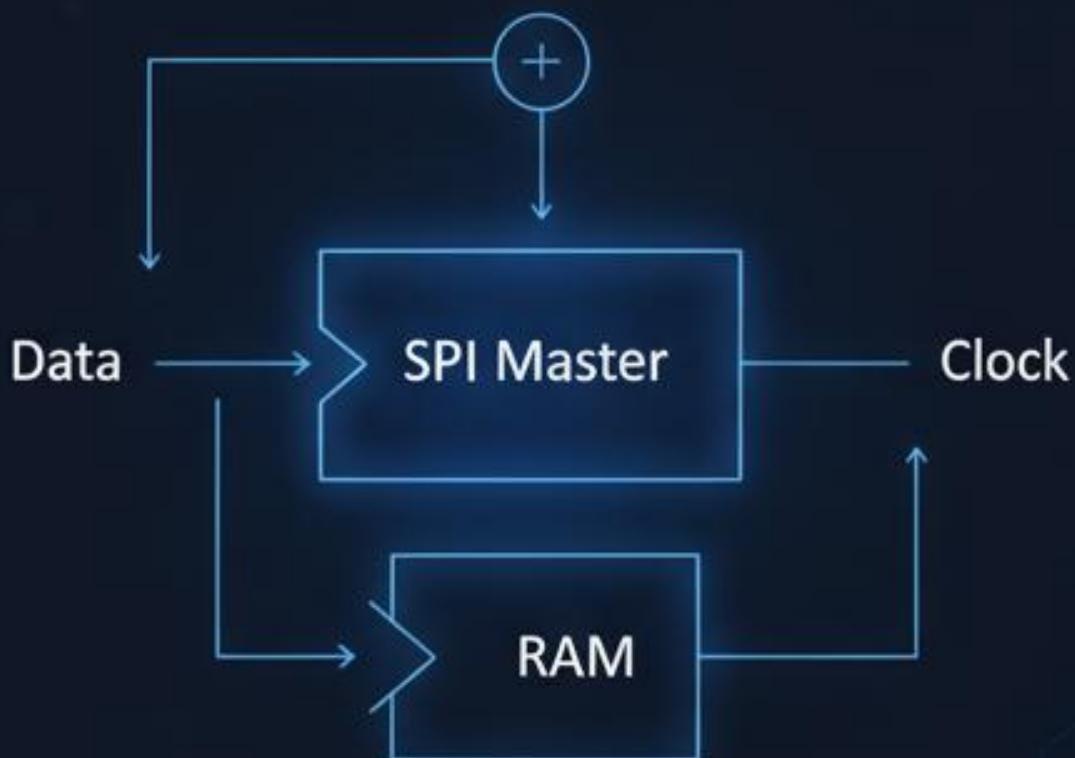


# PROJECT REPORT

## SPI Single Port RAM Verification

### USING UVM METHODOLOGY



Provided by:



Saeed Nabawy  
Mostafa Ahmed Sherif  
Zeyad Abdallah Shaban

## Table of Contents

<b>Introduction</b> .....	6
<b>SPI Slave Verification</b> .....	7
UVM Diagram .....	7
RTL Code.....	9
Bugs are fixed.....	11
Code Snippets .....	12
Top module.....	12
Module Interface .....	12
Golden Model Interface .....	12
Golden Model Module.....	13
SPI_slave_sva .....	14
SPI_RAM_config.....	14
SPI_slave_sequenceitem.....	15
SPI_slave_sequencer.....	16
SPI_slave_RST_sequence .....	16
SPI_slave_random_sequence .....	17
SPI_slave_driver .....	18
SPI_slave_monitor .....	19
SPI_slave_scoreboard .....	20
SPI_slave_coverage .....	21
SPI_slave_agent.....	22
SPI_slave_env.....	23
SPI_slave_test .....	24
QuestaSim Snippets .....	25
Functional Coverage .....	26
Assertion Summary.....	28
Assertion Coverage .....	29
Code Coverage.....	30

<b>RAM Verification .....</b>	36
<b>UVM Diagram .....</b>	36
<b>RTL Code.....</b>	38
<b>Bugs are fixed.....</b>	39
<b>Code Snippets .....</b>	40
<b>Top module.....</b>	40
<b>Module Interface .....</b>	40
<b>Golden Model Interface .....</b>	41
<b>Golden Model Module.....</b>	42
<b>RAM_sva .....</b>	43
<b>RAM_config .....</b>	43
<b>RAM_sequenceitem.....</b>	44
<b>RAM_sequencer.....</b>	45
<b>RAM_RST_SEQUENCE .....</b>	45
<b>RAM_WRITE_ONLY_SEQUENCE .....</b>	46
<b>RAM_WRITE_ONLY_SEQUENCE .....</b>	47
<b>RAM_READ_WRITE_SEQUENCE.....</b>	48
<b>RAM_driver .....</b>	49
<b>RAM_monitor.....</b>	50
<b>RAM_MONITOR_GOLDEN_MODEL.....</b>	51
<b>RAM_scoreboard .....</b>	52
<b>RAM_coverage .....</b>	53
<b>RAM_agent .....</b>	54
<b>SPI_SLAVE_ENV .....</b>	55
<b>SPI_SLAVE_TEST .....</b>	56
<b>QuestaSim Snippets .....</b>	57
<b>Reset_sequence .....</b>	57
<b>Write_only_sequence .....</b>	57
<b>Read_only_sequence.....</b>	58

<b>Read_write_sequence .....</b>	58
<b>Functional Coverage .....</b>	59
<b>Assertion Summary.....</b>	61
<b>Assertion Coverage .....</b>	62
<b>Code Coverage.....</b>	63
<b>Part 3 SPI-Wrapper Verification.....</b>	66
<b>UVM Diagram .....</b>	66
<b>RTL Code.....</b>	68
<b>Code Snippets .....</b>	69
<b>Top module.....</b>	69
<b>Module Interface .....</b>	69
<b>Golden Model Interface .....</b>	70
<b>Golden Model Module.....</b>	70
<b>Wrapper_sva.....</b>	71
<b>SPI_RAM_config “Changed to suit the whole verification process”.....</b>	71
<b>Wrapper_sequenceitem .....</b>	72
<b>Wrapper_sequencer .....</b>	74
<b>Wrapper_RST_sequence.....</b>	74
<b>Wrapper_read_only_sequence .....</b>	75
<b>Wrapper_write_only_sequence.....</b>	75
<b>Wrapper_read_write_sequence .....</b>	76
<b>Wrapper_driver.....</b>	77
<b>Wrapper_monitor .....</b>	78
<b>Wrapper_scoreboard.....</b>	79
<b>Wrapper_agent .....</b>	80
<b>Wrapper_env .....</b>	81
<b>Wrapper_test.....</b>	82
<b>Previous File changes to accommodate for the full environment .....</b>	84
<b>Source Files List.....</b>	86

<b>DO File</b> .....	87
<b>QuestaSim Snippets</b> .....	87
<b>Functional Coverage</b> .....	90
<b>Assertion Summary</b> .....	92
<b>Assertion Coverage</b> .....	93
<b>Code Coverage</b> .....	95

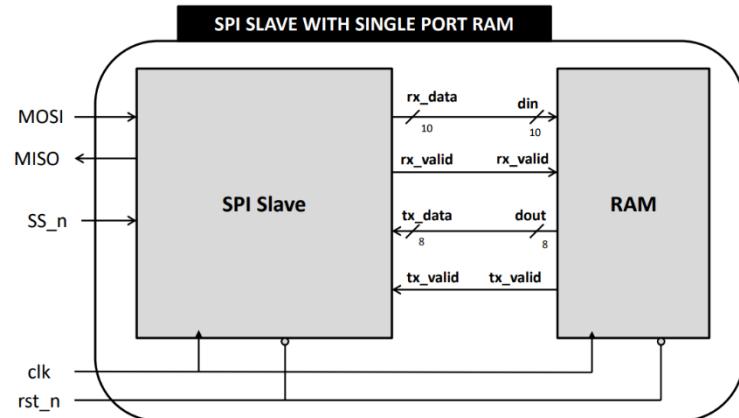
## Introduction

The **SPI Slave with Single Port RAM Verification Project** focuses on verifying the functional correctness and protocol compliance of a Serial Peripheral Interface (SPI) slave module connected to a single-port RAM using the **Universal Verification Methodology (UVM)**. The SPI protocol is a widely used serial communication standard that allows synchronous data exchange between a master and multiple slave devices. In this project, the SPI slave is responsible for receiving and transmitting serial data with the master while interacting internally with a single-port RAM for data storage and retrieval.

The main objective of this verification project is to ensure that the SPI slave correctly performs all **read, write, and command decoding operations** according to the SPI protocol specifications. The slave communicates using four main signals: **MOSI (Master Out Slave In)** for data reception, **MISO (Master In Slave Out)** for data transmission, **CLK (Serial Clock)** for synchronization, and **SS\_n (Slave Select)** for transaction control. The single-port RAM is used as a memory interface, where the data received through the SPI interface is written or from which data is read based on the issued commands.

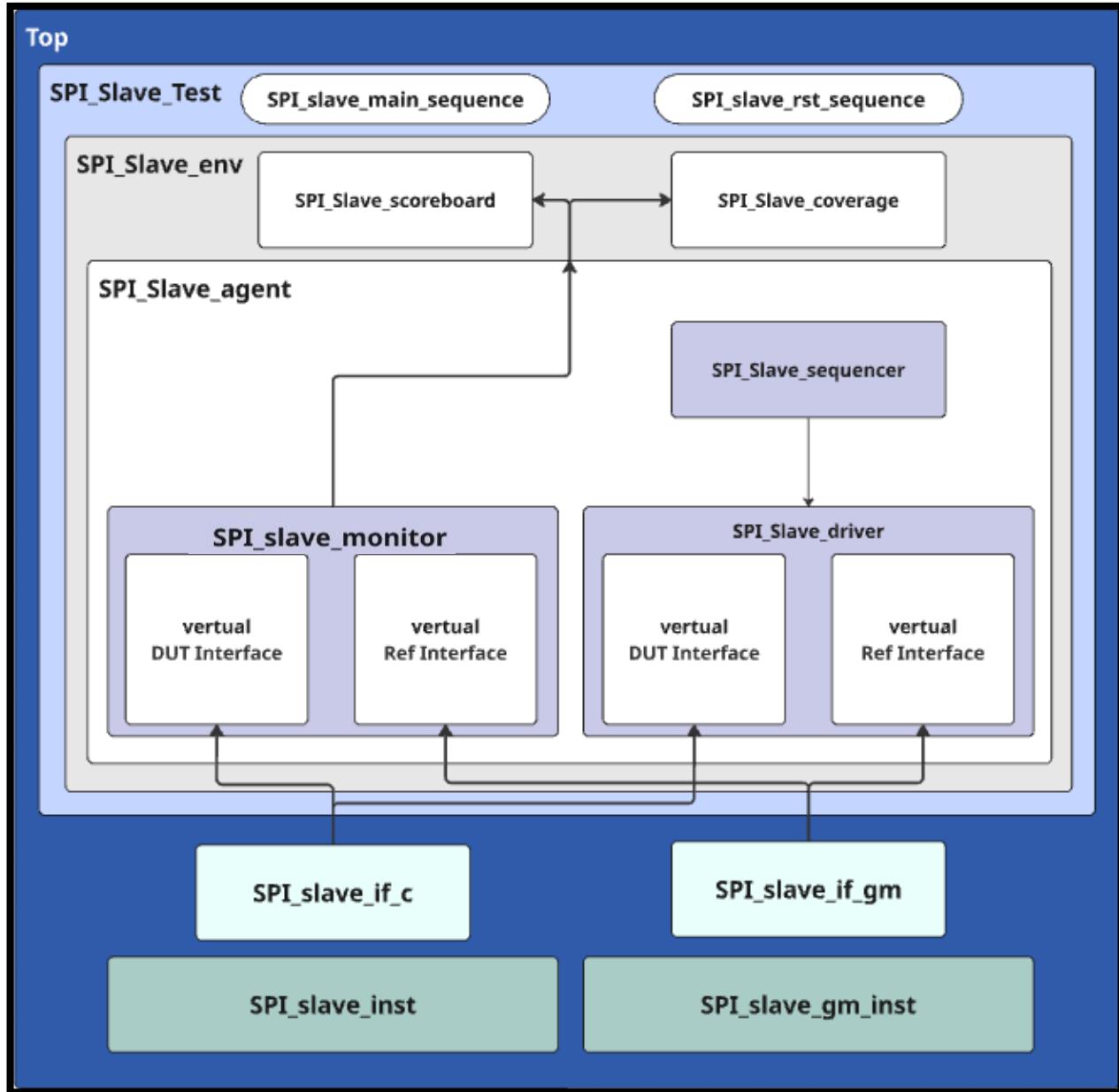
The verification environment was implemented using **SystemVerilog UVM**, providing a modular, reusable, and scalable testbench architecture. The environment includes key UVM components such as a **sequence item, sequencer, driver, monitor, agent, scoreboard, and environment**, all working together to generate stimulus, drive transactions, monitor responses, and compare expected versus actual outputs. Constrained random stimulus and functional coverage were applied to achieve high verification confidence and ensure that all SPI operations and corner cases were thoroughly exercised.

This project demonstrates the full UVM verification flow — from testbench architecture design and stimulus generation to coverage collection and result analysis — ensuring that the **SPI Slave with Single Port RAM** design meets its functional requirements and operates reliably under various conditions.



# SPI Slave Verification

## UVM Diagram



The top module instantiates two interfaces, one for the SPI Slave design and one for its golden reference model. In addition, it instantiates both the SPI Slave DUT and its corresponding reference model, each connected to its respective interface. These interfaces are then stored in the configuration database to be retrieved later during the verification phase. Finally, the simulation is initiated using the “run\_test” function, which begins the SPI Slave test.

The SPI Slave test begins by retrieving these interfaces from the configuration database and storing them in configuration objects that contain their virtual interface handles. The test then creates the SPI Slave environment, as shown in the diagram, and runs the main and reset sequences. These sequences generate constrained sequence items that are sent from the sequencer to the drivers.

Within the SPI Slave environment, three main components are instantiated: the scoreboard, the coverage collector, and the UVM agent.

The SPI Slave scoreboard compares the results obtained from the DUT and the reference model by receiving transaction data through analysis FIFOs and analysis ports. Any mismatches between the DUT and reference transactions are reported as errors.

The SPI Slave coverage collector also connects through analysis ports to receive the same transaction data and ensures that the functional coverage metrics and test goals are met during simulation.

The SPI Slave agent retrieves the configuration object and accordingly instantiates its sequencer and two separate driver components — one for the DUT and another for the golden reference. Each driver receives its respective virtual interfaces from the configuration objects and is responsible for driving stimulus transactions directly to both the DUT and reference interfaces, effectively replacing the need for a dedicated monitor component.

During simulation, the sequencer continuously sends sequence items to both drivers, which in turn generate and apply corresponding stimulus signals to the DUT and the reference model. The resulting transactions are then sent to the scoreboard and coverage collector for comparison and coverage evaluation. Once all test sequences have completed and all verification objectives are met, the final objection is dropped from the test, marking the successful completion of the SPI Slave UVM test.

## RTL Code

```
● ● ●
1 import shared_package::*;
2 module SPI_slave(SPI_slave_inf.dut if_c);
3
4     localparam IDLE      = 3'b000;
5     localparam CHK_CMD   = 3'b001;
6     localparam WRITE     = 3'b010;
7     localparam READ_ADD  = 3'b011;
8     localparam READ_DATA = 3'b100;
9
10    reg [3:0] counter;
11    reg      received_address;
12
13    reg [2:0] cs, ns;
14
15    always @(posedge if_c.clk) begin
16        if (~if_c.rst_n) begin
17            cs <= IDLE;
18        end
19        else begin
20            cs <= ns;
21        end
22        cs_shared <= cs; // shared variable
23        ns_shared <= ns; // shared variable
24    end
25
26    always @(*) begin
27        case (cs)
28            IDLE : begin
29                if (if_c.SS_n)
30                    ns = IDLE;
31                else
32                    ns = CHK_CMD;
33            end
34            CHK_CMD : begin
35                if (if_c.SS_n)
36                    ns = IDLE;
37                else begin
38                    if (~if_c.MOSI)
39                        ns = WRITE;
40                    else begin
41                        if (~received_address) //fix
42                            ns = READ_ADD;
43                        else
44                            ns = READ_DATA;
45                    end
46                end
47            end
48            WRITE : begin
49                if (if_c.SS_n)
50                    ns = IDLE;
51                else
52                    ns = WRITE;
53            end
54            READ_ADD : begin
55                if (if_c.SS_n)
56                    ns = IDLE;
57                else
58                    ns = READ_ADD;
59            end
60            READ_DATA : begin
61                if (if_c.SS_n)
62                    ns = IDLE;
63                else
64                    ns = READ_DATA;
65            end
66        endcase
67    end
```

```

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

```

```

1
2
3 `ifdef SIM
4
5 property p_state_operation_1;
6
7     @(posedge if_c.clk) disable iff(~if_c.rst_n)
8         (cs == IDLE) |=> (cs == IDLE || cs == CHK_CMD)
9 endproperty
10 assert property(p_state_operation_1);
11 cover property(p_state_operation_1);
12
13 property p_state_operation_2;
14
15     @(posedge if_c.clk) disable iff(~if_c.rst_n)
16         (cs == CHK_CMD) |=> (cs == IDLE ||
17         cs == WRITE || cs == READ_ADD || cs == READ_DATA)
18 endproperty
19 assert property(p_state_operation_2);
20 cover property(p_state_operation_2);
21
22 property p_state_operation_3;
23
24     @(posedge if_c.clk) disable iff(~if_c.rst_n)
25         (cs == WRITE) |=> (cs == WRITE || cs == IDLE)
26 endproperty
27 assert property(p_state_operation_3);
28 cover property(p_state_operation_3);
29
30 property p_state_operation_4;
31     @(posedge if_c.clk) disable iff(~if_c.rst_n)
32         (cs == READ_ADD) |=> (cs == READ_ADD || cs == IDLE)
33 endproperty
34 assert property(p_state_operation_4);
35 cover property(p_state_operation_4);
36
37 property p_state_operation_5;
38     @(posedge if_c.clk) disable iff(~if_c.rst_n)
39         (cs == READ_DATA) |=> (cs == READ_DATA || cs == IDLE)
40 endproperty
41 assert property(p_state_operation_5);
42 cover property(p_state_operation_5);
43
44 `endif
45
46 endmodule

```

## Bugs are fixed

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

In the CHC\_CMD State when deciding to the next state according to the received\_address internal signal should when received\_address is 0 the next state will be READ\_ADD and when received\_address is 1 the next state will be READ\_DATA

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

In the READ\_DATA State when all data in the tx\_data pushed in MISO the rx\_valid flag should be raise

# Code Snippets

## Top module

```
1 `include "uvm_macros.svh"
2 import uvm_pkg::*;
3 import SPI_slave_test_pkg::*;
4 module top;
5
6   bit clk;
7   initial begin
8     clk = 0;
9     forever #1 clk = ~clk;
10 end
11
12 SPI_slave_if if_c(.clk(clk));
13 SPI_slave_gm_if if_gm(.clk(clk));
14 SPI_slave SPI_slave_inst(if_c);
15 SPI_Slave_golden_model gm(if_gm);
16 SPI_Slave_sva sva(if_c);
17 // bind SPI_slave SPI_slave_sva sva(if_c);
18 initial begin
19   uvm_config_db #(virtual SPI_slave_if)::set(null, "uvm_test_top", "vif", if_c);
20   uvm_config_db #(virtual SPI_slave_gm_if)::set(null, "uvm_test_top", "vif_gm", if_gm);
21   run_test("SPI_slave_test");
22 end
23 endmodule
24
```

## Module Interface

```
1 interface SPI_slave_if(input clk);
2   logic MOSI, rst_n, SS_n, tx_valid;
3   logic [7:0] tx_data;
4   logic [9:0] rx_data;
5   logic rx_valid, MISO;
6
7   modport dut (
8     input clk, MOSI, rst_n, SS_n, tx_valid,tx_data,
9     output rx_data,rx_valid, MISO
10 );
11
12 modport mon (
13   input clk, MOSI, rst_n, SS_n, tx_valid,tx_data,
14   input rx_data,rx_valid, MISO
15 );
16 endinterface
```

## Golden Model Interface

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

## Golden Model Module

```
1 module SPI_Slave_golden_model(SPI_slave_gm_if#(if_gm);  
2  
3 reg [3:0] counter;  
4 reg received_address;  
5 reg [2:0] cs, ns;  
6  
7 localparam IDLE = 3'b000;  
8 localparam WRITE = 3'b001;  
9 localparam CHK_CMD = 3'b010;  
10 localparam READ_ADD = 3'b011;  
11 localparam READ_DATA = 3'b100;  
12  
13  
14 always @(posedge if_gm.clk) begin  
15   if (~if_gm.rst_n) begin  
16     if_gm.rx_data <= 0;  
17     if_gm.rx_valid <= 0;  
18     received_address <= 0;  
19     if_gm.MISO <= 0;  
20     counter <= 0;  
21   end  
22   else begin  
23     case (cs)  
24       IDLE : begin  
25         if_gm.rx_data <= 1;  
26         if_gm.rx_valid <= 0;  
27       end  
28       CHK_CMD : begin  
29         counter <= 10;  
30       end  
31       WRITE : begin  
32         if (counter > 0) begin  
33           if_gm.rx_data[counter-1] <= if_gm.MOSI;  
34           counter <= counter - 1;  
35         end  
36         else begin  
37           if_gm.rx_valid <= 1;  
38           // counter <= 8; // IGNORE  
39         end  
40       end  
41       READ_ADD : begin  
42         if (counter > 0) begin  
43           if_gm.rx_data[counter-1] <= if_gm.MOSI;  
44           counter <= counter - 1;  
45         end  
46         else begin  
47           if_gm.rx_valid <= 1;  
48           received_address <= 1;  
49         end  
50       end  
51       READ_DATA : begin  
52         if (if_gm.tx_valid) begin  
53           if_gm.rx_valid <= 0;  
54           if (counter > 0) begin  
55             if_gm.MISO <= if_gm.tx_data[counter-1];  
56             counter <= counter - 1;  
57           end  
58           else begin  
59             received_address <= 0;  
60             if_gm.rx_valid <= 1;  
61           end  
62         end  
63       end  
64     end  
65   end  
66 endmodule
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68 endmodule
```

## SPI\_slave\_sva

```
1 module SPI_slave_sva(SPI_slave_inf.dut if_c);
2
3     localparam IDLE      = 3'b000;
4     localparam CHK_CMD   = 3'b001;
5     localparam WRITE     = 3'b010;
6     localparam READ_ADD  = 3'b011;
7     localparam READ_DATA = 3'b100;
8
9
10    property p_reset;
11        @ (posedge if_c.clk)
12            (~if_c.rst_n |=> (if_c.MISO == 'b0 && if_c.rx_valid == 'b0 && if_c.rx_data == 'b0))
13    endproperty
14    assert property(p_reset);
15    cover property(p_reset);
16
17    property p_rx_valid_SS_n_operation_write_addr;
18        @ (posedge if_c.clk) disable iff (~if_c.rst_n)
19            (if_c.SS_n ##1 !if_c.SS_n ##1 (!if_c.MOSI ##1 !if_c.MOSI ##1 !if_c.MOSI)) |-> ##10 ($rose(if_c.rx_valid)) ##[1:$] if_c.SS_n;
20    endproperty
21    assert property(p_rx_valid_SS_n_operation_write_addr);
22    cover property(p_rx_valid_SS_n_operation_write_addr);
23
24    property p_rx_valid_SS_n_operation_write_data;
25        @ (posedge if_c.clk) disable iff (~if_c.rst_n)
26            (if_c.SS_n ##1 !if_c.SS_n ##1 (!if_c.MOSI ##1 !if_c.MOSI ##1 if_c.MOSI)) |-> ##10 ($rose(if_c.rx_valid)) ##[1:$] if_c.SS_n;
27    endproperty
28    assert property(p_rx_valid_SS_n_operation_write_data);
29    cover property(p_rx_valid_SS_n_operation_write_data);
30
31    property p_rx_valid_SS_n_operation_read_addr;
32        @ (posedge if_c.clk) disable iff (~if_c.rst_n)
33            (if_c.SS_n ##1 !if_c.SS_n ##1 if_c.MOSI ##1 if_c.MOSI ##1 !if_c.MOSI) |-> ##10 ($rose(if_c.rx_valid)) ##[1:$] if_c.SS_n;
34    endproperty
35    assert property(p_rx_valid_SS_n_operation_read_addr);
36    cover property(p_rx_valid_SS_n_operation_read_addr);
37
38    property p_rx_valid_SS_n_operation_read_data;
39        @ (posedge if_c.clk) disable iff (~if_c.rst_n)
40            (if_c.SS_n ##1 !if_c.SS_n ##1 if_c.MOSI ##1 if_c.MOSI ##1 if_c.MOSI) |-> ##10 ($rose(if_c.rx_valid)) ##[1:$] if_c.SS_n;
41    endproperty
42    assert property(p_rx_valid_SS_n_operation_read_data);
43    cover property(p_rx_valid_SS_n_operation_read_data);
44
45 endmodule : SPI_slave_sva
```

## SPI\_RAM\_config

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

## SPI\_slave\_sequenceitem

```
1 package SPI_slave_sequenceitem_pkg;
2 import uvm_pkg::*;
3 import shared_package::*;
4 `include "uvm_macros.svh"
5
6 class SPI_slave_sequenceitem extends uvm_sequence_item;
7   `uvm_object_utils(SPI_slave_sequenceitem)
8   logic MOSI;
9   rand logic [10:0] MOSI_arr;
10  logic [10:0] old_MOSI_arr = 3'b111;
11  logic [2:0] next_mosi = 3'b000;
12  rand logic SS_n = 1;
13  rand logic rst_n, tx_valid;
14  rand logic [7:0] tx_data;
15  logic [9:0] rx_data;
16  logic rx_valid, MISO;
17
18  int counter = 0;
19  function new(string name = "SPI_slave_sequenceitem");
20    super.new(name);
21  endfunction
22
23  function string convert2string();
24    return $sformatf("SPI_slave_sequenceitem: %s: MOSI=%b,
25      rst_n=%b, SS_n=%b, tx_valid=%b, tx_data=%b, rx_data=%b,
26      rx_valid=%b, MISO=%b",super.convert2string(), MOSI, rst_n,
27      SS_n, tx_valid, tx_data, rx_data, rx_valid, MISO);
28  endfunction
29
30  function void pre_randomize();
31    if(SS_n == 0) begin
32      MOSI_arr.rand_mode(0);
33    end
34    else begin
35      MOSI_arr.rand_mode(1);
36    end
37  endfunction
38
39  constraint cl {
40    rst_n dist {0:/1 , 1:/99};
41
42    if(counter == 23 && old_MOSI_arr[10:8] == 3'b111){
43      SS_n == 1;
44    }
45    if(counter == 13 && old_MOSI_arr[10:8] != 3'b111){
46      SS_n == 1;
47    }
48    if(!(counter == 23 && old_MOSI_arr[10:8] == 3'b111) && !(counter == 13 && old_MOSI_arr[10:8] != 3'b111)){
49      SS_n == 0;
50    }
51
52    if(old_MOSI_arr[10:8] == 3'b111){
53      tx_valid == 1;
54    }
55  }
56
57  function void post_randomize();
58    if(rst_n == 0) begin
59      counter = 0;
60      SS_n = 1;
61    end
62    else begin
63      if(SS_n == 0) begin
64        if(counter == 0) begin
65          MOSI_arr = {next_mosi,MOSI_arr[7:0]};
66          old_MOSI_arr = MOSI_arr;
67        end
68        counter = counter + 1;
69        MOSI = MOSI_arr[10];
70        MOSI_arr = MOSI_arr << 1;
71      end
72      else begin
73        counter = 0;
74
75        if(old_MOSI_arr[10:8] == 3'b000) begin
76          next_mosi = 3'b001;
77        end
78        else if(old_MOSI_arr[10:8] == 3'b001) begin
79          next_mosi = 3'b110;
80        end
81        else if(old_MOSI_arr[10:8] == 3'b110) begin
82          next_mosi = 3'b111;
83        end
84        else if(old_MOSI_arr[10:8] == 3'b111) begin
85          next_mosi = 3'b000;
86        end
87      end
88    end
89  endfunction
90 endclass
91 endpackage
```

## SPI\_slave\_sequencer

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

## SPI\_slave\_RST\_sequence

```
 1 package SPI_slave_RST_sequence_pkg;
 2   import uvm_pkg::*;
 3   import SPI_slave_sequenceitem_pkg::*;
 4   `include "uvm_macros.svh"
 5   class SPI_slave_RST_sequence extends uvm_sequence #(SPI_slave_sequenceitem);
 6     `uvm_object_utils(SPI_slave_RST_sequence)
 7     SPI_slave_sequenceitem item;
 8     function new(string name = "SPI_slave_RST_sequence");
 9       super.new(name);
10     endfunction
11
12     task body();
13       item = SPI_slave_sequenceitem::type_id::create("item");
14       start_item(item);
15       item.rst_n = 0;
16       item.tx_valid = 0;
17       item.tx_data = 0;
18       item.MISO = 0;
19       finish_item(item);
20     endtask
21   endclass
22 endpackage
```

## SPI\_slave\_random\_sequence

```
● ● ●

1 package SPI_slave_random_sequence_pkg;
2   import uvm_pkg::*;
3   import SPI_slave_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class SPI_slave_random_sequence extends uvm_sequence #(SPI_slave_sequenceitem);
6     `uvm_object_utils(SPI_slave_random_sequence)
7     SPI_slave_sequenceitem item;
8     function new(string name = "SPI_slave_random_sequence");
9       super.new(name);
10      endfunction
11
12      task body();
13        item = SPI_slave_sequenceitem::type_id::create("item");
14        for (int i = 0; i < 100000 ; i++) begin
15          start_item(item);
16          assert(item.randomize());
17          finish_item(item);
18        end
19      endtask
20    endclass
21  endpackage
```

## SPI\_slave\_driver

```
1 package SPI_slave_driver_pkg;
2     import uvm_pkg::*;
3     import SPI_RAM_config_pkg::*;
4     import SPI_slave_sequenceitem_pkg::*;
5     `include "uvm_macros.svh"
6 class SPI_slave_driver extends uvm_driver #(SPI_slave_sequenceitem);
7     `uvm_component_utils(SPI_slave_driver)
8     virtual SPI_slave_if SPI_slave_driver_if;
9     virtual SPI_slave_gm_if SPI_slave_driver_inf_gm;
10    SPI_slave_sequenceitem item;
11    function new(string name = "SPI_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        `uvm_info("SPI_slave_driver", "inside the SPI_slave driver", UVM_LOW);
18        item = SPI_slave_sequenceitem::type_id::create("item");
19        forever begin
20            seq_item_port.get_next_item(item);
21            SPI_slave_driver_if.MOSI = item.MOSI;
22            SPI_slave_driver_if.rst_n = item.rst_n;
23            SPI_slave_driver_if.SS_n = item.SS_n;
24            SPI_slave_driver_if.tx_valid = item.tx_valid;
25            SPI_slave_driver_if.tx_data = item.tx_data;
26            SPI_slave_driver_if_gm.tx_data = item.tx_data;
27            SPI_slave_driver_if_gm.MOSI = item.MOSI;
28            SPI_slave_driver_if_gm.rst_n = item.rst_n;
29            SPI_slave_driver_if_gm.SS_n = item.SS_n;
30            SPI_slave_driver_if_gm.tx_valid = item.tx_valid;
31            seq_item_port.item_done();
32            @(negedge SPI_slave_driver_if.clk);
33            item.MISO = SPI_slave_driver_if.MISO;
34            item.rx_valid = SPI_slave_driver_if.rx_valid;
35            item.rx_data = SPI_slave_driver_if.rx_data;
36            `uvm_info("SPI_slave_driver", item.convert2string(), UVM_LOW);
37        end
38    endtask
39
40    endclass
41 endpackage
```

## SPI\_slave\_monitor

```
1 package SPI_slave_monitor_pkg;
2   import uvm_pkg::*;
3   import SPI_RAM_config_pkg::*;
4   import SPI_slave_sequenceitem_pkg::*;
5   `include "uvm_macros.svh"
6   class SPI_slave_monitor extends uvm_monitor;
7     `uvm_component_utils(SPI_slave_monitor)
8     virtual SPI_slave_inf SPI_slave_monitor_inf;
9     virtual SPI_slave_gm_inf SPI_slave_monitor_inf_gm;
10    SPI_slave_sequenceitem item;
11    SPI_slave_sequenceitem item_gm;
12    uvm_analysis_port#(SPI_slave_sequenceitem) mon_ap;
13    uvm_analysis_port#(SPI_slave_sequenceitem) mon_ap_gm;
14
15    function new(string name = "SPI_slave_monitor", 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      mon_ap = new("mon_ap", this);
22      mon_ap_gm = new("mon_ap_gm", this);
23    endfunction
24
25    task run_phase(uvm_phase phase);
26      super.run_phase(phase);
27      `uvm_info("SPI_slave_monitor", "inside the SPI_slave monitor", UVM_LOW);
28      forever begin
29        @(negedge SPI_slave_monitor_inf.clk);
30
31        item_gm = SPI_slave_sequenceitem::type_id::create("item_gm");
32        item_gm.MOSI = SPI_slave_monitor_inf_gm.MOSI;
33        item_gm.rst_n = SPI_slave_monitor_inf_gm.rst_n;
34        item_gm.SS_n = SPI_slave_monitor_inf_gm.SS_n;
35        item_gm.tx_valid = SPI_slave_monitor_inf_gm.tx_valid;
36        item_gm.tx_data = SPI_slave_monitor_inf_gm.tx_data;
37        item_gm.MISO = SPI_slave_monitor_inf_gm.MISO;
38        item_gm.rx_valid = SPI_slave_monitor_inf_gm.rx_valid;
39        item_gm.rx_data = SPI_slave_monitor_inf_gm.rx_data;
40
41
42        item = SPI_slave_sequenceitem::type_id::create("item");
43        item.MOSI = SPI_slave_monitor_inf.MOSI;
44        item.rst_n = SPI_slave_monitor_inf.rst_n;
45        item.SS_n = SPI_slave_monitor_inf.SS_n;
46        item.tx_valid = SPI_slave_monitor_inf.tx_valid;
47        item.tx_data = SPI_slave_monitor_inf.tx_data;
48        item.MISO = SPI_slave_monitor_inf.MISO;
49        item.rx_valid = SPI_slave_monitor_inf.rx_valid;
50        item.rx_data = SPI_slave_monitor_inf.rx_data;
51
52        mon_ap_gm.write(item_gm);
53        mon_ap.write(item);
54        `uvm_info("SPI_slave_monitor", item.convert2string(), UVM_LOW);
55      end
56    endtask
57
58  endclass
59 endpackage
```

## SPI\_slave\_scoreboard

```
1 package SPI_slave_scoreboard_pkg;
2   import uvm_pkg::*;
3   import SPI_slave_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class SPI_slave_scoreboard extends uvm_scoreboard;
6     `uvm_component_utils(SPI_slave_scoreboard)
7     uvm_analysis_export #(SPI_slave_sequenceitem) sb_export;
8     uvm_tlm_analysis_fifo #(SPI_slave_sequenceitem) sb_fifo;
9     uvm_analysis_export #(SPI_slave_sequenceitem) sb_export_gm;
10    uvm_tlm_analysis_fifo #(SPI_slave_sequenceitem) sb_fifo_gm;
11    SPI_slave_sequenceitem item;
12    SPI_slave_sequenceitem item_gm;
13
14
15    int correct_count = 0 , error_count = 0;
16    function new(string name = "SPI_slave_scoreboard", 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      sb_fifo = new("sb_fifo", this);
23      sb_export = new("sb_export", this);
24      sb_fifo_gm = new("sb_fifo_gm", this);
25      sb_export_gm = new("sb_export_gm", this);
26      sb_export.connect(sb_fifo.analysis_export);
27      sb_export_gm.connect(sb_fifo_gm.analysis_export);
28    endfunction
29
30    task run_phase(uvm_phase phase);
31      super.run_phase(phase);
32      forever begin
33        sb_fifo.get(item);
34        sb_fifo_gm.get(item_gm);
35
36        if(item.MISO === item_gm.MISO && item.rx_valid === item_gm.rx_valid && item.rx_data === item_gm.rx_data) begin
37          correct_count = correct_count + 1;
38        end
39        else begin
40          `uvm_error("run_phase", $sformatf("MISO = %d, rx_data = %d, rx_valid = %d", item.MISO, item.rx_data, item.rx_valid));
41          `uvm_error("run_phase", $sformatf("MISO = %d, rx_data = %d, rx_valid = %d", item_gm.MISO, item_gm.rx_data, item_gm.rx_valid));
42          error_count = error_count + 1;
43        end
44      end
45    endtask
46
47
48    function void report_phase(uvm_phase phase);
49      super.report_phase(phase);
50      `uvm_info("report_phase", $sformatf("correct_count = %d", correct_count), UVM_LOW);
51      `uvm_info("report_phase", $sformatf("error_count = %d", error_count), UVM_LOW);
52    endfunction
53  endclass
54 endpackage
```

## SPI\_slave\_coverage

```
1 package SPI_slave_coverage_pkg;
2   import uvm_pkg::*;
3   import SPI_slave_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class SPI_slave_coverage extends uvm_component;
6     `uvm_component_utils(SPI_slave_coverage)
7     SPI_slave_sequenceitem item;
8     uvm_analysis_export #(SPI_slave_sequenceitem) cv_export;
9     uvm_tlm_analysis_fifo #(SPI_slave_sequenceitem) cv_fifo;
10    bit previous_SS_n;
11
12   covergroup cg;
13     rx_data_cp: coverpoint item.rx_data[9:8];
14     SS_n_cp: coverpoint item.SS_n{
15       bins transition13 = (1'b1 => 0[*13] => 1'b1);
16       bins transition23 = (1'b1 => 0[*23] => 1'b1);
17     }
18     MOSI_add_cp: coverpoint item.MOSI iff (item.SS_n == 1){
19       option.auto_bin_max = 0;
20       bins write_address = (0 => 0 => 0);
21       bins write_data = (0 => 0 => 1);
22       bins read_address = (1 => 1 => 0);
23       bins read_data = (1 => 1 => 1);
24     }
25
26     SS_n_MOSI_cc: cross SS_n_cp,MOSI_add_cp {
27       ignore_bins illegal2 = binsof(SS_n_cp.transition23) && binsof(MOSI_add_cp.write_data);
28       ignore_bins illegal4 = binsof(SS_n_cp.transition13) && binsof(MOSI_add_cp.read_data);
29       ignore_bins illegal5 = binsof(SS_n_cp.transition13) && (binsof(MOSI_add_cp.write_data));
30       ignore_bins illegal6 = binsof(SS_n_cp.transition23) && binsof(MOSI_add_cp.read_data);
31     }
32   endgroup
33
34   function new(string name = "SPI_slave_coverage", uvm_component parent = null);
35     super.new(name, parent);
36     cg = new();
37   endfunction
38
39   function void build_phase(uvm_phase phase);
40     super.build_phase(phase);
41     cv_export = new("cv_export", this);
42     cv_fifo = new("cv_fifo", this);
43   endfunction
44
45   function void connect_phase(uvm_phase phase);
46     super.connect_phase(phase);
47     cv_export.connect(cv_fifo.analysis_export);
48   endfunction
49
50   task run_phase(uvm_phase phase);
51     super.run_phase(phase);
52     forever begin
53       cv_fifo.get(item);
54       previous_SS_n = item.SS_n;
55       cg.sample();
56     end
57   endtask
58
59   endclass
60 endpackage
```

## SPI\_slave\_agent

```
 1 package SPI_slave_agent_pkg;
 2   import uvm_pkg::*;
 3   `include "uvm_macros.svh"
 4   import SPI_RAM_config_pkg::*;
 5   import SPI_slave_sequencer_pkg::*;
 6   import SPI_slave_driver_pkg::*;
 7   import SPI_slave_monitor_pkg::*;
 8   import SPI_slave_sequenceitem_pkg::*;
 9   class SPI_slave_agent extends uvm_agent;
10     `uvm_component_utils(SPI_slave_agent)
11     SPI_slave_driver driver;
12     SPI_slave_monitor monitor;
13     SPI_slave_sequencer sequencer;
14     SPI_RAM_config cfg;
15     uvm_analysis_port #(SPI_slave_sequenceitem) agt_ap;
16     uvm_analysis_port #(SPI_slave_sequenceitem) agt_ap_gm;
17
18     function new(string name = "SPI_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#(SPI_RAM_config)::get(this, "", "SPI_slave_config_test", cfg)) begin
25         `uvm_fatal("SPI_slave_agent", "virtual interface must be set for SPI_slave_agent")
26       end
27       sequencer = SPI_slave_sequencer::type_id::create("sequencer", this);
28       driver = SPI_slave_driver::type_id::create("driver", this);
29       monitor = SPI_slave_monitor::type_id::create("monitor", this);
30       agt_ap = new("agt_ap", this);
31       agt_ap_gm = new("agt_ap_gm", this);
32     endfunction
33
34     function void connect_phase(uvm_phase phase);
35       super.connect_phase(phase);
36       driver.SPI_slave_driver_inf = cfg.SPI_vif;
37       driver.SPI_slave_driver_inf_gm = cfg.SPI_vif_gm;
38       monitor.SPI_slave_monitor_inf = cfg.SPI_vif;
39       monitor.SPI_slave_monitor_inf_gm = cfg.SPI_vif_gm;
40       driver.seq_item_port.connect(sequencer.seq_item_export);
41       monitor.mon_ap.connect(agt_ap);
42       monitor.mon_ap_gm.connect(agt_ap_gm);
43     endfunction
44
45   endclass
46
47 endpackage
```

## SPI\_slave\_env

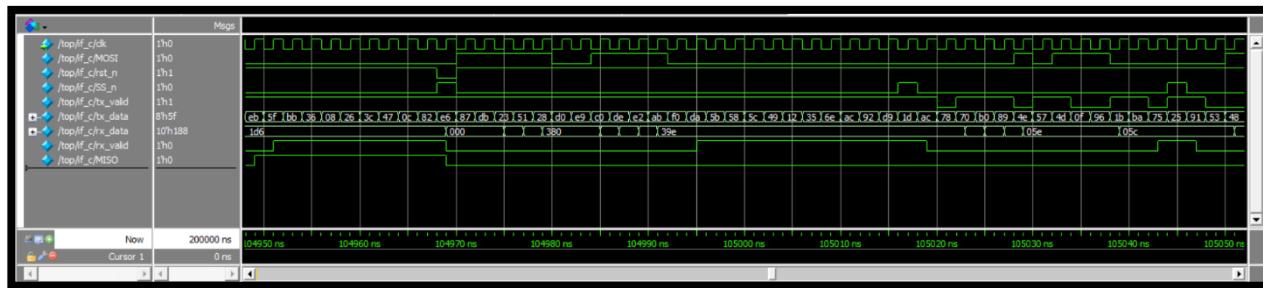
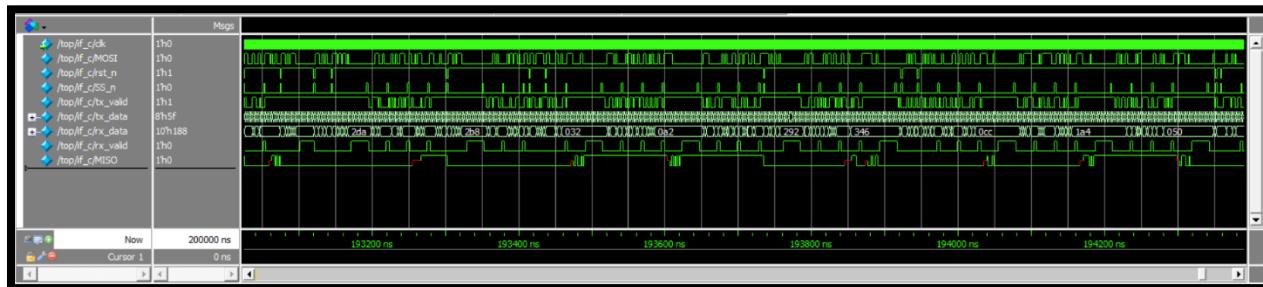
```
● ● ●

1 package SPI_slave_env_pkg;
2     import uvm_pkg::*;
3     import SPI_slave_agent_pkg::*;
4     import SPI_slave_coverage_pkg::*;
5     import SPI_slave_scoreboard_pkg::*;
6     `include "uvm_macros.svh"
7     class SPI_slave_env extends uvm_env;
8         `uvm_component_utils(SPI_slave_env)
9         SPI_slave_scoreboard scoreboard;
10        SPI_slave_coverage coverage;
11        SPI_slave_agent agent;
12        function new(string name = "SPI_slave_env", 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            agent = SPI_slave_agent::type_id::create("agent", this);
19            scoreboard = SPI_slave_scoreboard::type_id::create("scoreboard", this);
20            coverage = SPI_slave_coverage::type_id::create("coverage", this);
21        endfunction
22
23        function void connect_phase(uvm_phase phase);
24            super.connect_phase(phase);
25            agent.agt_ap.connect(coverage.cv_export);
26            agent.agt_ap.connect(scoreboard.sb_export);
27            agent.agt_ap_gm.connect(scoreboard.sb_export_gm);
28        endfunction
29    endclass
30 endpackage
```

## SPI\_slave\_test

```
1 package SPI_slave_test_pkg;
2   import uvm_pkg::*;
3   import SPI_slave_env_pkg::*;
4   import SPI_RAM_config_pkg::*;
5   import SPI_slave_rst_sequence_pkg::*;
6   import SPI_slave_random_sequence_pkg::*;
7   `include "uvm_macros.svh"
8   class SPI_slave_test extends uvm_test;
9     `uvm_component_utils(SPI_slave_test)
10
11   SPI_slave_env env;
12   SPI_RAM_config cfg;
13   SPI_slave_rst_sequence rst_sequence;
14   SPI_slave_random_sequence random_sequence;
15   function new(string name = "SPI_slave_test", 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     env = SPI_slave_env::type_id::create("env", this);
22     cfg = SPI_RAM_config::type_id::create("cfg");
23     rst_sequence = SPI_slave_rst_sequence::type_id::create("rst_sequence");
24     random_sequence = SPI_slave_random_sequence::type_id::create("random_sequence");
25     if (!uvm_config_db#(virtual SPI_slave_if)::get(this, "", "vif", cfg.SPI_vif)) begin
26       `uvm_fatal("SPI_slave_test", "virtual interface must be set for SPI_slave_test")
27     end
28     if (!uvm_config_db#(virtual SPI_slave_gm_if)::get(this, "", "vif_gm", cfg.SPI_vif_gm)) begin
29       `uvm_fatal("SPI_slave_test", "virtual interface must be set for SPI_slave_test")
30     end
31     uvm_config_db#(SPI_RAM_config)::set(this, "*", "SPI_slave_config_test", cfg);
32   endfunction
33
34   task run_phase(uvm_phase phase);
35     super.run_phase(phase);
36     phase.raise_objection(this);
37     `uvm_info("SPI_slave_test", "running reset test", UVM_LOW);
38     rst_sequence.start(env.agent.sequencer);
39     `uvm_info("Finish_test", "reset test finished", UVM_LOW);
40     #2;
41     `uvm_info("SPI_slave_test", "running random test", UVM_LOW);
42     random_sequence.start(env.agent.sequencer);
43     `uvm_info("Finish_test", "random test finished", UVM_LOW);
44     phase.drop_objection(this);
45   endtask
46 endclass
47 endpackage
```

# QuestaSim Snippets



```
# UVM_INFO SPI_slave_test.sv(43) @ 200000: uvm_test_top [Finish_test] random test finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 200000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO SPI_slave_scoreboard.sv(50) @ 200000: uvm_test_top.env.scoreboard [report_phase] correct_count =      100000
# UVM_INFO SPI_slave_scoreboard.sv(51) @ 200000: uvm_test_top.env.scoreboard [report_phase] error_count =      0
--- UVM Report Summary ---
** Report counts by severity
UVM_INFO :200012
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
#[Finish_test] 2
#[Questasim] 2
#[RNTST] 1
#[SPI_slave_driver] 100001
#[SPI_slave_monitor] 100001
#[SPI_slave_test] 2
#[TEST_DONE] 1
#[report_phase] 2
** Note: $finish : C:/questasim64_2021.1/win64//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 200 us Iteration: 65 Instance: /top
# 1
# 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
$SIM 2>]
```

# Functional Coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/SPI_slave_coverage_pkg/SPI_slave_coverage/cg		100.00%	100	100.00%		✓			auto(0)
CVP_cg::rx_data_cp	Coverpoint	100.00%	100	100.00%		✓			
CVP_cg::SS_n_cp	Coverpoint	100.00%	100	100.00%		✓			
CVP_cg::MOSI_add_cp	Coverpoint	100.00%	100	100.00%		✓			
CROSS_cg::SS_n_MOSI_c...	Cross	100.00%	100	100.00%		✓			
INST /SPI_slave_coverage...	Instance	100.00%	100	100.00%		✓			0
CVP rx_data_cp	Coverpoint	100.00%	100	100.00%		✓			
bin auto[0]	Bin	56804	1	100.00%		✓			
bin auto[1]	Bin	15212	1	100.00%		✓			
bin auto[2]	Bin	15201	1	100.00%		✓			
bin auto[3]	Bin	12783	1	100.00%		✓			
CVP SS_n_cp	Coverpoint	100.00%	100	100.00%		✓			
bin transition13	Bin	4158	1	100.00%		✓			
bin transition23	Bin	1382	1	100.00%		✓			
CVP MOSI_add_cp	Coverpoint	100.00%	100	100.00%		✓			
bin write_address	Bin	38557	1	100.00%		✓			
bin write_data	Bin	10595	1	100.00%		✓			
bin read_address	Bin	9275	1	100.00%		✓			
bin read_data	Bin	8248	1	100.00%		✓			
CROSS_SS_n_MOSI_c...	Cross	100.00%	100	100.00%		✓			
bin <transition23,w...	Bin	1382	1	100.00%		✓			
bin <transition13,w...	Bin	4158	1	100.00%		✓			
ignore_bin illegal8	Bin	0	-	-					
ignore_bin illegal7	Bin	0	-	-					
ignore_bin illegal6	Bin	0	-	-					
ignore_bin illegal5	Bin	0	-	-					
ignore_bin illegal4	Bin	0	-	-					
ignore_bin illegal2	Bin	0	-	-					

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

Covergroup Coverage:
Covergroups          1      na      na  100.00%
  Coverpoints/Crosses 4      na      na  na
    Covergroup Bins   12     12      0  100.00%
-----
```

Covergroup	Metric	Goal	Bins	Status
TYPE /SPI_slave_coverage_pkg/SPI_slave_coverage/cg	100.00%	100	-	Covered
covered/total bins:	12	12	-	
missing/total bins:	0	12	-	
% Hit:	100.00%	100	-	
Coverpoint rx_data_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Coverpoint SS_n_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Coverpoint MOSI_add_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
Cross SS_n_MOSI_cc	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Covergroup instance \SPI_slave_coverage_pkg::SPT_slave_coverage::cg	100.00%	100	-	Covered
covered/total bins:	12	12	-	
missing/total bins:	0	12	-	
% Hit:	100.00%	100	-	
Coverpoint rx_data_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
bin auto[0]	56804	1	-	Covered
bin auto[1]	15212	1	-	Covered
bin auto[2]	15201	1	-	Covered
bin auto[3]	12783	1	-	Covered
Coverpoint SS_n_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin transition13	4158	1	-	Covered
bin transition23	1382	1	-	Covered

Coverpoint MOST_add_cp	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	
bin write_address	38557	1	-	Covered
bin write_data	10595	1	-	Covered
bin read_address	9275	1	-	Covered
bin read_data	8248	1	-	Covered
Cross SS_n_MOST_cc	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <transition23,write_address>	1382	1	-	Covered
bin <transition13,write_address>	4158	1	-	Covered
Illegal and Ignore Bins:				
ignore_bin illegal18	0		-	ZERO
ignore_bin illegal17	0		-	ZERO
ignore_bin illegal16	0		-	ZERO
ignore_bin illegal15	0		-	ZERO
ignore_bin illegal14	0		-	ZERO
ignore_bin illegal12	0		-	ZERO

## Assertion Summary

Feature	Assertion
<b>Whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all low.</b>	<pre>@(posedge if_c.clk)     (~if_c.rst_n  =&gt; (if_c.MISO == 'b0 &amp;&amp; if_c.rx_valid == 'b0 &amp;&amp; if_c.rx_data == 'b0))</pre>
<b>After write_add_seq (000), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (if_c.SS_n ##1 !if_c.SS_n ##1 (!if_c.MOSI ##1 !if_c.MOSI ##1 !if_c.MOSI))  =&gt; ##10 (\$rose(if_c.rx_valid)) ##[1:\$] if_c.SS_n;</pre>
<b>After write_data_seq (001), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (if_c.SS_n ##1 !if_c.SS_n ##1 (!if_c.MOSI ##1 !if_c.MOSI ##1 if_c.MOSI))  =&gt; ##10 (\$rose(if_c.rx_valid)) ##[1:\$] if_c.SS_n;</pre>
<b>After read_add_seq(110), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (if_c.SS_n ##1 !if_c.SS_n ##1 if_c.MOSI ##1 if_c.MOSI ##1 !if_c.MOSI)  =&gt; ##10 (\$rose(if_c.rx_valid)) ##[1:\$] if_c.SS_n;</pre>
<b>After read_data_seq(111), the rx_valid signal must assert exactly after 10 cycles and the SS_n should eventually after the 10 cycles to close communication.</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (if_c.SS_n ##1 !if_c.SS_n ##1 if_c.MOSI ##1 if_c.MOSI ##1 if_c.MOSI)  =&gt; ##10 (\$rose(if_c.rx_valid)) ##[1:\$] if_c.SS_n;</pre>
<b>FSM transition: IDLE → CHK_CMD</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (cs == IDLE)  =&gt; (cs == IDLE    cs == CHK_CMD)</pre>
<b>FSM transition: CHK_CMD → WRITE or READ_ADD or READ_DATA</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (cs == CHK_CMD)  =&gt; (cs == IDLE    cs == WRITE    cs == READ_ADD    cs == READ_DATA)</pre>
<b>FSM transition: WRITE → IDLE</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (cs == WRITE)  =&gt; (cs == WRITE    cs == IDLE)</pre>
<b>FSM transition: READ_ADD → IDLE</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (cs == READ_ADD)  =&gt; (cs == READ_ADD    cs == IDLE)</pre>
<b>FSM transition: READ_DATA → IDLE</b>	<pre>@(posedge if_c.clk) disable iff(~if_c.rst_n)     (cs == READ_DATA)  =&gt; (cs == READ_DATA    cs == IDLE)</pre>

# Assertion Coverage

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative
/top/SPI_slave_inst/cover_p_state_operation_5	SVA	✓	Off	26462	1	Unl...	1	100%		✓	0	0	0 ns	
/top/SPI_slave_inst/cover_p_state_operation_4	SVA	✓	Off	24198	1	Unl...	1	100%		✓	0	0	0 ns	
/top/SPI_slave_inst/cover_p_state_operation_3	SVA	✓	Off	34541	1	Unl...	1	100%		✓	0	0	0 ns	
/top/SPI_slave_inst/cover_p_state_operation_2	SVA	✓	Off	6322	1	Unl...	1	100%		✓	0	0	0 ns	
/top/SPI_slave_inst/cover_p_state_operation_1	SVA	✓	Off	6388	1	Unl...	1	100%		✓	0	0	0 ns	
/top/sva/cover_p_rx_valid_SS_n_operation_read_data	SVA	✓	Off	529	1	Unl...	1	100%		✓	0	0	0 ns	
/top/sva/cover_p_rx_valid_SS_n_operation_read_add	SVA	✓	Off	535	1	Unl...	1	100%		✓	0	0	0 ns	
/top/sva/cover_p_rx_valid_SS_n_operation_write_data	SVA	✓	Off	521	1	Unl...	1	100%		✓	0	0	0 ns	
/top/sva/cover_p_rx_valid_SS_n_operation_write_add	SVA	✓	Off	508	1	Unl...	1	100%		✓	0	0	0 ns	
/top/sva/cover_p_reset	SVA	✓	Off	1051	1	Unl...	1	100%		✓	0	0	0 ns	

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Mem
/SPI_slave_random_sequence_pkg::SPI_slave_random_sequence::body::#anonblk#1301127#14#...	Immediate	SVA	on	0	1	-	-	0B	0B
/top/SPI_slave_inst/assert_p_state_operation_1	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/SPI_slave_inst/assert_p_state_operation_2	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/SPI_slave_inst/assert_p_state_operation_3	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/SPI_slave_inst/assert_p_state_operation_4	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/SPI_slave_inst/assert_p_state_operation_5	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/sva/assert_p_reset	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/sva/assert_p_rx_valid_SS_n_operation_read_data	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/sva/assert_p_rx_valid_SS_n_operation_write_data	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/sva/assert_p_rx_valid_SS_n_operation_write_add	Concurrent	SVA	on	0	1	-	0B	0B	0B
/top/sva/uvm_reg_map::do_read#(ubiK#(215181159#1771)immed_1775	Immediate	SVA	on	0	0	-	-	-	-
/top/sva/uvm_reg_map::do_write#(ubiK#(215181159#1731)immed_1735	Immediate	SVA	on	0	0	-	-	-	-

```
=====
== Instance: /top/sva
== Design Unit: work.SPI_slave_sva
=====

Assertion Coverage:
 Assertions      5      5      0  100.00%
 -----
Name          File(Line)          Failure  Pass |
                   Count        Count
-----
/tоп/sва/assert_p_rx_valid_SS_n_operation_read_data
          SPI_slave_sva.sv(42)      0      1
/tоп/sва/assert_p_rx_valid_SS_n_operation_read_add
          SPI_slave_sva.sv(35)      0      1
/tоп/sва/assert_p_rx_valid_SS_n_operation_write_data
          SPI_slave_sva.sv(28)      0      1
/tоп/sва/assert_p_rx_valid_SS_n_operation_write_add
          SPI_slave_sva.sv(21)      0      1
/tоп/sва/assert_p_reset
          SPI_slave_sva.sv(14)      0      1

Directive Coverage:
 Directives      5      5      0  100.00%
 -----
DIRECTIVE COVERAGE:
 -----
Name          Design Design Lang File(Line)   Hits Status
Unit       UnitType
-----
/tоп/sва/cover_p_rx_valid_SS_n_operation_read_data
          SPI_slave_sva Verilog SVA SPI_slave_sva.sv(43) 529 Covered
/tоп/sва/cover_p_rx_valid_SS_n_operation_read_add
          SPI_slave_sva Verilog SVA SPI_slave_sva.sv(36) 535 Covered
/tоп/sва/cover_p_rx_valid_SS_n_operation_write_data
          SPI_slave_sva Verilog SVA SPI_slave_sva.sv(29) 521 Covered
/tоп/sва/cover_p_rx_valid_SS_n_operation_write_add
          SPI_slave_sva Verilog SVA SPI_slave_sva.sv(22) 508 Covered
/tоп/sва/cover_p_reset
          SPI_slave_sva Verilog SVA SPI_slave_sva.sv(15) 1051 Covered
```

## Code Coverage

```
=====
== Instance: /top/ifc
== Design Unit: work.SPI_slave_ifc
=====

Toggle Coverage:
Enabled Coverage          Bins    Hits    Misses  Coverage
-----      -----      -----      -----
Toggles                  50      50       0   100.00%

=====Toggle Details=====
Toggle Coverage for instance /top/ifc --

          Node      1H->0L      0L->1H  "Coverage"
-----
          MISO        1          1   100.00
          MOSI        1          1   100.00
          SS_n        1          1   100.00
          clk         1          1   100.00
          rst_n       1          1   100.00
          rx_data[9-0] 1          1   100.00
          rx_valid    1          1   100.00
          tx_data[7-0] 1          1   100.00
          tx_valid    1          1   100.00

Total Node Count = 25
Toggled Node Count = 25
Untoggled Node Count = 0

Toggle Coverage = 100.00% (50 of 50 bins)
```

```

Branch Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      -----
  Branches          39      39      0   100.00%
=====
Branch Details=====
Branch Coverage for instance /top/SPI_slave_inst

  Line     Item      Count      Source
  -----  -----
File SPI_slave.sv
-----IF Branch-----
  15                      25698  Count coming in to IF
  15      1                  1048   if (~if_c.rst_n) begin
  18      1                  24650  else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----CASE Branch-----
  24                      65151  Count coming in to CASE
  25      1                  12941  IDLE : begin
  31      1                  6532   CHK_CMD : begin
  45      1                  20975  WRITE : begin
  51      1                  13882  READ_ADD : begin
  57      1                  10820  READ_DATA : begin
  1                   All False Count
Branch totals: 6 hits of 6 branches = 100.00%
-----IF Branch-----
  26                      12941  Count coming in to IF
  26      1                  6481   if (if_c.SS_n)
  28      1                  6460   else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
  32                      6532   Count coming in to IF
  32      1                  72     if (if_c.SS_n)
  34      1                  6460   else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
  35                      6460   Count coming in to IF
  35      1                  3151   if (~if_c.MOSI)
  37      1                  3309   else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
  38                      3309   Count coming in to IF
  38      1                  1788   if (~received_address) //fix
  40      1                  1521   else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
  46                      20975  Count coming in to IF
  46      1                  3112   if (if_c.SS_n)
  48      1                  17863  else
Branch totals: 2 hits of 2 branches = 100.00%

```

```

-----IF Branch-----
 58          10820   Count coming in to IF
 58          1503    if (~if_c.SS_n)
 60          9317    else
Branch totals: 2 hits of 2 branches = 100.00%

-----IF Branch-----
 67          100000  Count coming in to IF
 67          1051    if (~if_c.RST_n) begin
 74          98949   else begin
Branch totals: 2 hits of 2 branches = 100.00%

-----CASE Branch-----
 75          98949   Count coming in to CASE
 76          6460    IDLE : begin
 80          6388    CHK_CMD : begin
 83          34919   WRITE : begin
 93          24445   READ_ADD : begin
103          26737   default : begin
Branch totals: 5 hits of 5 branches = 100.00%

-----IF Branch-----
 84          34919   Count coming in to IF
 84          32149   if (counter > 0) begin
 88          2770    else begin
Branch totals: 2 hits of 2 branches = 100.00%

-----IF Branch-----
 94          24445   Count coming in to IF
 94          16835   if (counter > 0) begin
 98          7610    else begin
Branch totals: 2 hits of 2 branches = 100.00%

-----IF Branch-----
 104         26737   Count coming in to IF
 104         25019   if (~if_c.TX_VALID) begin
 115         1718    else begin
Branch totals: 2 hits of 2 branches = 100.00%

-----IF Branch-----
 106         25019   Count coming in to IF
 106         12721   if (counter > 0) begin
 110         12298   else begin
Branch totals: 2 hits of 2 branches = 100.00%

-----IF Branch-----
 116         1718    Count coming in to IF
 116         1525    if (counter > 0) begin
 120         193     else begin
Branch totals: 2 hits of 2 branches = 100.00%

```

=====FSM Details=====

FSM Coverage for instance /top/SPI\_slave\_inst --

FSM\_ID: cs

Current State Object : cs

-----  
State Value MapInfo :

Line	State Name	Value
25	IDLE	0
31	CHK_CMD	1
57	READ_DATA	4
51	READ_ADD	3
45	WRITE	2

Covered States :

State	Hit_count
IDLE	6528
CHK_CMD	6460
READ_DATA	2982
READ_ADD	3531
WRITE	6197

Covered Transitions :

Line	Trans_ID	Hit_count	Transition
29	0	6460	IDLE -> CHK_CMD
41	1	1503	CHK_CMD -> READ_DATA
39	2	1772	CHK_CMD -> READ_ADD
36	3	3113	CHK_CMD -> WRITE
33	4	72	CHK_CMD -> IDLE
59	5	1503	READ_DATA -> IDLE
53	6	1772	READ_ADD -> IDLE
47	7	3112	WRITE -> IDLE

Summary

Bins Hits Misses Coverage

-----

FSM States 5 5 0 100.00%

FSM Transitions 8 8 0 100.00%

Statement Coverage:

Enabled Coverage

Bins Hits Misses Coverage

-----

Statements 42 42 0 100.00%

=====Statement Details=====

Statement Coverage for instance /top/SPI\_slave\_inst --

Line	Item	Count	Source
---	---	----	-----
File SPI_slave.sv			
1			module SPI_slave(SPI_slave_if#(ifc) ifc);
2			localparam IDLE       = 3'b000;
3			localparam CHK_CMD   = 3'b001;
4			localparam WRITE     = 3'b010;
5			localparam READ_ADD  = 3'b011;
6			localparam READ_DATA = 3'b100;
7			
8			reg [3:0] counter;
9			reg          received_address;
10			
11			reg [2:0] cs, ns;
12			
13			always @ (posedge ifc.clk) begin
14	1	25698	if (~ifc.rst_n) begin
15			cs <= IDLE;
16	1	1048	end
17			else begin
18			cs <= ns;
19	1	24650	end
20			end
21			
22			always @ (*) begin
23	1	65151	case (cs)
24			IDLE : begin
25			if (ifc.SS_n)
26	1	6481	ns = IDLE;
27			else
28	1	6460	ns = CHK_CMD;
29			end
30			CHK_CMD : begin
31			if (ifc.SS_n)
32			ns = IDLE;
33	1	72	else begin
34			if (~ifc.MOST)
35	1	3151	ns = WRITE;
36			else begin
37			if (~received_address) //fix
38			ns = READ_ADD;
39	1	1788	else
40			ns = READ_DATA;
41	1	1521	end
42			end
43			end
44			WRITE : begin
45			if (ifc.SS_n)
46	1	3112	ns = IDLE;
47			else
48	1	17863	ns = WRITE;
49			end
50			READ_ADD : begin
51			if (ifc.SS_n)
52			ns = IDLE;
53	1	1772	

```

54      1           12110
55      1           12110
56      1           12110
57      1           12110
58      1           1503
59      1           1503
60      1           9317
61      1           9317
62      1           100000
63      1           100000
64      1           100000
65      1           100000
66      1           100000
67      1           100000
68      1           1051
69      1           1051
70      1           1051
71      1           1051
72      1           1051
73      1           1051
74      1           1051
75      1           1051
76      1           1051
77      1           6460
78      1           6460
79      1           6460
80      1           6388
81      1           6388
82      1           6388
83      1           6388
84      1           32149
85      1           32149
86      1           32149
87      1           32149
88      1           2770
89      1           2770
90      1           2770
91      1           2770
92      1           2770
93      1           2770
94      1           2770
95      1           16835
96      1           16835
97      1           16835
98      1           16835
99      1           7610
100     1           7610
101     1           7610
102     1           7610
103     1           7610
104     1           25019
105     1           25019
106     1           25019
107     1           12721
108     1           12721
109     1           12721
110     1           12721
111     1           12298
112     1           12298
113     1           12298
114     1           12298
115     1           12298
116     1           12298
117     1           1525
118     1           1525
119     1           1525
120     1           193
121     1           193
122     1           193

```

```

112     1           12298
113     1           12298
114     1           12298
115     1           12298
116     1           12298
117     1           1525
118     1           1525
119     1           1525
120     1           1525
121     1           193
122     1           193

```

Toggle Coverage:  
 Enabled Coverage      Bins    Hits    Misses    Coverage  
 -----                -----    -----    -----    -----  
 Toggles                22      22      0        100.00%

=====Toggle Details=====

Toggle Coverage for instance /top/SPI\_slave\_inst --

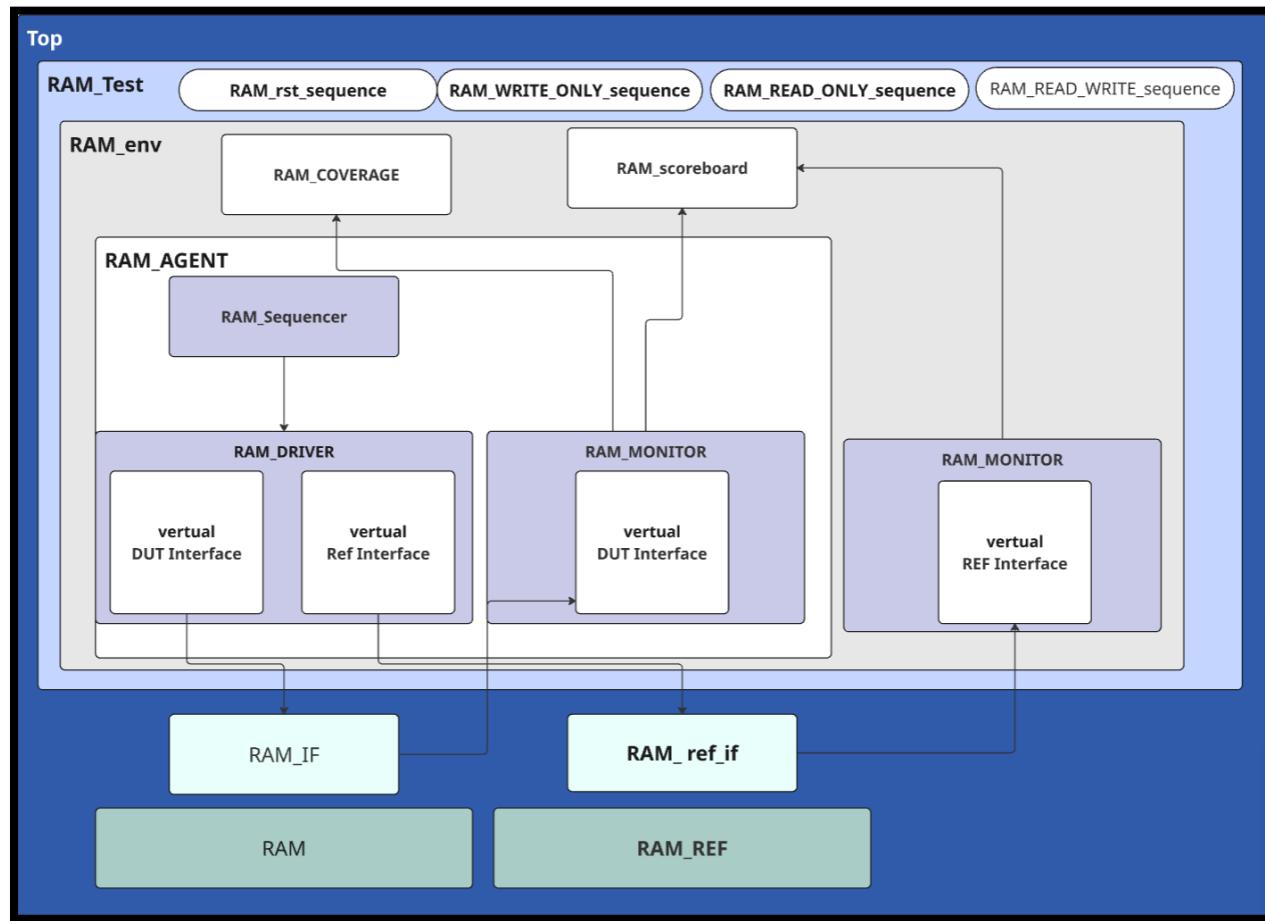
Node	1H->0L	0L->1H	"Coverage"
counter[3:0]	1	1	100.00
cs[2:0]	1	1	100.00
ns[2:0]	1	1	100.00
received_address	1	1	100.00

Total Node Count    =    11  
 Toggled Node Count =    11  
 Untoggled Node Count =    0

Toggle Coverage    =    100.00% (22 of 22 bins)

# RAM Verification

## UVM Diagram



The top module instantiates two interfaces, one for the RAM design and one for its golden reference model. Additionally, it instantiates both the RAM DUT and its reference model, each connected to its corresponding interface. Finally, both interfaces are stored in the configuration database to be retrieved later during the verification process, and the simulation is started by the “run\_test” function for starting the RAM test.

The RAM test begins by retrieving these interfaces from the configuration database and embedding them into configuration objects that hold their respective virtual interfaces. The test then creates the RAM environment, which will be explained below, and triggers the four sequences, reset, write-only, read-only, and read & write, as shown in the diagram test

sector. Each sequence generates constrained sequence items that are sent from the sequencer to the driver.

Inside the RAM environment, three main components are created: the scoreboard, the coverage collector, and the UVM agent.

The RAM scoreboard is responsible for comparing the DUT outputs with the reference model outputs by using two analysis FIFOs and analysis ports that receive the transactions from the monitors. Any mismatches between the DUT and the reference are reported for debugging.

The RAM coverage collector operates similarly by capturing the same transactions through its analysis ports to verify that all required functional coverage points are met during the test execution.

The RAM agent receives the configuration object to determine the environment's activity mode. Since this environment is active, the agent instantiates its driver, sequencer, and monitor components. The driver sends stimulus transactions from the sequencer to the DUT through its virtual DUT interface, while the monitors observe both the DUT and reference interfaces to capture transactions and forward them to the scoreboard and coverage collector.

This process continues throughout the duration of all sequence operations. Once all test iterations and stimulus generations are completed successfully, the final objection is dropped from the test, signaling the end of the UVM test.

## RTL Code

```
● ○ ●
1 module RAM (RAM_if.DUT DUT_if);
2
3 logic[7:0] MEM [255:0];
4 logic [7:0] Rd_Addr, Wr_Addr;
5
6 always @(posedge DUT_if.clk) begin
7     if (~DUT_if.rst_n) begin
8         DUT_if.dout <= 0;
9         DUT_if.tx_valid <= 0;
10        Rd_Addr <= 0;
11        Wr_Addr <= 0;
12    end
13    else begin //fix
14        if (DUT_if.rx_valid) begin
15            case (DUT_if.din[9:8])
16                2'b00 : Wr_Addr <= DUT_if.din[7:0];
17                2'b01 : MEM[Wr_Addr] <= DUT_if.din[7:0];
18                2'b10 : Rd_Addr <= DUT_if.din[7:0];
19                2'b11 : DUT_if.dout <= MEM[Rd_Addr];//fix
20                default : DUT_if.dout <= 0;
21            endcase
22        end
23        DUT_if.tx_valid <= (DUT_if.din[9] && DUT_if.din[8] && DUT_if.rx_valid)? 1'b1 : 1'b0;
24    end
25 end
26
27 endmodule
```

## Bugs are fixed

### 1\_ read data from write data address

### 2\_there is no begin \_end in else so the tx\_valid condition can be high when the reset is asserted

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

# Code Snippets

## Top module

```
● ○ ●
1 import uvm_pkg::*;
2 import RAM_test::*;
3 `include "uvm_macros.svh"
4 module RAM_top();
5   bit clk;
6   initial begin
7     clk = 0;
8     forever #1 clk = ~clk;
9   end
10  RAM_if RAM_if_inst(clk);
11  RAM_if_ref RAM_if_ref_inst(clk);
12  RAM DUT(RAM_if_inst);
13  RAM_ref REF(RAM_if_ref_inst);
14  assign RAM_if_ref_inst.din = RAM_if_inst.din;
15  assign RAM_if_ref_inst.rx_valid = RAM_if_inst.rx_valid;
16  assign RAM_if_ref_inst.rst_n = RAM_if_inst.rst_n;
17  bind RAM RAM_sva RAM_sva_inst(RAM_if_inst.DUT);
18
19  initial begin
20    uvm_config_db#(virtual RAM_if)::set(null, "uvm_test_top", "vif_DUT", RAM_if_inst);
21    uvm_config_db#(virtual RAM_if_ref)::set(null, "uvm_test_top", "vif_REF", RAM_if_ref_inst);
22    run_test("RAM_test");
23  end
24
25 endmodule
```

## Module Interface

```
● ○ ●
1 interface RAM_if(clk);
2   input clk;
3   logic [9:0] din;
4   logic rst_n, rx_valid;
5   logic [7:0] dout;
6   logic tx_valid;
7   modport DUT (input din, clk, rst_n, rx_valid, output dout, tx_valid);
8 endinterface
```

## Golden Model Interface

```
● ● ●  
1 interface RAM_if_ref(clk);  
2   input clk ;  
3   logic [9:0] din;  
4   logic rst_n, rx_valid;  
5   logic [7:0] dout;  
6   logic tx_valid;  
7   modport REF (input din, clk, rst_n, rx_valid, output dout, tx_valid);  
8 endinterface
```

## Golden Model Module

```
 1 module RAM_ref (RAM_if_ref.REF REF_if);
 2
 3 logic [7:0] MEM [255:0];
 4 logic [7:0] Rd_Addr, Wr_Addr;
 5
 6 always @ (posedge REF_if.clk) begin
 7     if (~REF_if.rst_n) begin
 8         REF_if.dout <= 0;
 9         REF_if.tx_valid <= 0;
10         Rd_Addr <= 0;
11         Wr_Addr <= 0;
12     end
13     else begin
14         if (REF_if.rx_valid) begin
15             case (REF_if.din[9:8])
16                 2'b00 : Wr_Addr <= REF_if.din[7:0];
17                 2'b01 : MEM[Wr_Addr] <= REF_if.din[7:0];
18                 2'b10 : Rd_Addr <= REF_if.din[7:0];
19                 2'b11 : REF_if.dout <= MEM[Rd_Addr];
20                 default : REF_if.dout <= 0;
21             endcase
22         end
23         REF_if.tx_valid <= (REF_if.din[9] && REF_if.din[8] && REF_if.rx_valid)? 1'b1 : 1'b0;
24     end
25 end
26
27 endmodule
```

## RAM\_sva

```
1 module RAM_sva (RAM_if.dut DUT_if);
2
3 //==reset assertion
4 property rst_check;
5 @posedge DUT_if.clk !DUT_if.rst_n |=> ((DUT_if.dout)==8'b00000000 &&DUT_if.tx_valid==0);
6 endproperty
7 assert property (rst_check) else $error("Reset assertion failed");
8 cover property (rst_check);
9
10 // ===tx_valid_low assertion===
11 property tx_valid_low_check;
12 @posedge DUT_if.clk disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]!=2'b11) |=> (DUT_if.tx_valid)==0;
13 endproperty
14 assert property (tx_valid_low_check) else $error("tx_valid_low assertion failed");
15 cover property (tx_valid_low_check);
16
17 // ===tx_valid_high assertion===
18 property tx_valid_high_check;
19 @posedge DUT_if.clk disable iff (!DUT_if.rst_n) ($past(DUT_if.din[9:8]==2'b10&& DUT_if.din[9:8]==2'b11) |=> ($rose(DUT_if.tx_valid))|=>##[1:$]($fell(DUT_if.tx_valid)));
20 endproperty
21 assert property (tx_valid_high_check) else $error("tx_valid_high assertion failed");
22 cover property (tx_valid_high_check);
23
24 // ===write address then write data assertion===
25 property write_addr_data_check;
26 @posedge DUT_if.clk disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]==2'b00) |=> ##[1:$] (DUT_if.din[9:8]==2'b01);
27 endproperty
28 assert property (write_addr_data_check) else $error("write_addr_data assertion failed");
29 cover property (write_addr_data_check);
30
31 // ===read address then read data assertion===
32 property read_addr_data_check;
33 @posedge DUT_if.clk disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]==2'b10) |=> ##[1:$] (DUT_if.din[9:8]==2'b11);
34 endproperty
35 assert property (read_addr_data_check) else $error("read_addr_data assertion failed");
36 cover property (read_addr_data_check);
37 endmodule
```

## RAM\_config

```
1 package RAM_config;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 class RAM_config extends uvm_object;
5 `uvm_object_utils(RAM_config)
6 virtual RAM_if vif;
7 virtual RAM_if_ref vif_ref;
8 function new(string name="RAM_config");
9     super.new(name);
10 endfunction
11 endclass
12 endpackage
```

## RAM\_sequenceitem

```
● ● ●
1 package RAM_seq_item;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 class RAM_seq_item extends uvm_sequence_item;
5 `uvm_object_utils(RAM_seq_item)
6 rand logic [9:0] din;
7 rand logic rx_valid;
8 rand logic rst_n;
9 rand logic [7:0] dout;
10 rand logic tx_valid;
11 logic [1:0]old_op=2'b00;
12
13 function new(string name="RAM_seq_item");
14     super.new(name);
15 endfunction
16
17 function string convert2string();
18     return $sformatf("din=%0h, rx_valid=%0b, dout=%0h, tx_valid=%0b", din, rx_valid, dout, tx_valid);
19 endfunction
20
21 // ===constraints===
22 constraint reset_cn {
23     rst_n dist {0:=1, 1:=9};
24 }
25
26 constraint rx_valid_cn {
27     rx_valid dist {0:=1, 1:=9};
28 }
29
30 constraint write_cn {
31     din[9:8] inside {[0:1]};
32 }
33
34 constraint read_cn {
35     if (old_op==2){
36
37         din[9:8]==3;
38     }
39     else if (old_op==3) {
40         din[9:8]==2;
41     }
42 }
43 constraint read_write_cn {
44     if (old_op==2'b00) {
45         din[9:8] inside {[0:1]};
46     }
47     else if (old_op==2'b01) {
48         din[9:8] dist {0:=40, 2:=60};
49     }
50     else if (old_op==2'b10) {
51         din[9:8] inside {[2:3]};
52     }
53     else if (old_op==2'b11) {
54         din[9:8] dist {0:=60, 2:=40};
55     }
56 }
57
58
59
60 }
61 function void post_randomize();
62 old_op=din[9:8];
63
64 endfunction
65
66
67 endclass
68
69 endpackage
```

Activate Windows  
Go to Settings to activate W

## RAM\_sequencer

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

## RAM\_rst\_sequence

```
● ● ●  
1 package RAM_reset_sequence;  
2 import uvm_pkg::*;  
3 `include "uvm_macros.svh"  
4 import RAM_seq_item::*;  
5 class reset_sequence extends uvm_sequence #(RAM_seq_item);  
6 `uvm_object_utils(reset_sequence)  
7 RAM_seq_item seq_item;  
8 function new(string name = "reset_sequence");  
9     super.new(name);  
10 endfunction  
11  
12 task body();  
13 seq_item = RAM_seq_item::type_id::create("seq_item");  
14 start_item(seq_item);  
15 seq_item.rst_n = 0;  
16 seq_item.rx_valid = 0;  
17 seq_item.din = 0;  
18 seq_item.dout = 0;  
19 seq_item.tx_valid = 0;  
20 finish_item(seq_item);  
21  
22 endtask  
23 endclass  
24 endpackage
```

## RAM\_write\_only\_sequence

```
● ● ●

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

## RAM\_write\_only\_sequence

```
● ● ●

1 package RAM_read_only_sequence;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import RAM_seq_item::*;
5 class read_only_sequence extends uvm_sequence #(RAM_seq_item);
6 `uvm_object_utils(read_only_sequence)
7 RAM_seq_item seq_item;
8 function new(string name = "read_only_sequence");
9     super.new(name);
10 endfunction
11
12 task body();
13     seq_item = RAM_seq_item::type_id::create("seq_item");
14     repeat(100) begin
15         start_item(seq_item);
16         seq_item.constraint_mode(0);
17         seq_item.reset_cn.constraint_mode(1);
18         seq_item.rx_valid_cn.constraint_mode(1);
19         seq_item.read_cn.constraint_mode(1);
20         assert(seq_item.randomize());
21         finish_item(seq_item);
22     end
23
24
25 endtask
26 endclass
27 endpackage
```

## RAM\_read\_write\_sequence

```
● ● ●

1 package RAM_read_write_sequence;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import RAM_seq_item::*;
5 class read_write_sequence extends uvm_sequence #(RAM_seq_item);
6 `uvm_object_utils(read_write_sequence)
7 RAM_seq_item seq_item;
8 function new(string name = "read_write_sequence");
9     super.new(name);
10 endfunction
11
12 task body();
13     seq_item = RAM_seq_item::type_id::create("seq_item");
14     repeat(100) begin
15         start_item(seq_item);
16         seq_item.constraint_mode(0);
17         seq_item.reset_cn.constraint_mode(1);
18         seq_item.rx_valid_cn.constraint_mode(1);
19         seq_item.read_write_cn.constraint_mode(1);
20         assert(seq_item.randomize());
21         finish_item(seq_item);
22     end
23
24
25 endtask
26 endclass
27 endpackage
```

## RAM\_driver

```
● ● ●

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

## RAM\_monitor

```
● ● ●

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

## RAM\_monitor\_golden\_model

```
1 package RAM_monitor_ref;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import RAM_seq_item::*;
5 import RAM_config::*;
6 class RAM_monitor_ref extends uvm_monitor;
7 `uvm_component_utils(RAM_monitor_ref)
8 virtual RAM_if_ref vif_ref;
9 RAM_seq_item seq_item;
10 uvm_analysis_port #(RAM_seq_item) mon_ref_ap;
11 RAM_config cfg;
12 function new(string name="RAM_monitor_ref", 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_ref_ap = new("mon_ref_ap", this);
19     if(!uvm_config_db#(RAM_config)::get(this, "", "cfg", cfg)) begin
20         `uvm_fatal("NOVIF","Virtual interface reference must be set for: via configuration database");
21     end
22 endfunction
23 function void connect_phase(uvm_phase phase);
24     super.connect_phase(phase);
25     vif_ref = cfg.vif_ref;
26 endfunction
27
28 task run_phase(uvm_phase phase);
29 super.run_phase(phase);
30 forever begin
31     seq_item = RAM_seq_item::type_id::create("seq_item");
32     @(negedge vif_ref.clk);
33     seq_item.rst_n = vif_ref.rst_n;
34     seq_item.rx_valid = vif_ref.rx_valid;
35     seq_item.din = vif_ref.din;
36     seq_item.dout = vif_ref.dout;
37     seq_item.tx_valid = vif_ref.tx_valid;
38     mon_ref_ap.write(seq_item);
39 end
40 endtask
41
42 endclass
43 endpackage
```

## RAM\_scoreboard

```
● ● ●
1 package RAM_scoreboard;
2 import uvm_pkg::*;
3 import RAM_seq_item::*;
4 import RAM_config::*;
5 `include "uvm_macros.svh"
6 class RAM_scoreboard extends uvm_scoreboard;
7 `uvm_component_utils(RAM_scoreboard)
8 int correct_count=0;
9 int error_count=0;
10 uvm_analysis_export #(RAM_seq_item) sb_export;
11 uvm_tlm_analysis_fifo #(RAM_seq_item) sb_fifo;
12 uvm_analysis_export #(RAM_seq_item) sb_export_ref;
13 uvm_tlm_analysis_fifo #(RAM_seq_item) sb_fifo_ref;
14 RAM_seq_item seq_item;
15 RAM_seq_item seq_item_ref;
16 function new(string name="RAM_scoreboard", 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     sb_export = new("sb_export", this);
23     sb_fifo = new("sb_fifo", this);
24     sb_export_ref = new("sb_export_ref", this);
25     sb_fifo_ref = new("sb_fifo_ref", this);
26 endfunction
27
28 function void connect_phase(uvm_phase phase);
29     super.connect_phase(phase);
30     sb_export.connect(sb_fifo.analysis_export);
31     sb_export_ref.connect(sb_fifo_ref.analysis_export);
32 endfunction
33
34 task run_phase(uvm_phase phase);
35     super.run_phase(phase);
36     forever begin
37         sb_fifo.get(seq_item);
38         sb_fifo_ref.get(seq_item_ref);
39
40         if(seq_item.dout!=seq_item_ref.dout&&seq_item.tx_valid== seq_item_ref.tx_valid) begin
41             `uvm_error("SCOREBOARD", $sformatf("Mismatch Detected! Expected Dout: %0h, Actual Dout: %0h, Expected Tx_Valid: %0b, Actual Tx_Valid: %0b", seq_item.dout, seq_item_ref.dout, seq_item.tx_valid, seq_item_ref.tx_valid));
42             error_count++;
43         end
44         else begin
45             correct_count++;
46         end
47     end
48 endtask
49 function void report_phase(uvm_phase phase);
50     super.report_phase(phase);
51     `uvm_info("SCOREBOARD", $sformatf("Correct Transactions: %0d, Error Transactions: %0d", correct_count, error_count), UVM_LOW);
52 endfunction
53
54 endclass
55 endpackage
```

## RAM\_coverage

```
1 package RAM_coverage;
2 import uvm_pkg::*;
3 `include "uvm_macros.svh"
4 import RAM_seq_item::*;
5 class RAM_coverage extends uvm_component;
6 `uvm_component_utils(RAM_coverage)
7 uvm_analysis_export #(RAM_seq_item) cov_export;
8 uvm_tlm_analysis_fifo #(RAM_seq_item) cov_fifo;
9 RAM_seq_item seq_item;
10
11 // =====covergroups=====
12
13 covergroup COV;
14 c1:coverpoint seq_item.din[9:8];
15     bins write_addr = {2'b00};
16     bins write_data = {2'b01};
17     bins read_addr = {2'b10};
18     bins read_data = {2'b11};
19     bins write_addr_data =(0=>1);
20     bins read_addr_data =(2=>3);
21     bins write_read = {0[*14]=>1[*14]=>2[*14]=>3[*14]};
22
23 }
24 rx_valid_cp:coverpoint seq_item.rx_valid{
25     bins rx_valid_0 = {0};
26     bins rx_valid_1 = {1};
27 }
28 tx_valid_cp:coverpoint seq_item.tx_valid{
29     bins tx_valid_0 = {0};
30     bins tx_valid_1 = {1};
31 }
32 cross rx_valid_cp,c1{
33     option.cross_auto_bin_max=0;
34     bins b1 = binsof(rx_valid_cp.rx_valid_0) && binsof(c1.write_addr);
35     bins b2 = binsof(rx_valid_cp.rx_valid_1) && binsof(c1.write_data);
36     bins b3 = binsof(rx_valid_cp.tx_valid_1) && binsof(c1.read_addr);
37     bins b4 = binsof(rx_valid_cp.tx_valid_1) && binsof(c1.read_data);
38 }
39 cross tx_valid_cp,c1{
40     option.cross_auto_bin_max=0;
41     bins read_data_valid = binsof(tx_valid_cp.tx_valid_1) && binsof(c1.read_data);
42
43 }
44
45 endgroup
46
47
48
49 function new(string name="RAM_coverage", uvm_component parent=null);
50     super.new(name, parent);
51     COV= new();
52 endfunction
53
54 function void build_phase(uvm_phase phase);
55     super.build_phase(phase);
56     cov_export = new("cov_export", this);
57     cov_fifo = new("cov_fifo", this);
58 endfunction
59
60 function void connect_phase(uvm_phase phase);
61     super.connect_phase(phase);
62     cov_export.connect(cov_fifo.analysis_export);
63 endfunction
64
65 task run_phase(uvm_phase phase);
66     super.run_phase(phase);
67     forever begin
68         cov_fifo.get(seq_item);
69         COV.sample();
70     end
71 endtask
72
73
74
75
76 endclass
77
78 endpackage
```

## RAM\_agent

```
● ● ●
1 package RAM_agent;
2 import uvm_pkg::*;
3 import RAM_driver::*;
4 import RAM_sequencer::*;
5 import RAM_monitor::*;
6 import RAM_config::*;
7 import RAM_seq_item::*;
8 `include "uvm_macros.svh"
9 class RAM_agent extends uvm_agent;
10 `uvm_component_utils(RAM_agent)
11   RAM_driver driver;
12   RAM_sequencer sequencer;
13   RAM_monitor monitor;
14   RAM_config cfg;
15   uvm_analysis_port #(RAM_seq_item) agent_ap;
16
17   function new(string name="RAM_agent", 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     if(!uvm_config_db#(RAM_config)::get(this, "", "cfg", cfg)) begin
24       `uvm_fatal("NOVIF", "Virtual interface must be set for: via configuration database");
25     end
26     driver = RAM_driver::type_id::create("driver", this);
27     sequencer = RAM_sequencer::type_id::create("sequencer", this);
28     monitor = RAM_monitor::type_id::create("monitor", this);
29     agent_ap = new("agent_ap", this);
30   endfunction
31
32   function void connect_phase(uvm_phase phase);
33     super.connect_phase(phase);
34     driver.vif = cfg.vif;
35     monitor.vif = cfg.vif;
36     monitor.mon_ap.ap.connect(agent_ap);
37     driver.seq_item_port.connect(sequencer.seq_item_export);
38   endfunction
39 endclass
40
41
42 endpackage
```

## SPI\_slave\_env

```
● ● ●

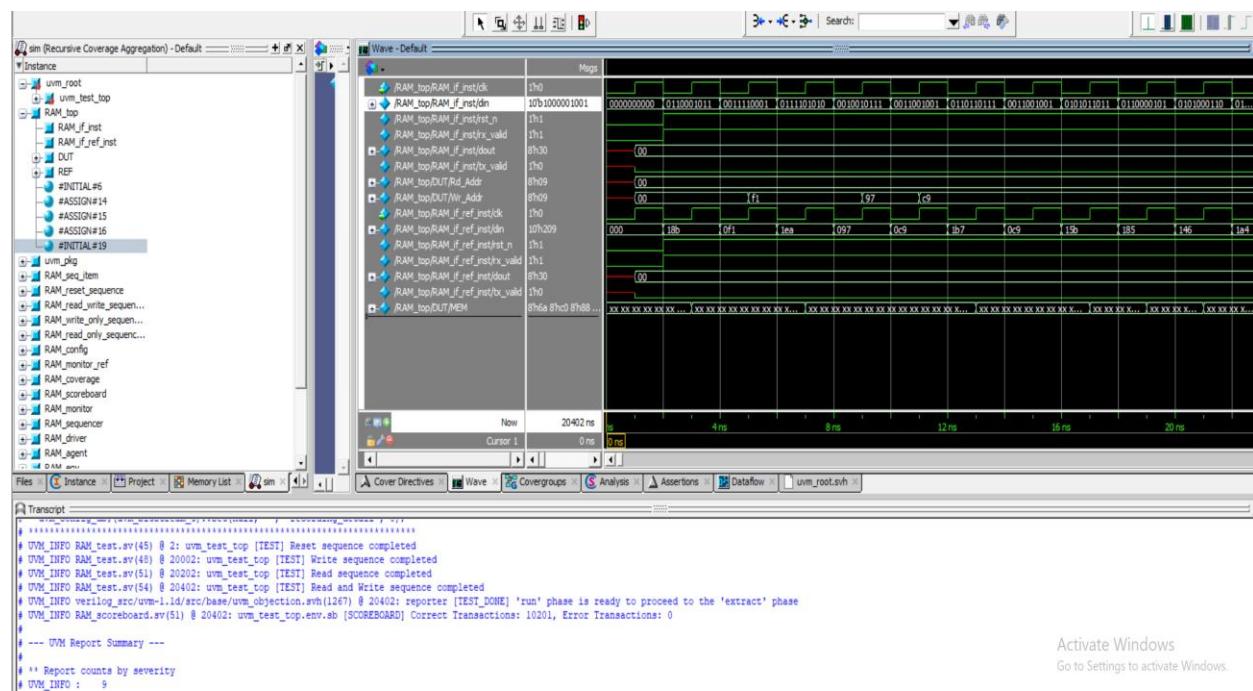
1 package RAM_env;
2 import uvm_pkg::*;
3 import RAM_seq_item::*;
4 import RAM_agent::*;
5 import RAM_scoreboard::*;
6 import RAM_coverage::*;
7 import RAM_monitor_ref::*;
8 `include "uvm_macros.svh"
9 class RAM_env extends uvm_env;
10 `uvm_component_utils(RAM_env)
11 RAM_agent agent;
12 RAM_scoreboard sb;
13 RAM_monitor_ref mon_ref;
14 RAM_coverage cov;
15 function new(string name="RAM_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     agent = RAM_agent::type_id::create("agent", this);
22     sb = RAM_scoreboard::type_id::create("sb", this);
23     cov = RAM_coverage::type_id::create("cov", this);
24     mon_ref = RAM_monitor_ref::type_id::create("mon_ref", this);
25 endfunction
26 function void connect_phase(uvm_phase phase);
27     super.connect_phase(phase);
28     agent.agent_ap.connect(sb.sb_export);
29     agent.agent_ap.connect(cov.cov_export);
30     mon_ref.mon_ref_ap.connect(sb.sb_export_ref);
31 endfunction
32
33 endclass
34 endpackage
```

## SPI\_slave\_test

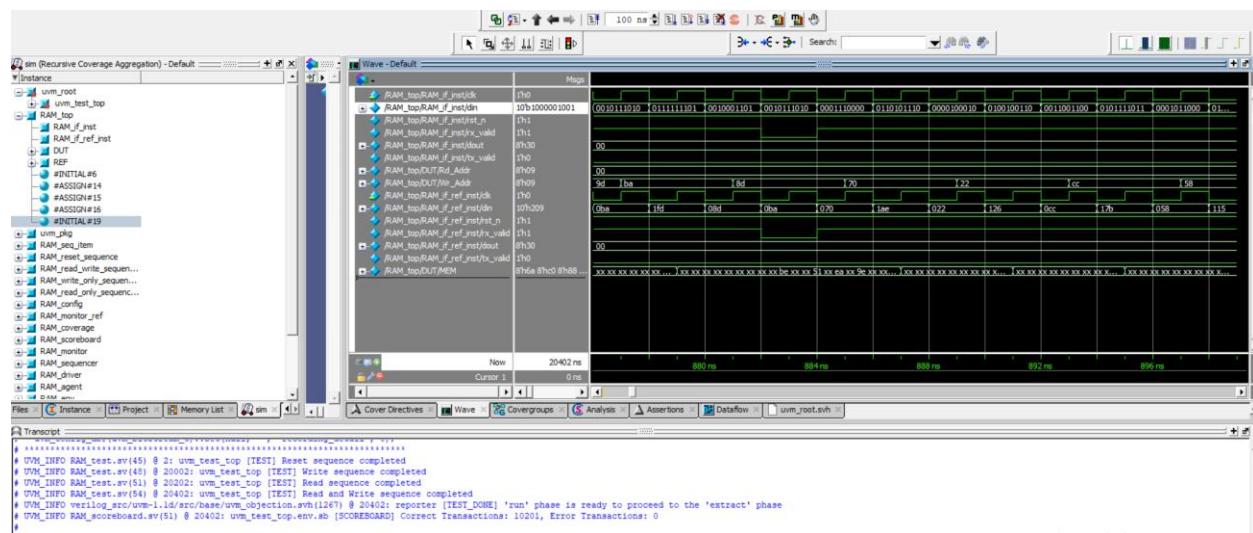
```
1 package RAM_test;
2 import uvm_pkg::*;
3 import RAM_env::*;
4 import RAM_config::*;
5 import RAM_read_only_sequence::*;
6 import RAM_write_only_sequence::*;
7 import RAM_read_write_sequence::*;
8 import RAM_reset_sequence::*;
9 `include "uvm_macros.svh"
10 class RAM_test extends uvm_test;
11
12 `uvm_component_utils(RAM_test)
13 RAM_env env;
14 RAM_config cfg;
15 read_only_sequence read_seq;
16 write_only_sequence write_seq;
17 read_write_sequence read_write_seq;
18 reset_sequence reset_seq;
19
20 function new(string name="RAM_test", uvm_component parent=null);
21     super.new(name, parent);
22 endfunction
23
24 function void build_phase(uvm_phase phase);
25     super.build_phase(phase);
26     env = RAM_env::type_id::create("env", this);
27     cfg = RAM_config::type_id::create("cfg", this);
28     read_seq = read_only_sequence::type_id::create("read_seq", this);
29     write_seq = write_only_sequence::type_id::create("write_seq", this);
30     read_write_seq = read_write_sequence::type_id::create("read_write_seq", this);
31     reset_seq = reset_sequence::type_id::create("reset_seq", this);
32     if(!uvm_config_db#(virtual RAM_if)::get(this, "", "vif_DUT", cfg.vif)) begin
33         `uvm_fatal("NOVIF","Virtual interface must be set for: via configuration database");
34     end
35     if(!uvm_config_db#(virtual RAM_if_ref)::get(this, "", "vif_REF", cfg.vif_ref)) begin
36         `uvm_fatal("NOVIF","Virtual interface reference must be set for: via configuration database");
37     end
38     uvm_config_db#(RAM_config)::set(this, "", "cfg", cfg);
39 endfunction
40 task run_phase(uvm_phase phase);
41     super.run_phase(phase);
42     phase.raise_objection(this);
43     //reset sequence
44     reset_seq.start(env.agent.sequencer);
45     `uvm_info("TEST", "Reset sequence completed", UVM_LOW);
46     //write sequence
47     write_seq.start(env.agent.sequencer);
48     `uvm_info("TEST", "Write sequence completed", UVM_LOW);
49     //read sequence
50     read_seq.start(env.agent.sequencer);
51     `uvm_info("TEST", "Read sequence completed", UVM_LOW);
52     //read and write sequence
53     read_write_seq.start(env.agent.sequencer);
54     `uvm_info("TEST", "Read and Write sequence completed", UVM_LOW);
55
56     phase.drop_objection(this);
57 endtask
58
59
60 endclass
61 endpackage
```

# QuestaSim Snippets

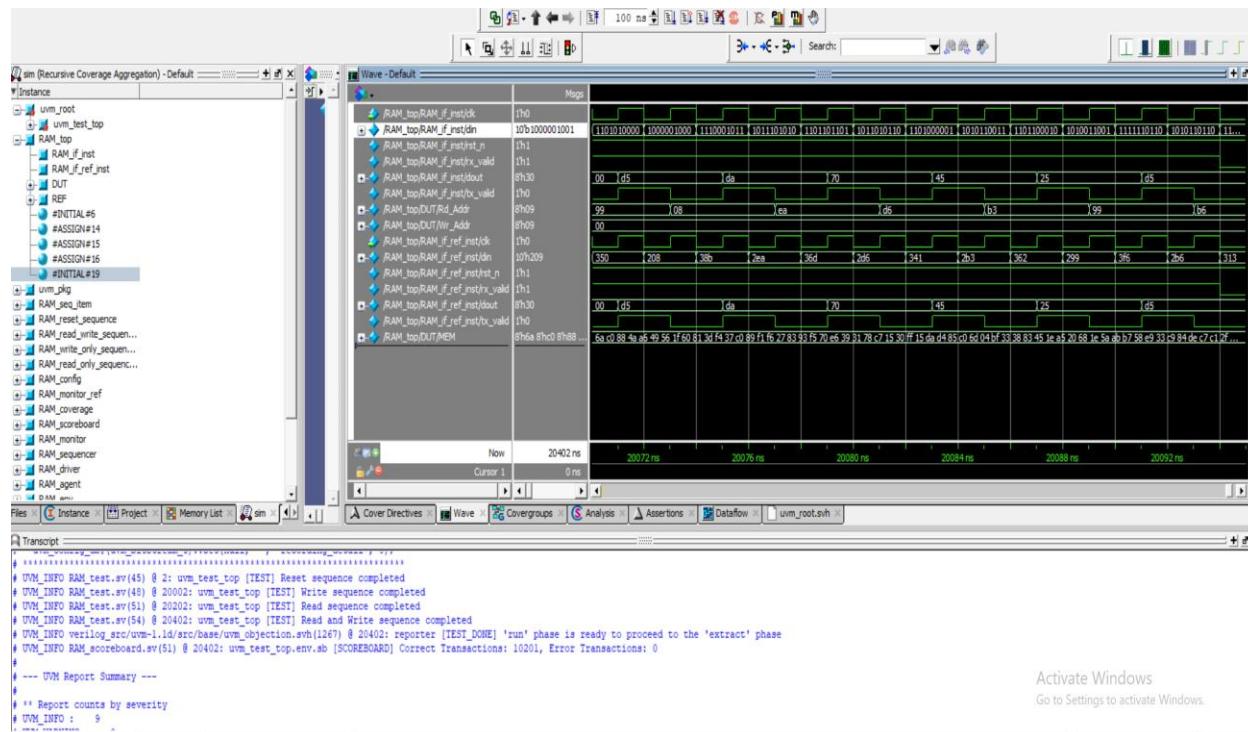
## Reset\_sequence



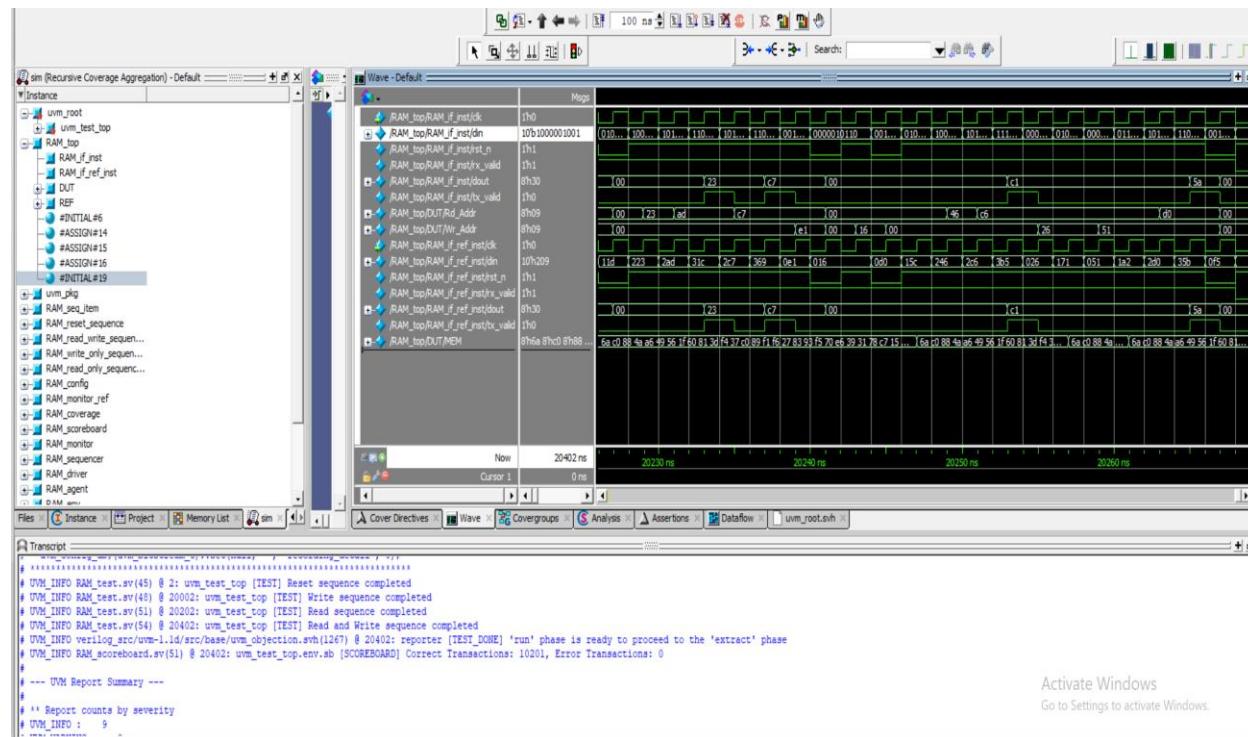
## Write\_only\_sequence



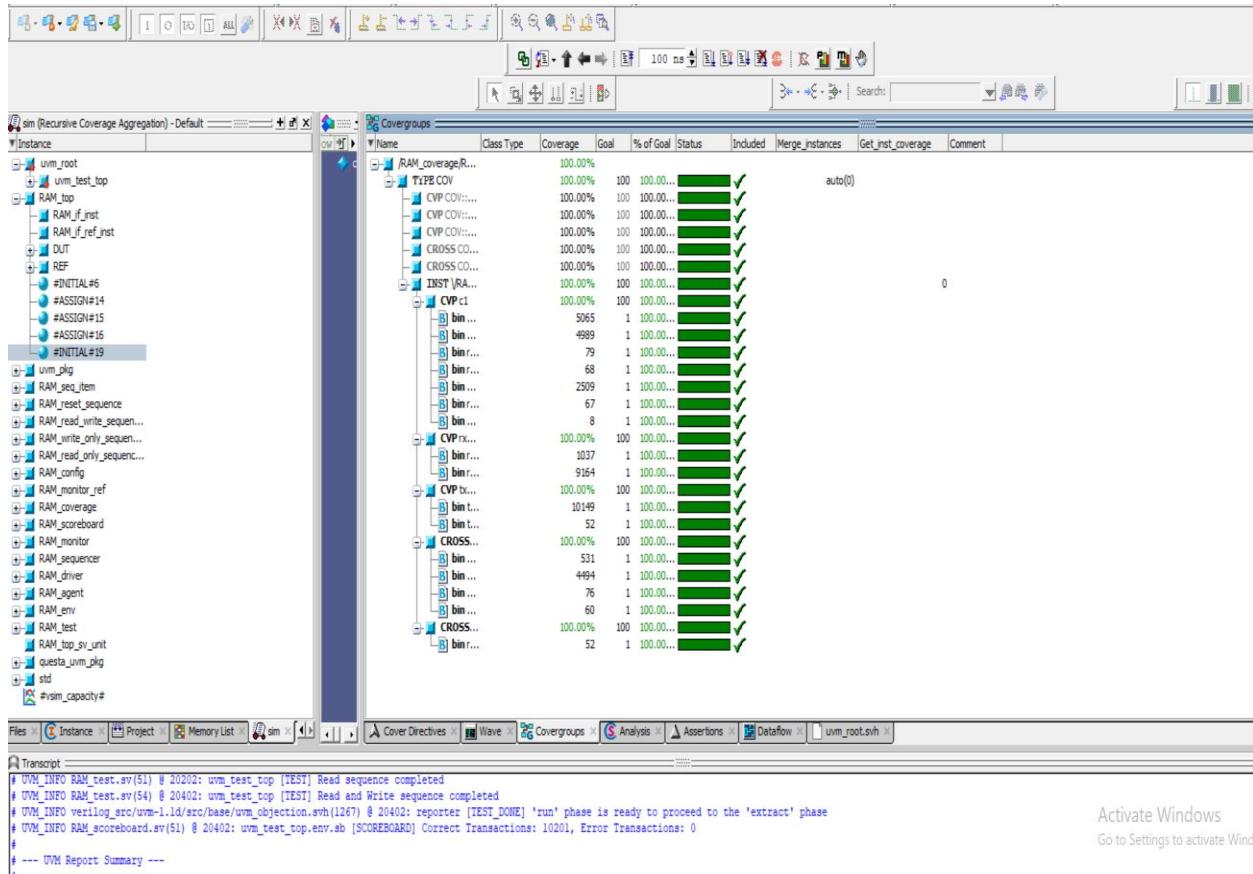
## Read\_only\_sequence



## Read\_write\_sequence



# Functional Coverage



```

==== Instance: /RAM_coverage
==== Design Unit: work.RAM_coverage
=====
Covergroup Coverage:
  Covergroups      1    na    na  100.00%
  | Coverpoints/Crosses   5    na    na  na
  |   Covergroup_Bins     16   16    0  100.00%
=====
Covergroup          Metric   Goal   Bins Status
=====
TYPE /RAM_coverage/RAM_coverage/COV      100.00%  100   - Covered
  covered/total bins:                 16    16   -
  missing/total bins:                0     16   -
  % Hit:                            100.00%  100   -
  Coverpoint c1                     100.00%  100   - Covered
    covered/total bins:               7     7   -
    missing/total bins:              0     7   -
    % Hit:                            100.00%  100   -
  Coverpoint rx_valid_cp            100.00%  100   - Covered
    covered/total bins:               2     2   -
    missing/total bins:              0     2   -
    % Hit:                            100.00%  100   -
  Coverpoint tx_valid_cp            100.00%  100   - Covered
    covered/total bins:               2     2   -
    missing/total bins:              0     2   -
    % 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   -
  Cross #cross_1#
    covered/total bins:               1     1   -
    missing/total bins:              0     1   -
    % Hit:                            100.00%  100   -
Covergroup instance \/RAM_coverage::RAM_coverage::COV
  covered/total bins:                 16    16   -
  missing/total bins:                0     16   -
  % Hit:                            100.00%  100   -
  Coverpoint c1                     100.00%  100   - Covered
    covered/total bins:               7     7   -
    missing/total bins:              0     7   -
    % Hit:                            100.00%  100   -
    bin write_addr                  5065   1   - Covered
    bin write_data                  4989   1   - Covered
    bin read_addr                  79     1   - Covered
    bin read_data                  68     1   - Covered
    bin write_addr_data             2509   1   - Covered
    bin read_addr_data             67     1   - Covered
    bin write_read                 8      1   - Covered
  Coverpoint rx_valid_cp            100.00%  100   - Covered
    covered/total bins:               2     2   -
    missing/total bins:              0     2   -
    % Hit:                            100.00%  100   -
    bin rx_valid_0                  1037   1   - Covered
    bin rx_valid_1                  9164   1   - Covered
  Coverpoint tx_valid_cp            100.00%  100   - Covered
    covered/total bins:               2     2   -
    missing/total bins:              0     2   -
    % Hit:                            100.00%  100   -
    bin tx_valid_0                  10149  1   - Covered
    bin tx_valid_1                  52     1   - Covered
  Cross #cross_0#
    covered/total bins:               4     4   -
    missing/total bins:              0     4   -
    % Hit:                            100.00%  100   -
    Auto, Default and User Defined Bins:
      bin b1                         531    1   - Covered
      bin b2                         4494   1   - Covered
      bin b3                         76     1   - Covered
      bin b4                         60     1   - Covered
  Cross #cross_1#
    covered/total bins:               1     1   -
    missing/total bins:              0     1   -
    % Hit:                            100.00%  100   -
    Auto, Default and User Defined Bins:
      bin read_data_valid            52     1   - Covered

```

## Assertion Summary

Feature	Assertion
<b>Whenever reset is asserted, the output signals (tx_valid and dout) are low</b>	<code>@(posedge DUT_if.clk) !DUT_if.rst_n  =&gt; ((DUT_if.dout)==8'b00000000 &amp;&amp;DUT_if.tx_valid==0);</code>
<b>During address or data input phases (write_add_seq, write_data_seq, read_add_seq), the tx_valid signal must remain deasserted.</b>	<code>@(posedge DUT_if.clk) disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]!=2'b11)  =&gt; (DUT_if.tx_valid)==0;</code>
<b>After a read_data_seq occurs, the tx_valid signal must rise to indicate valid output and after it rises by one clock cycle, it should eventually fall.</b>	<code>@(posedge DUT_if.clk) disable iff (!DUT_if.rst_n) (\$past(DUT_if.din[9:8])==2'b10&amp;&amp; DUT_if.din[9:8]==2'b11)  =&gt; (\$rose(DUT_if.tx_valid)) =&gt;##[1:\$](\$fell(DUT_if.tx_valid));</code>
<b>Every Write Address operation must be eventually followed by a Write Data operation.</b>	<code>@(posedge DUT_if.clk) disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]==2'b00)  =&gt; ##[1:\$] (DUT_if.din[9:8]==2'b01);</code>
<b>Every Read Address operation must be eventually followed by a Read Data operation.</b>	<code>@(posedge DUT_if.clk) disable iff (!DUT_if.rst_n) (DUT_if.din[9:8]==2'b10)  =&gt; ##[1:\$] (DUT_if.din[9:8]==2'b11);</code>

## Assertion Coverage

Cover Directives														
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative
/RAM_top/DUT/RAM_sva_inst/cover__read_addr_data_check	SVA	✓	Off	56	1	Unl...	1	100%		✓	0	0	0 ns	
/RAM_top/DUT/RAM_sva_inst/cover__write_addr_data_check	SVA	✓	Off	3334	1	Unl...	1	100%		✓	0	0	0 ns	
/RAM_top/DUT/RAM_sva_inst/cover_tx_valid_high_check	SVA	✓	Off	30	1	Unl...	1	100%		✓	0	0	0 ns	
/RAM_top/DUT/RAM_sva_inst/cover_tx_valid_low_check	SVA	✓	Off	8216	1	Unl...	1	100%		✓	0	0	0 ns	
/RAM_top/DUT/RAM_sva_inst/cover__rst_check	SVA	✓	Off	1017	1	Unl...	1	100%		✓	0	0	0 ns	

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

Directive Coverage:
  Directives      5      5      0  100.00%
DIRECTIVE COVERAGE:
-----
Name          Design Design   Lang File(Line)      Hits Status
           Unit    UnitType
-----
/RAM_top/DUT/RAM_sva_inst/cover__read_addr_data_check
                           RAM_sva Verilog SVA  RAM_sva.sv(35)      56 Covered
/RAM_top/DUT/RAM_sva_inst/cover__write_addr_data_check
                           RAM_sva Verilog SVA  RAM_sva.sv(28)    3334 Covered
/RAM_top/DUT/RAM_sva_inst/cover_tx_valid_high_check
                           RAM_sva Verilog SVA  RAM_sva.sv(21)      30 Covered
/RAM_top/DUT/RAM_sva_inst/cover_tx_valid_low_check
                           RAM_sva Verilog SVA  RAM_sva.sv(15)    8216 Covered
/RAM_top/DUT/RAM_sva_inst/cover__rst_check
                           RAM_sva Verilog SVA  RAM_sva.sv(8)     1017 Covered
=====

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

# Code Coverage

```
1  === Instance: /RAM_top/DUT
2  === Design Unit: work.RAM
3  =====
4  Branch Coverage:
5      Enabled Coverage          Bins   Hits   Misses  Coverage
6  -----  -----
7      Branches                  8      8      0    100.00%
8
9  =====Branch Details=====
10
11 Branch Coverage for instance /RAM_top/DUT
12
13     Line      Item           Count   Source
14     ---      ---           ----   -----
15 File RAM.sv
16 -----IF Branch-----
17     7                      10201  Count coming in to IF
18     7         1              1017    if (~DUT_if.rst_n) begin
19
20     13        1              9184    else begin //fix
21
22 Branch totals: 2 hits of 2 branches = 100.00%
23
24 -----IF Branch-----
25     14                     9184    Count coming in to IF
26     14         1              8241    if (DUT_if.rx_valid) begin
27
28                     943    All False Count
29 Branch totals: 2 hits of 2 branches = 100.00%
30
31 -----CASE Branch-----
32     15                     8241    Count coming in to CASE
33     16         1              4090    2'b00 : Wr_Addr <= DUT_if.din[7:0];
34
35     17         1              4071    2'b01 : MEM[Wr_Addr] <= DUT_if.din[7:0];
36
37     18         1              46      2'b10 : Rd_Addr <= DUT_if.din[7:0];
38
39     19         1              34      2'b11 : DUT_if.dout <= MEM[Rd_Addr];//fix
40
41 Branch totals: 4 hits of 4 branches = 100.00%
42
43
44 Expression Coverage:
45     Enabled Coverage          Bins   Covered   Misses  Coverage
46  -----  -----
47     Expressions                3      3      0    100.00%
```

```

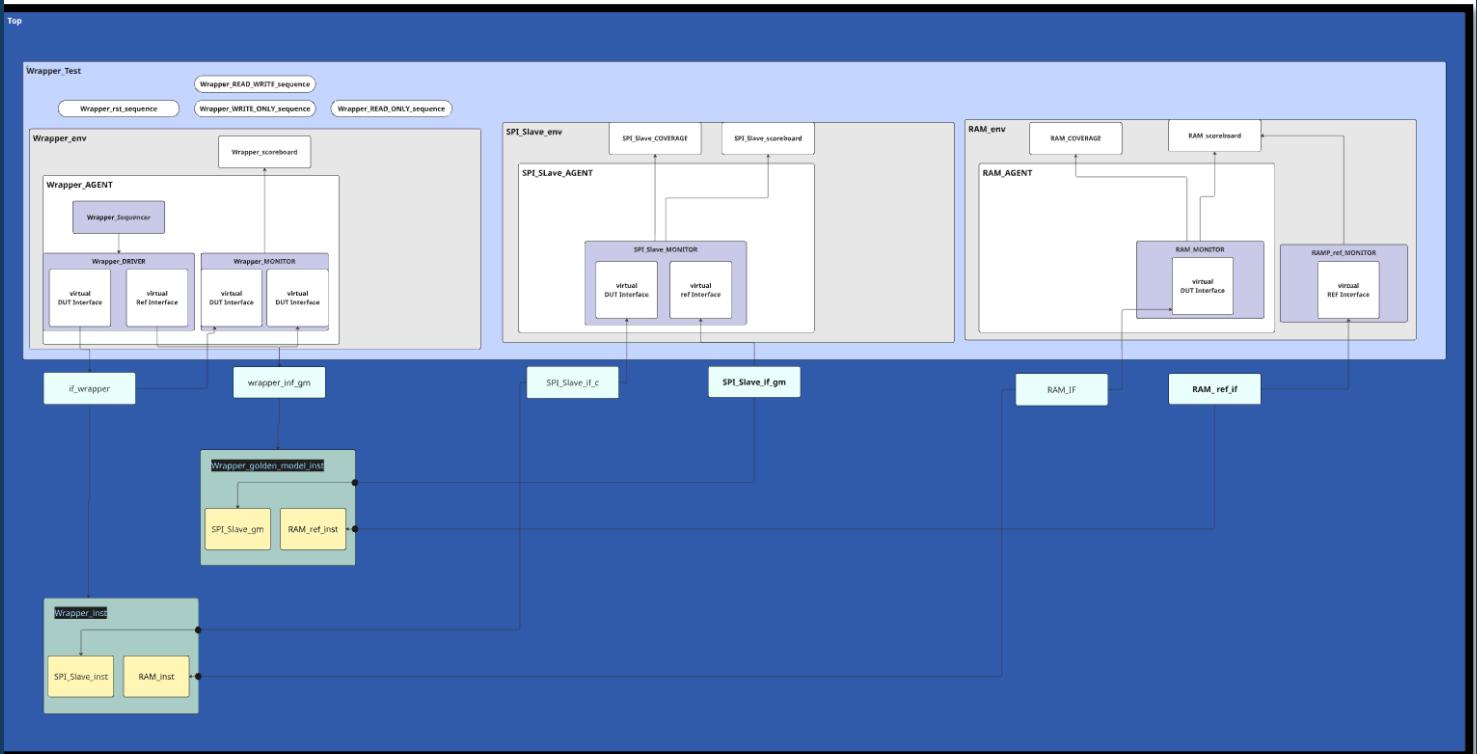
1 Statement Coverage:
2   Enabled Coverage           Bins    Hits    Misses  Coverage
3   -----      ----  -----  -----
4   Statements                  10      10      0  100.00%
5
6 =====Statement Details=====
7
8 Statement Coverage for instance /RAM_top/DUT --
9
10  Line     Item            Count  Source
11  ---     ---
12  File RAM.sv
13  1                      module RAM (RAM_if.DUT DUT_if);
14
15  2
16
17  3                      logic[7:0] MEM [255:0];
18
19  4                      logic [7:0] Rd_Addr, Wr_Addr;
20
21  5
22
23  6          1          10201  always @(posedge DUT_if.clk) begin
24
25          7
26          if (~DUT_if.rst_n) begin
27          8          1          1017    DUT_if.dout <= 0;
28
29          9          1          1017    DUT_if.tx_valid <= 0;
30
31          10         1          1017    Rd_Addr <= 0;
32
33          11         1          1017    Wr_Addr <= 0;
34
35          12
36
37          13
38          else begin //fix
39
40          14
41          if (DUT_if.rx_valid) begin
42
43          15
44          case (DUT_if.din[9:8])
45
46          16         1          4090    2'b00 : Wr_Addr <= DUT_if.din[7:0];
47
48          17         1          4071    2'b01 : MEM[Wr_Addr] <= DUT_if.din[7:0];
49
50          18         1          46      2'b10 : Rd_Addr <= DUT_if.din[7:0];
51
52          19         1          34      2'b11 : DUT_if.dout <= MEM[Rd_Addr];//fix
53
54
55          20
56
57          21
58          0;          1          9184    DUT_if.tx_valid <= (DUT_if.din[9] && DUT_if.din[8] && DUT_if.rx_valid)? 1'b1 : 1'b
59

```

```
1 =====
2 === Instance: /RAM_top/RAM_if_inst
3 === Design Unit: work.RAM_if
4 =====
5 Toggle Coverage:
6      Enabled Coverage          Bins    Hits    Misses  Coverage
7      -----                  ---     ---     ---   -----
8      Toggles                   44      44      0   100.00%
9
10 =====Toggle Details=====
11
12 Toggle Coverage for instance /RAM_top/RAM_if_inst --
13
14
15
16
17
18
19
20
21
22
23 Total Node Count      =      22
24 Toggled Node Count   =      22
25 Untoggled Node Count =      0
26
27 Toggle Coverage      =      100.00% (44 of 44 bins)
```

# Part 3 SPI-Wrapper Verification

## UVM Diagram



The top module instantiates six interfaces, one for each design unit and one for their golden models. Additionally, it instantiates the main SPI wrapper design and its golden “reference” model where they instantiate the SPI & RAM or their golden models, being passed their respective interfaces corresponding to its wrapper. Finally, all interfaces are sent to the configuration database to be retrieved later in the verification process, and it finishes its operation by running the wrapper test using the “run\_test” function.

In turn, the wrapper test gets the interfaces from the config db and sends them back in different configuration objects containing their respective virtual interfaces and the “is\_active” enum variable that later controls the environment type. The test also creates three environments for each design, which will be elaborated on later, and starts each sequence from the four sequences stated above and shown in the diagram, and sends the constrained sequence items through the sequencer to the driver.

Inside each and every environment, three main things occur, which are creating the scoreboard, the coverage collector, and the UVM Agent.

The scoreboard is mainly responsible for comparing the outputs from the sequence items sent to it with the outputs from reference sequence items for each module through two TLM FIFOs and analysis ports, and prints any mismatches found.

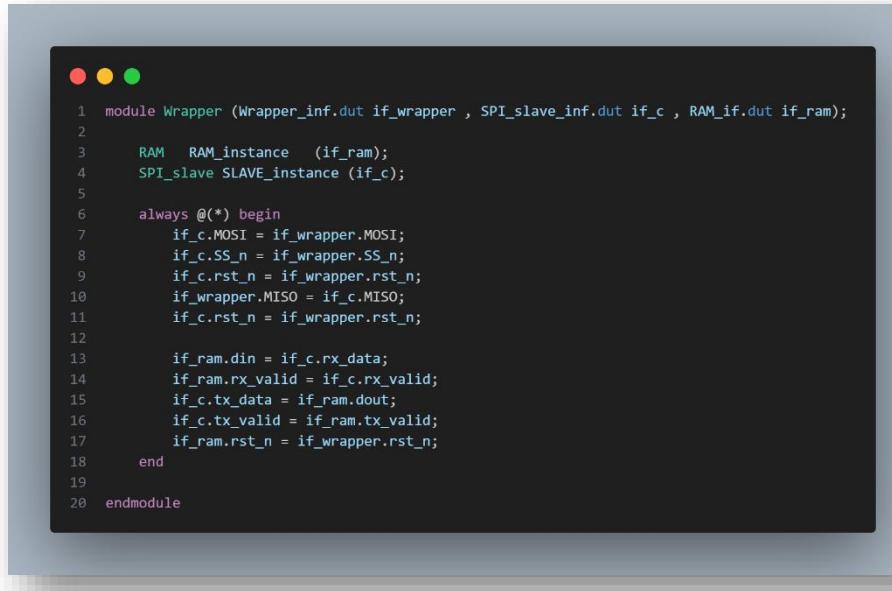
The coverage collector does pretty much the same with taking these sequence items from the TLM FIFOs and through analysis ports, and checking the functional coverage requirements.

The UVM agent of each environment should then get the config objects and see whether these environments are active or passive. If active as the wrapper, then the driver, sequencer, & monitor are created, or if passive, so it is only monitored by them as the SPI & the RAM.

Each agent assigns their existing virtual interfaces to the interfaces from the configuration objects, so the driver can drive the stimulus to the interface, and the monitor can send the response to the scoreboard and the coverage collector.

Finally, until all repetitions end for each sequence, the final objection is dropped from the test, and the UVM test finishes successfully.

## RTL Code



```
1 module Wrapper (Wrapper_if.dut if_wrapper , SPI_slave_if.dut if_c , RAM_if.dut if_ram);
2
3     RAM    RAM_instance  (if_ram);
4     SPI_slave SLAVE_instance (if_c);
5
6     always @(*) begin
7         if_c.MOSI = if_wrapper.MOSI;
8         if_c.SS_n = if_wrapper.SS_n;
9         if_c.rst_n = if_wrapper.rst_n;
10        if_wrapper.MISO = if_c.MISO;
11        if_c.rst_n = if_wrapper.rst_n;
12
13        if_ram.din = if_c.rx_data;
14        if_ram.rx_valid = if_c.rx_valid;
15        if_c.tx_data = if_ram.dout;
16        if_c.tx_valid = if_ram.tx_valid;
17        if_ram.rst_n = if_wrapper.rst_n;
18    end
19
20 endmodule
```

No major design changes are made, just some design adjustments to suit the whole verification process.

# Code Snippets

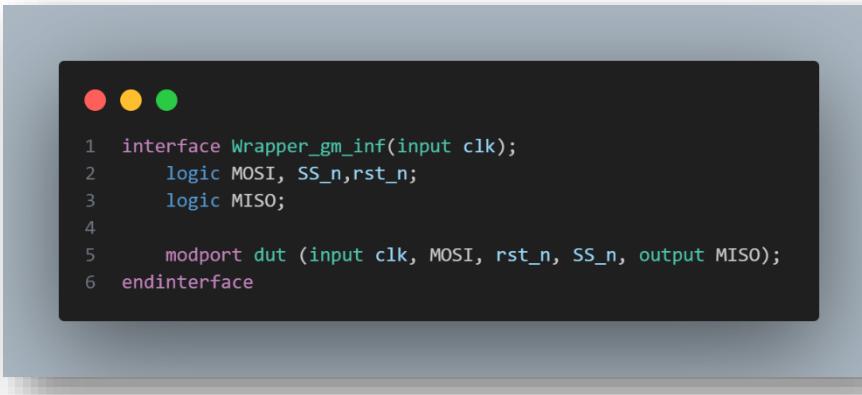
## Top module

```
 1 `include "uvm_macros.svh"
 2 import uvm_pkg::*;
 3 import Wrapper_test_pkg::*;
 4 module top;
 5
 6   bit clk;
 7   initial begin
 8     clk = 0;
 9     forever #1 clk = ~clk;
10   end
11
12   SPI_slave_if if_c(.clk(clk));
13   SPI_slave_gm_if if_gm(.clk(clk));
14   Wrapper_if if_wrapper(.clk(clk));
15   Wrapper_gm_if wrapper_inf_gm(.clk(clk));
16   RAM_if if_ram(.clk(clk));
17   RAM_if_ref RAM_if_ref_inst(.clk(clk));
18
19   Wrapper_golden_model Wrapper_golden_model_inst(wrapper_inf_gm, if_gm, RAM_if_ref_inst);
20
21   Wrapper Wrapper_inst(if_wrapper, if_c, if_ram);
22
23   Wrapper_sva    sva_Wrapper(if_wrapper);
24   RAM_sva       sva_RAM(if_ram);
25   SPI_slave_sva sva_SPI_slave(if_c);
26
27   initial begin
28     uvm_config_db #(virtual SPI_slave_if)::set(null, "uvm_test_top", "vif", if_c);
29     uvm_config_db #(virtual SPI_slave_gm_if)::set(null, "uvm_test_top", "vif_gm", if_gm);
30
31     uvm_config_db#(virtual RAM_if)::set(null, "uvm_test_top", "vif_DUT", if_ram);
32     uvm_config_db#(virtual RAM_if_ref)::set(null, "uvm_test_top", "vif_REF", RAM_if_ref_inst);
33
34     uvm_config_db#(virtual Wrapper_if)::set(null, "uvm_test_top", "wrap_vif", if_wrapper);
35     uvm_config_db#(virtual Wrapper_gm_if)::set(null, "uvm_test_top", "wrap_ref", wrapper_inf_gm);
36
37     run_test("Wrapper_test");
38   end
39 endmodule
40
```

## Module Interface

```
 1 interface Wrapper_inf(input clk);
 2   logic MOSI, SS_n, rst_n;
 3   logic MISO;
 4
 5   modport dut (input clk, MOSI, rst_n, SS_n, output MISO);
 6 endinterface
```

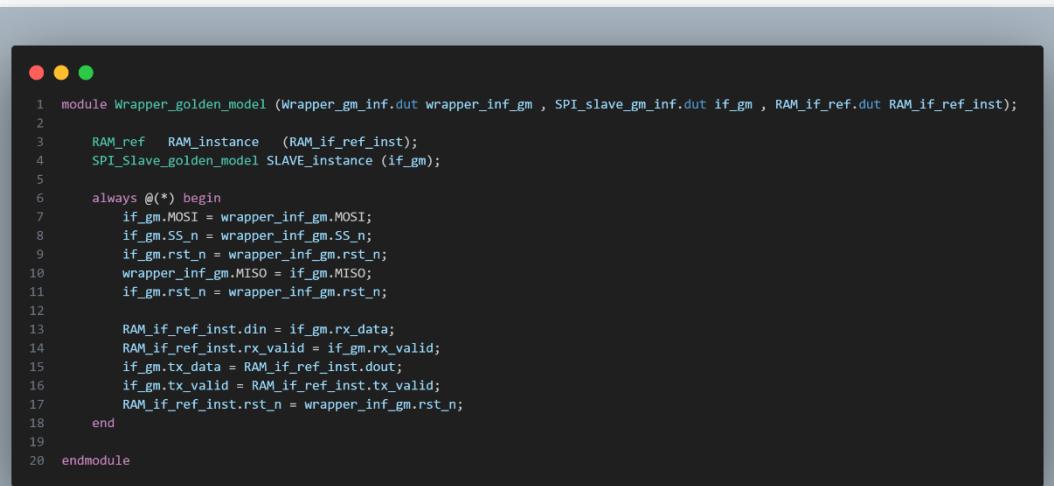
## Golden Model Interface



```
● ● ●

1 interface Wrapper_gm_inf(input clk);
2     logic MOSI, SS_n,rst_n;
3     logic MISO;
4
5     modport dut (input clk, MOSI, rst_n, SS_n, output MISO);
6 endinterface
```

## Golden Model Module



```
● ● ●

1 module Wrapper_golden_model (Wrapper_gm_inf.dut wrapper_inf_gm , SPI_slave_gm_inf.dut if_gm , RAM_if_ref.dut RAM_if_ref_inst);
2
3     RAM_ref    RAM_instance    (RAM_if_ref_inst);
4     SPI_Slave_golden_model SLAVE_instance (if_gm);
5
6     always @(*) begin
7         if_gm.MOSI = wrapper_inf_gm.MOSI;
8         if_gm.SS_n = wrapper_inf_gm.SS_n;
9         if_gm.rst_n = wrapper_inf_gm.rst_n;
10        wrapper_inf_gm.MISO = if_gm.MISO;
11        if_gm.rst_n = wrapper_inf_gm.rst_n;
12
13        RAM_if_ref_inst.din = if_gm.rx_data;
14        RAM_if_ref_inst.rx_valid = if_gm.rx_valid;
15        if_gm.tx_data = RAM_if_ref_inst.dout;
16        if_gm.tx_valid = RAM_if_ref_inst.tx_valid;
17        RAM_if_ref_inst.rst_n = wrapper_inf_gm.rst_n;
18    end
19
20 endmodule
```

## Wrapper\_sva

```
1 module Wrapper_sva (Wrapper_inf.dut if_wrapper);
2
3   property p_reset;
4     @ (posedge if_wrapper.clk)
5       (~if_wrapper.rst_n |=> (if_wrapper.MISO == 'b0))
6   endproperty
7   assert property(p_reset);
8   cover property(p_reset);
9
10  property MISO_stable_when_not_read;
11    @ (posedge if_wrapper.clk)
12      disable iff (!if_wrapper.rst_n) // disable when reset active or slave not selected
13      (if_wrapper.SS_n ## 1 !if_wrapper.SS_n ## 1 (if_wrapper.MOSI ## 1 if_wrapper.MOSI ## 1 if_wrapper.MOSI)) |=> $stable(if_wrapper.MISO) throughout (!if_wrapper.SS_n);
14  endproperty
15
16  assert property(MISO_stable_when_not_read);
17  cover property(MISO_stable_when_not_read);
18 endmodule
```

## SPI\_RAM\_config “Changed to suit the whole verification process”

```
1 package SPI_RAM_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class SPI_RAM_config extends uvm_object;
5     `uvm_object_utils(SPI_RAM_config)
6
7     virtual SPI_slave_if SPI_vif;
8     virtual SPI_slave_gm_if SPI_vif_gm;
9     virtual RAM_if vif;
10    virtual RAM_if_ref vif_ref;
11    virtual Wrapper_inf wrapper_inf;
12    virtual Wrapper_gm_inf wrapper_inf_gm;
13    uvm_active_passive_enum is_active;
14    function new(string name = "SPI_RAM_config");
15      super.new(name);
16    endfunction
17  endclass
18 endpackage
```

## Wrapper\_sequenceitem

```
 1 package Wrapper_sequenceitem_pkg;
 2   import uvm_pkg::*;
 3   import shared_package::*;
 4   `include "uvm_macros.svh"
 5   typedef enum bit [2:0] {
 6     WRITE_ADD = 3'b000,
 7     WRITE_DATA = 3'b001,
 8     READ_ADD = 3'b110,
 9     READ_DATA = 3'b111
10   } cmd_t;
11
12 class Wrapper_sequenceitem extends uvm_sequence_item;
13   `uvm_object_utils(Wrapper_sequenceitem)
14   // number of transactions
15   cmd_t cmd_seq_old = WRITE_ADD;
16   rand cmd_t cmd_seq = WRITE_DATA;
17   rand bit [10:0] MOSI_arr;
18   logic [10:0] old_MOSI_arr = 11'b1110000000;
19   rand logic SS_n = 1;
20   rand logic rst_n;
21   logic MOSI;
22   logic MISO;
23
24   int counter = 0;
25   bit [10:0] active_MOSI_word;
26
27   function new(string name = "Wrapper_sequenceitem");
28     super.new(name);
29   endfunction
30
31   function string convert2string();
32     return $sformatf("Wrapper_sequenceitem: %s: MOSI=%b, rst_n=%b, SS_n=%b",super.convert2string(), MOSI, rst_n, SS_n);
33   endfunction
34
35   function void pre_randomize();
36     if(SS_n == 0) begin
37       MOSI_arr.rand_mode(0);
38       ordering_rules.constraint_mode(0);
39       cmd_seq.rand_mode(0);
40     end
41     else begin
42       MOSI_arr.rand_mode(1);
43       ordering_rules.constraint_mode(1);
44       cmd_seq.rand_mode(1);
45     end
46   endfunction
```

```

1      constraint ordering_rules {
2
3          if(cmd_seq.old == WRITE_ADD){
4              cmd_seq inside {WRITE_ADD, WRITE_DATA};
5          }
6          else if(cmd_seq.old == WRITE_DATA){
7              cmd_seq dist {READ_ADD:= 60,WRITE_ADD:= 40};
8          }
9          else if(cmd_seq.old == READ_ADD){
10             cmd_seq == READ_DATA;
11         }
12         else if(cmd_seq.old == READ_DATA){
13             cmd_seq dist {WRITE_ADD:= 60,READ_ADD:= 40};
14         }
15     }
16 }
17
18
19 // Random payload for remaining bits
20 constraint payload {
21     MOSI_arr[7:0] inside {[8'h00 : 8'hFF]};
22 }
23
24 constraint c1 {
25     rst_n dist {0 := 1, 1 := 99};
26
27     if(counter == 23 && old莫斯I_arr[10:8] == 3'b111){
28         SS_n == 1;
29     }
30     if(counter == 13 && old莫斯I_arr[10:8] != 3'b111){
31         SS_n == 1;
32     }
33     if(!(counter == 23 && old莫斯I_arr[10:8] == 3'b111) && !(counter == 13 && old莫斯I_arr[10:8] != 3'b111)){
34         SS_n == 0;
35     }
36 }
37
38 function void post_randomize();
39     if (rst_n == 0) begin
40         counter = 0;
41         SS_n = 1;
42     end
43     else begin
44         // === SPI active (chip selected) ===
45         if (SS_n == 0) begin
46             if(counter == 0) begin
47                 MOSI_arr = {cmd_seq, MOSI_arr[7:0]};
48                 active莫斯I_word = MOSI_arr;
49                 old莫斯I_arr = active莫斯I_word;
50                 cmd_seq.old = cmd_seq;
51             end
52             else if(counter > 0) begin
53                 MOSI = active莫斯I_word[10];
54
55                 active莫斯I_word = active莫斯I_word << 1;
56             end
57             counter++;
58         end
59         else begin
60             counter = 0;
61         end
62     end
63     $display("莫斯I : %b",MOSI);
64     $display("莫斯I_word : %b",old莫斯I_arr);
65     $display("cmd_seq : %b",cmd_seq);
66 endfunction
67
68 endclass
69 endpackage

```

## Wrapper\_sequencer

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

## Wrapper\_RST\_sequence

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

## Wrapper\_read\_only\_sequence

```
1 package Wrapper_read_only_sequence_pkg;
2   import uvm_pkg::*;
3   import Wrapper_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class Wrapper_read_only_sequence extends uvm_sequence #(Wrapper_sequenceitem);
6     `uvm_object_utils(Wrapper_read_only_sequence)
7     Wrapper_sequenceitem item;
8     function new(string name = "Wrapper_read_only_sequence");
9       super.new(name);
10      endfunction
11
12      task body();
13        item = Wrapper_sequenceitem::type_id::create("item");
14
15        for (int i = 0; i < 500 ; i++) begin
16          start_item(item);
17          assert(item.randomize());
18          if(item.SS_n == 1) begin
19            if(i % 2 == 0) begin
20              item.active_MOSI_word = {3'b110, item.active_MOSI_word[7:0]};
21            end
22            else begin
23              item.active_MOSI_word = {3'b111, item.active_MOSI_word[7:0]};
24            end
25          end
26          finish_item(item);
27        end
28      endtask
29    endclass
30  endpackage
```

## Wrapper\_write\_only\_sequence

```
1 package Wrapper_write_only_sequence_pkg;
2   import uvm_pkg::*;
3   import Wrapper_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class Wrapper_write_only_sequence extends uvm_sequence #(Wrapper_sequenceitem);
6     `uvm_object_utils(Wrapper_write_only_sequence)
7     Wrapper_sequenceitem item;
8     function new(string name = "Wrapper_write_only_sequence");
9       super.new(name);
10      endfunction
11
12      task body();
13        item = Wrapper_sequenceitem::type_id::create("item");
14        for (int i = 0; i < 500 ; i++) begin
15          start_item(item);
16          assert(item.randomize());
17          if(item.SS_n == 1) begin
18            if(i % 2 == 0) begin
19              item.active_MOSI_word = {3'b000, item.active_MOSI_word[7:0]};
20            end
21            else begin
22              item.active_MOSI_word = {3'b001, item.active_MOSI_word[7:0]};
23            end
24          end
25          finish_item(item);
26        end
27      endtask
28    endclass
29  endpackage
```

## Wrapper\_read\_write\_sequence

```
1 package Wrapper_read_write_sequence_pkg;
2   import uvm_pkg::*;
3   import Wrapper_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class Wrapper_read_write_sequence extends uvm_sequence #(Wrapper_sequenceitem);
6     `uvm_object_utils(Wrapper_read_write_sequence)
7     Wrapper_sequenceitem item;
8     function new(string name = "Wrapper_read_write_sequence");
9       super.new(name);
10      endfunction
11
12      task body();
13        item = Wrapper_sequenceitem::type_id::create("item");
14        for (int i = 0; i < 10000 ; i++) begin
15          start_item(item);
16          assert(item.randomize());
17          finish_item(item);
18        end
19      endtask
20    endclass
21  endpackage
```

## Wrapper\_driver

```
1 package Wrapper_driver_pkg;
2     import uvm_pkg::*;
3     import SPI_RAM_config_pkg::*;
4     import Wrapper_sequenceitem_pkg::*;
5     `include "uvm_macros.svh"
6     class Wrapper_driver extends uvm_driver #(Wrapper_sequenceitem);
7         `uvm_component_utils(Wrapper_driver)
8         virtual Wrapper_inf Wrapper_driver_inf;
9         virtual Wrapper_gm_inf Wrapper_driver_inf_gm;
10        Wrapper_sequenceitem item;
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            `uvm_info("Wrapper_driver", "inside the Wrapper driver", UVM_LOW);
18            item = Wrapper_sequenceitem::type_id::create("item");
19            forever begin
20                seq_item_port.get_next_item(item);
21                Wrapper_driver_inf.MOSI = item.MOSI;
22                Wrapper_driver_inf.rst_n = item.rst_n;
23                Wrapper_driver_inf.SS_n = item.SS_n;
24
25                Wrapper_driver_inf_gm.MOSI = item.MOSI;
26                Wrapper_driver_inf_gm.rst_n = item.rst_n;
27                Wrapper_driver_inf_gm.SS_n = item.SS_n;
28
29                seq_item_port.item_done();
30                @(negedge Wrapper_driver_inf.clk);
31                item.MISO = Wrapper_driver_inf.MISO;
32                `uvm_info("Wrapper_driver", item.convert2string(), UVM_LOW);
33            end
34        endtask
35
36    endclass
37 endpackage
```

## Wrapper\_monitor

```
 1 package Wrapper_monitor_pkg;
 2     import uvm_pkg::*;
 3     import SPI_RAM_config_pkg::*;
 4     import Wrapper_sequenceitem_pkg::*;
 5     `include "uvm_macros.svh"
 6     class Wrapper_monitor extends uvm_monitor;
 7         `uvm_component_utils(Wrapper_monitor)
 8         virtual Wrapper_inf Wrapper_monitor_inf;
 9         virtual Wrapper_gm_inf Wrapper_monitor_inf_gm;
10         Wrapper_sequenceitem item;
11         Wrapper_sequenceitem item_gm;
12         uvm_analysis_port #(Wrapper_sequenceitem) mon_ap;
13         uvm_analysis_port #(Wrapper_sequenceitem) mon_ap_gm;
14
15         function new(string name = "Wrapper_monitor", 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             mon_ap = new("mon_ap", this);
22             mon_ap_gm = new("mon_ap_gm", this);
23         endfunction
24
25         task run_phase(uvm_phase phase);
26             super.run_phase(phase);
27             `uvm_info("Wrapper_monitor", "inside the Wrapper monitor", UVM_LOW);
28             forever begin
29                 @(negedge Wrapper_monitor_inf.clk);
30
31                 item_gm = Wrapper_sequenceitem::type_id::create("item_gm");
32                 item_gm.MOSI = Wrapper_monitor_inf_gm.MOSI;
33                 item_gm.rst_n = Wrapper_monitor_inf_gm.rst_n;
34                 item_gm.SS_n = Wrapper_monitor_inf_gm.SS_n;
35                 item_gm.MISO = Wrapper_monitor_inf_gm.MISO;
36
37                 item = Wrapper_sequenceitem::type_id::create("item");
38                 item.MOSI = Wrapper_monitor_inf.MOSI;
39                 item.rst_n = Wrapper_monitor_inf.rst_n;
40                 item.SS_n = Wrapper_monitor_inf.SS_n;
41                 item.MISO = Wrapper_monitor_inf.MISO;
42
43                 mon_ap_gm.write(item_gm);
44                 mon_ap.write(item);
45                 `uvm_info("Wrapper_monitor", item.convert2string(), UVM_LOW);
46             end
47         endtask
48
49     endclass
50
51 endpackage
```

## Wrapper\_scoreboard

```
● ○ ●
1 package Wrapper_scoreboard_pkg;
2   import uvm_pkg::*;
3   import Wrapper_sequenceitem_pkg::*;
4   `include "uvm_macros.svh"
5   class Wrapper_scoreboard extends uvm_scoreboard;
6     `uvm_component_utils(Wrapper_scoreboard)
7     uvm_analysis_export #(Wrapper_sequenceitem) sb_export;
8     uvm_tlm_analysis_fifo #(Wrapper_sequenceitem) sb_fifo;
9     uvm_analysis_export #(Wrapper_sequenceitem) sb_export_gm;
10    uvm_tlm_analysis_fifo #(Wrapper_sequenceitem) sb_fifo_gm;
11    Wrapper_sequenceitem item;
12    Wrapper_sequenceitem item_gm;
13
14    int correct_count = 0 , error_count = 0;
15    function new(string name = "Wrapper_scoreboard", 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      sb_fifo = new("sb_fifo", this);
22      sb_export = new("sb_export", this);
23      sb_fifo_gm = new("sb_fifo_gm", this);
24      sb_export_gm = new("sb_export_gm", this);
25      sb_export.connect(sb_fifo.analysis_export);
26      sb_export_gm.connect(sb_fifo_gm.analysis_export);
27    endfunction
28
29    task run_phase(uvm_phase phase);
30      super.run_phase(phase);
31      forever begin
32        sb_fifo.get(item);
33        sb_fifo_gm.get(item_gm);
34
35        if(item.MISO === item_gm.MISO) begin
36          correct_count = correct_count + 1;
37        end
38        else begin
39          `uvm_error("run_phase", $sformatf("MISO = %d", item.MISO));
40          `uvm_error("run_phase", $sformatf("MISO = %d", item_gm.MISO));
41          error_count = error_count + 1;
42        end
43      end
44    endtask
45
46    function void report_phase(uvm_phase phase);
47      super.report_phase(phase);
48      `uvm_info("report_phase", $sformatf("correct_count = %d", correct_count), UVM_LOW);
49      `uvm_info("report_phase", $sformatf("error_count = %d", error_count), UVM_LOW);
50    endfunction
51  endclass
52 endpackage
```

## Wrapper\_agent

```
 1 package Wrapper_agent_pkg;
 2   import uvm_pkg::*;
 3   `include "uvm_macros.svh"
 4   import SPI_RAM_config_pkg::*;
 5   import Wrapper_sequencer_pkg::*;
 6   import Wrapper_driver_pkg::*;
 7   import Wrapper_monitor_pkg::*;
 8   import Wrapper_sequenceitem_pkg::*;
 9   class Wrapper_agent extends uvm_agent;
10     `uvm_component_utils(Wrapper_agent)
11     Wrapper_driver driver;
12     Wrapper_monitor monitor;
13     Wrapper_sequencer sequencer;
14     SPI_RAM_config cfg;
15     uvm_analysis_port #(Wrapper_sequenceitem) agt_ap;
16     uvm_analysis_port #(Wrapper_sequenceitem) agt_ap_gm;
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#(SPI_RAM_config)::get(this, "", "wrapper_cfg", cfg)) begin
25         `uvm_fatal("Wrapper_agent", "virtual interface must be set for Wrapper_agent")
26       end
27       if(cfg.is_active == UVM_ACTIVE) begin
28         sequencer = Wrapper_sequencer::type_id::create("sequencer", this);
29         driver = Wrapper_driver::type_id::create("driver", this);
30       end
31       monitor = Wrapper_monitor::type_id::create("monitor", this);
32       agt_ap = new("agt_ap", this);
33       agt_ap_gm = new("agt_ap_gm", this);
34     endfunction
35
36     function void connect_phase(uvm_phase phase);
37       super.connect_phase(phase);
38       driver.Wrapper_driver_inf = cfg.wrapper_inf;
39       driver.Wrapper_driver_inf_gm = cfg.wrapper_inf_gm;
40       monitor.Wrapper_monitor_inf = cfg.wrapper_inf;
41       monitor.Wrapper_monitor_inf_gm = cfg.wrapper_inf_gm;
42       driver.seq_item_port.connect(sequencer.seq_item_export);
43       monitor.mon_ap.connect(agt_ap);
44       monitor.mon_ap_gm.connect(agt_ap_gm);
45     endfunction
46
47   endclass
48
49 endpackage
```

## Wrapper\_env

```
1 package Wrapper_env_pkg;
2     import uvm_pkg::*;
3     import Wrapper_agent_pkg::*;
4     import Wrapper_scoreboard_pkg::*;
5     `include "uvm_macros.svh"
6     class Wrapper_env extends uvm_env;
7         `uvm_component_utils(Wrapper_env)
8         Wrapper_scoreboard scoreboard;
9         Wrapper_agent agent;
10        function new(string name = "Wrapper_env", uvm_component parent = null);
11            super.new(name, parent);
12        endfunction
13
14        function void build_phase(uvm_phase phase);
15            super.build_phase(phase);
16            agent = Wrapper_agent::type_id::create("agent", this);
17            scoreboard = Wrapper_scoreboard::type_id::create("scoreboard", this);
18        endfunction
19
20        function void connect_phase(uvm_phase phase);
21            super.connect_phase(phase);
22            agent.agt_ap.connect(scoreboard.sb_export);
23            agent.agt_ap_gm.connect(scoreboard.sb_export_gm);
24        endfunction
25    endclass
26 endpackage
```

## Wrapper\_test

```
 1 package Wrapper_test_pkg;
 2   import uvm_pkg::*;
 3   import SPI_slave_env_pkg::*;
 4   import SPI_RAM_config_pkg::*;
 5   import Wrapper_write_only_sequence_pkg::*;
 6   import Wrapper_read_only_sequence_pkg::*;
 7   import Wrapper_rst_sequence_pkg::*;
 8   import Wrapper_read_write_sequence_pkg::*;
 9   import RAM_env::*;
10   import Wrapper_env_pkg::*;
11   `include "uvm_macros.svh"
12   class Wrapper_test extends uvm_test;
13     `uvm_component_utils(Wrapper_test)
14
15     SPI_slave_env spi_env;
16     RAM_env env;
17     Wrapper_env wrapper_env;
18     SPI_RAM_config spi_cfg;
19     SPI_RAM_config ram_cfg;
20     SPI_RAM_config wrapper_cfg;
21     Wrapper_write_only_sequence write_only_seq;
22     Wrapper_read_only_sequence read_only_seq;
23     Wrapper_rst_sequence rst_seq;
24     Wrapper_read_write_sequence read_write_seq;
25
26     function new(string name = "Wrapper_test", uvm_component parent = null);
27       super.new(name, parent);
28     endfunction
29
30     function void build_phase(uvm_phase phase);
31       super.build_phase(phase);
32       spi_env = SPI_slave_env::type_id::create("spi_env", this);
33       env = RAM_env::type_id::create("env", this);
34       wrapper_env = Wrapper_env::type_id::create("wrapper_env", this);
35       spi_cfg = SPI_RAM_config::type_id::create("spi_cfg");
36       ram_cfg = SPI_RAM_config::type_id::create("ram_cfg");
37       wrapper_cfg = SPI_RAM_config::type_id::create("wrapper_cfg");
38       // create the rst and main sequence
39       rst_seq = Wrapper_rst_sequence::type_id::create("rst_seq");
40       write_only_seq = Wrapper_write_only_sequence::type_id::create("write_only_seq");
41       read_only_seq = Wrapper_read_only_sequence::type_id::create("read_only_seq");
42       read_write_seq = Wrapper_read_write_sequence::type_id::create("read_write_seq");
```

```
1      if (!uvm_config_db#(virtual SPI_slave_inf)::get(this, "", "vif", spi_cfg.SPI_vif)) begin
2          `uvm_fatal("Wrapper_test", "virtual interface must be set for Wrapper_test")
3      end
4      if (!uvm_config_db#(virtual SPI_slave_gm_inf)::get(this, "", "vif_gm", spi_cfg.SPI_vif_gm)) begin
5          `uvm_fatal("Wrapper_test", "virtual interface must be set for Wrapper_test")
6      end
7
8      if (!uvm_config_db#(virtual Wrapper_inf)::get(this, "", "wrap_vif", wrapper_cfg.wrapper_inf)) begin
9          `uvm_fatal("Wrapper_test", "virtual interface must be set for Wrapper_test")
10     end
11     if (!uvm_config_db#(virtual Wrapper_gm_inf)::get(this, "", "wrap_ref", wrapper_cfg.wrapper_inf_gm)) begin
12         `uvm_fatal("Wrapper_test", "virtual interface must be set for Wrapper_test")
13     end
14
15     if(!uvm_config_db#(virtual RAM_if)::get(this, "", "vif_DUT", ram_cfg.vif)) begin
16         `uvm_fatal("NOVIF","Virtual interface must be set for: via configuration database");
17     end
18     if(!uvm_config_db#(virtual RAM_if_ref)::get(this, "", "vif_REF", ram_cfg.vif_ref)) begin
19         `uvm_fatal("NOVIF","Virtual interface reference must be set for: via configuration database");
20     end
21
22     spi_cfg.is_active = UVM_PASSIVE;
23     ram_cfg.is_active = UVM_PASSIVE;
24     wrapper_cfg.is_active = UVM_ACTIVE;
25
26     uvm_config_db#(SPI_RAM_config)::set(this, "*", "SPI_slave_config_test", spi_cfg);
27     uvm_config_db#(SPI_RAM_config)::set(this, "*", "cfg", ram_cfg);
28     uvm_config_db#(SPI_RAM_config)::set(this, "*", "wrapper_cfg", wrapper_cfg);
29 endfunction
30
31 task run_phase(uvm_phase phase);
32     super.run_phase(phase);
33     phase.raise_objection(this);
34     `uvm_info("Wrapper_test", "rst_sequence start", UVM_LOW);
35     rst_seq.start(wrapper_env.agent.sequencer);
36     `uvm_info("Wrapper_test", "rst_sequence end", UVM_LOW);
37     `uvm_info("Wrapper_test", "write_only_sequence start", UVM_LOW);
38     write_only_seq.start(wrapper_env.agent.sequencer);
39     `uvm_info("Wrapper_test", "write_only_sequence end", UVM_LOW);
40     `uvm_info("Wrapper_test", "read_only_sequence start", UVM_LOW);
41     read_only_seq.start(wrapper_env.agent.sequencer);
42     `uvm_info("Wrapper_test", "read_only_sequence end", UVM_LOW);
43     `uvm_info("Wrapper_test", "read_write_sequence start", UVM_LOW);
44     read_write_seq.start(wrapper_env.agent.sequencer);
45     `uvm_info("Wrapper_test", "read_write_sequence end", UVM_LOW);
46     phase.drop_objection(this);
47 endtask
48 endclass
49 endpackage
```

## Previous File changes to accommodate for the full environment

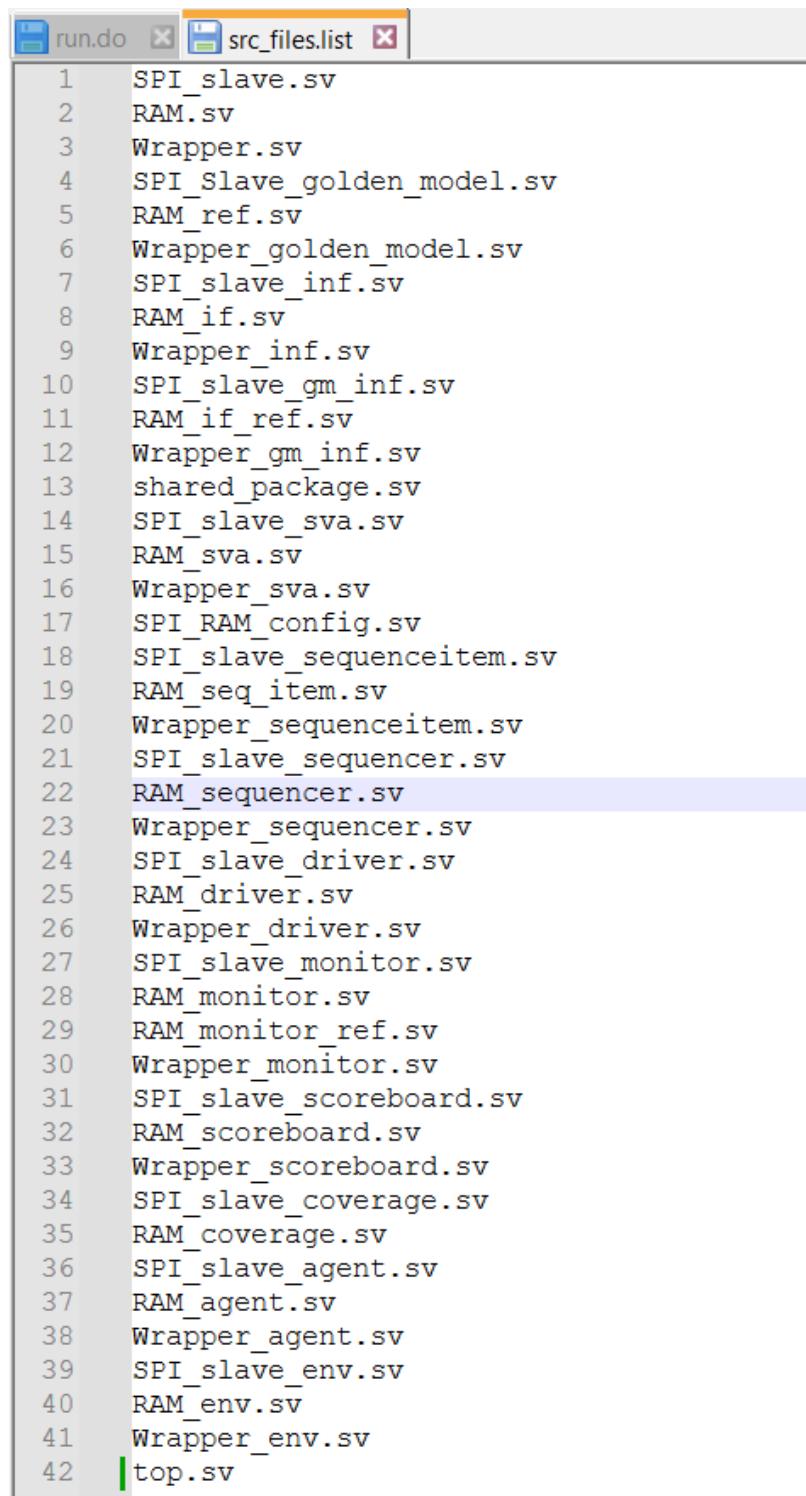
```
1  function void build_phase(uvm_phase phase);
2      super.build_phase(phase);
3      if(!uvm_config_db#(SPI_RAM_config)::get(this, "", "SPI_slave_config_test", cfg)) begin
4          `uvm_fatal("SPI_slave_agent", "virtual interface must be set for SPI_slave_agent")
5      end
6      if(cfg.is_active == UVM_ACTIVE) begin
7          sequencer = SPI_slave_sequencer::type_id::create("sequencer", this);
8          driver = SPI_slave_driver::type_id::create("driver", this);
9      end
10     monitor = SPI_slave_monitor::type_id::create("monitor", this);
11     agt_ap = new("agt_ap", this);
12     agt_ap_gm = new("agt_ap_gm", this);
13 endfunction
14
15 function void connect_phase(uvm_phase phase);
16     super.connect_phase(phase);
17     monitor.SPI_slave_monitor_inf = cfg.SPI_vif;
18     monitor.SPI_slave_monitor_inf_gm = cfg.SPI_vif_gm;
19     if(cfg.is_active == UVM_ACTIVE) begin
20         driver.seq_item_port.connect(sequencer.seq_item_export);
21         driver.SPI_slave_driver_inf = cfg.SPI_vif;
22         driver.SPI_slave_driver_inf_gm = cfg.SPI_vif_gm;
23     end
24     monitor.mon_ap.connect(agt_ap);
25     monitor.mon_ap_gm.connect(agt_ap_gm);
26 endfunction
```

A section that shows the changes performed on the SPI\_SLAVE agent to accommodate for the passive creation of these environments.

```
1 function void build_phase(uvm_phase phase);
2     super.build_phase(phase);
3     if(!uvm_config_db#(SPI_RAM_config)::get(this, "", "cfg", cfg)) begin
4         `uvm_fatal("NOVIF","Virtual interface must be set for: via configuration database");
5     end
6     if(cfg.is_active == UVM_ACTIVE) begin
7         sequencer = RAM_sequencer::type_id::create("sequencer", this);
8         driver = RAM_driver::type_id::create("driver", this);
9     end
10    monitor = RAM_monitor::type_id::create("monitor", this);
11    agent_ap = new("agent_ap", this);
12 endfunction
13
14 function void connect_phase(uvm_phase phase);
15     super.connect_phase(phase);
16     monitor.vif = cfg.vif;
17     monitor.mon_ap.connect(agent_ap);
18     if(cfg.is_active == UVM_ACTIVE) begin
19         driver.vif = cfg.vif;
20         driver.seq_item_port.connect(sequencer.seq_item_export);
21     end
22 endfunction
23 endclass
```

A section that shows the changes performed on the RAM agent to accommodate for the passive creation of these environments.

## Source Files List



The screenshot shows a software interface with a window titled "src\_files.list". The window contains a list of 42 source files, each preceded by a line number. The files are listed as follows:

```
1 SPI_slave.sv
2 RAM.sv
3 Wrapper.sv
4 SPI_Slave_golden_model.sv
5 RAM_ref.sv
6 Wrapper_golden_model.sv
7 SPI_slave_inf.sv
8 RAM_if.sv
9 Wrapper_inf.sv
10 SPI_slave_gm_inf.sv
11 RAM_if_ref.sv
12 Wrapper_gm_inf.sv
13 shared_package.sv
14 SPI_slave_sva.sv
15 RAM_sva.sv
16 Wrapper_sva.sv
17 SPI_RAM_config.sv
18 SPI_slave_sequenceitem.sv
19 RAM_seq_item.sv
20 Wrapper_sequenceitem.sv
21 SPI_slave_sequencer.sv
22 RAM_sequencer.sv
23 Wrapper_sequencer.sv
24 SPI_slave_driver.sv
25 RAM_driver.sv
26 Wrapper_driver.sv
27 SPI_slave_monitor.sv
28 RAM_monitor.sv
29 RAM_monitor_ref.sv
30 Wrapper_monitor.sv
31 SPI_slave_scoreboard.sv
32 RAM_scoreboard.sv
33 Wrapper_scoreboard.sv
34 SPI_slave_coverage.sv
35 RAM_coverage.sv
36 SPI_slave_agent.sv
37 RAM_agent.sv
38 Wrapper_agent.sv
39 SPI_slave_env.sv
40 RAM_env.sv
41 Wrapper_env.sv
42 top.sv
```

## DO File

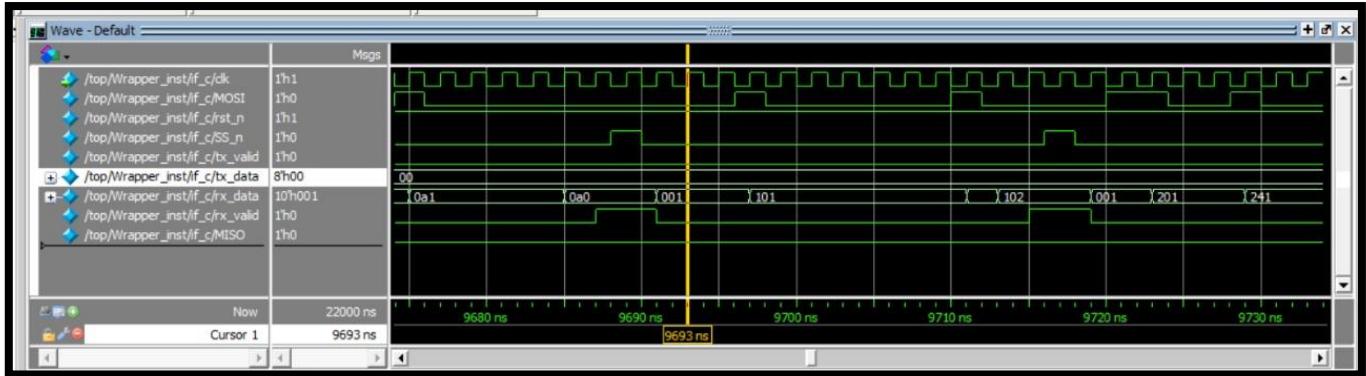
```
run.do      src_files.list
1  vlib work
2  vlog -f src_files.list +define+SIM -cover bcesf +cover +acc -covercells
3  vsim -coverage -voptargs=+acc work.top -classdebug -uvmcontrol=all -sv_seed random
4  coverage save top.ucdb -onexit
5  #add wave -position insertpoint sim:/top/if_wrapper/*
6  add wave -position insertpoint sim:/top/Wrapper_inst/if_c/*
7  run 0
8  run -all
```

## QuestaSim Snippets

### Showcasing the sequence “000”



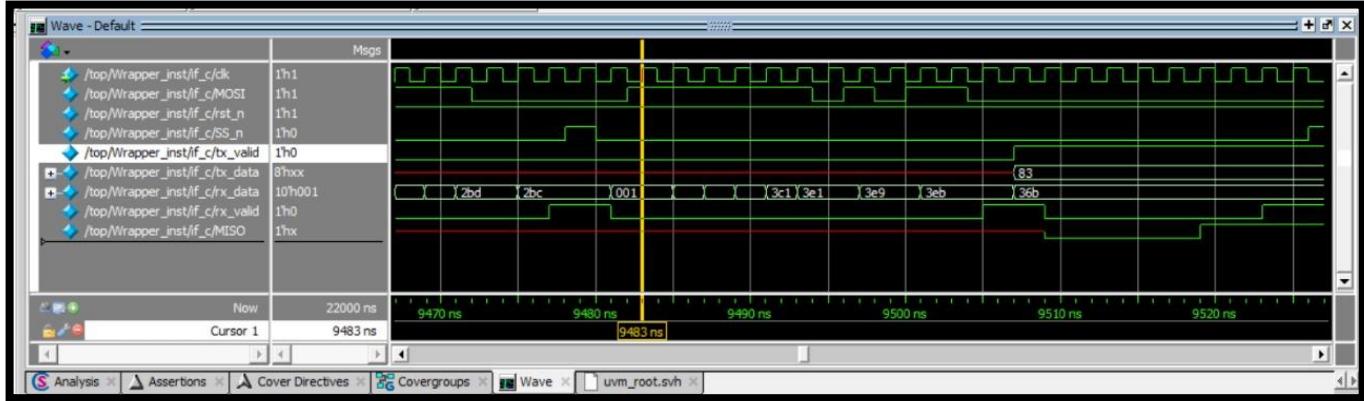
## Showcasing the sequence “001”



## Showcasing the sequence “110”

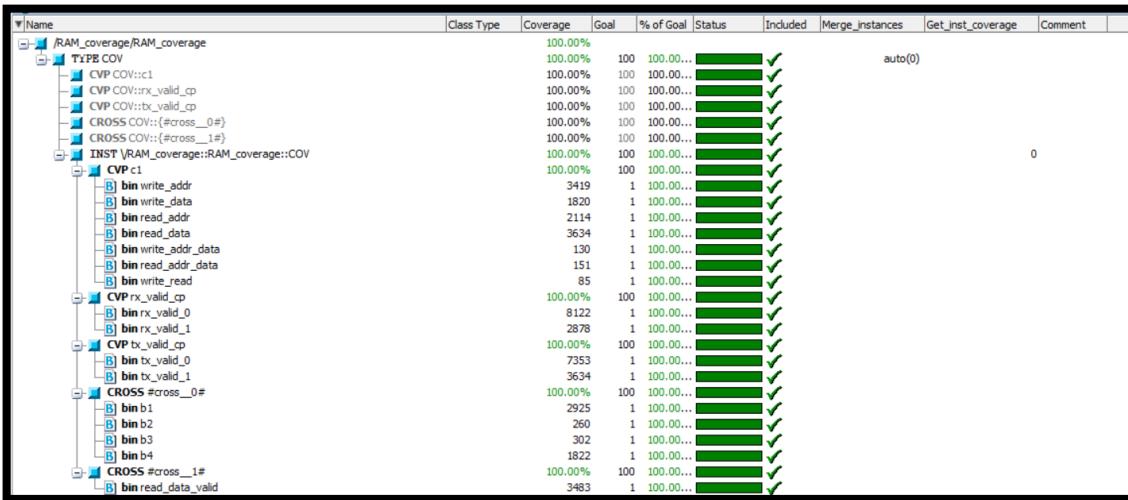
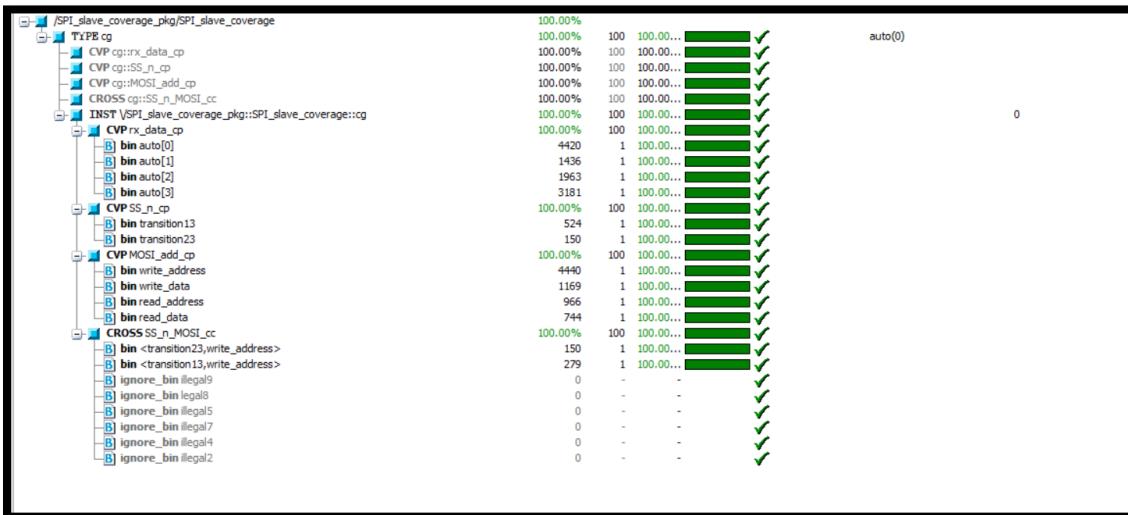


# Showcasing the sequence “111”



```
# UVM_INFO SPI_slave_monitor.sv(54) @ 22000: uvm_test_top.spi_env.agent.monitor [SPI_slave_monitor] SPI_slave_sequenceitem: : MOSI=1,
#           rst_n=1, SS_n=0, tx_valid=1, tx_data=11010110, rx_data=0101010001,
#           rx_valid=0, MISO=0
# UVM_INFO Wrapper_driver.sv(32) @ 22000: uvm_test_top.wrapper_env.agent.driver [Wrapper_driver] Wrapper_sequenceitem: : MOSI=1, rst_n=1, SS_n=0
# UVM_INFO Wrapper_monitor.sv(46) @ 22000: uvm_test_top.wrapper_env.agent.monitor [Wrapper_monitor] Wrapper_sequenceitem: : MOSI=1, rst_n=1, SS_n=0
# MOSI : 1
# MOSI_word : 00010110000
# cmd_seq : 000
# UVM_INFO D:/Summer2025/Digital_Verification/Saeed Nabawy_Project2/Wrapper/Wrapper_test.sv(87) @ 22000: uvm_test_top [Wrapper_test] read_write_sequence end
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 22000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO RAM_scoreboard.sv(51) @ 22000: uvm_test_top.env.sb [report_phase] correct_count = 11000
# UVM_INFO RAM_scoreboard.sv(52) @ 22000: uvm_test_top.env.sb [report_phase] error_count = 0
# UVM_INFO SPI_slave_scoreboard.sv(50) @ 22000: uvm_test_top.spi_env.scoreboard [report_phase] correct_count = 11000
# UVM_INFO SPI_slave_scoreboard.sv(51) @ 22000: uvm_test_top.spi_env.scoreboard [report_phase] error_count = 0
# UVM_INFO Wrapper_scoreboard.sv(50) @ 22000: uvm_test_top.wrapper_env.scoreboard [report_phase] correct_count = 11000
# UVM_INFO Wrapper_scoreboard.sv(51) @ 22000: uvm_test_top.wrapper_env.scoreboard [report_phase] error_count = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :33021
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questasim] 2
# [RNTST] 1
# [SPI_slave_monitor] 11001
# [TEST_DONE] 1
# [Wrapper_driver] 11001
# [Wrapper_monitor] 11001
# [Wrapper_test] 8
# [report_phase] 6
# ** Note: $finish : C:/questasim64_2021.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 22 us Iteration: 65 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
```

# Functional Coverage



## COVERGROUP COVERAGE:

Covergroup	Metric	Goal	Bins	Status
TYPE /RAM_coverage/RAM_coverage/COV	100.00%	100	-	Covered
covered/total bins:	16	16	-	
missing/total bins:	0	16	-	
% Hit:	100.00%	100	-	
Coverpoint c1	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
Coverpoint rx_valid_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
Coverpoint tx_valid_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% 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	-	
Cross #cross_1#	100.00%	100	-	Covered
covered/total bins:	1	1	-	
missing/total bins:	0	1	-	
% Hit:	100.00%	100	-	
Covergroup instance \RAM_coverage::RAM_coverage::COV	100.00%	100	-	Covered
covered/total bins:	16	16	-	
missing/total bins:	0	16	-	
% Hit:	100.00%	100	-	
Coverpoint c1	100.00%	100	-	Covered
covered/total bins:	7	7	-	
missing/total bins:	0	7	-	
% Hit:	100.00%	100	-	
bin write_addr	4590	1	-	Covered
bin write_data	1934	1	-	Covered
bin read_addr	1603	1	-	Covered
bin read_data	2860	1	-	Covered
bin write_addr_data	129	1	-	Covered
bin read_addr_data	96	1	-	Covered
bin write_read	57	1	-	Covered
Coverpoint rx_valid_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin rx_valid_0	9200	1	-	Covered
bin rx_valid_1	1800	1	-	Covered
Coverpoint tx_valid_cp	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin tx_valid_0	8155	1	-	Covered
bin tx_valid_1	2832	1	-	Covered
Cross #cross_0#	100.00%	100	-	Covered
covered/total bins:	4	4	-	
missing/total bins:	0	4	-	
% Hit:	100.00%	100	-	

covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Auto, Default and User Defined Bins:			
bin b1	3960	1	- Covered
bin b2	264	1	- Covered
bin b3	214	1	- Covered
bin b4	692	1	- Covered
Cross #cross_1#	100.00%	100	- Covered
covered/total bins:	1	1	-
missing/total bins:	0	1	-
% Hit:	100.00%	100	-
Auto, Default and User Defined Bins:			
bin read_data_valid	2725	1	- Covered
TYPE /SPI_slave_coverage_pkg/SPI_slave_coverage/cg	100.00%	100	- Covered
covered/total bins:	12	12	-
missing/total bins:	0	12	-
% Hit:	100.00%	100	-
Coverpoint rx_data_cp	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Coverpoint SS_n_cp	100.00%	100	- Covered
covered/total bins:	2	2	-
missing/total bins:	0	2	-
% Hit:	100.00%	100	-
Coverpoint MOSI_add_cp	100.00%	100	- Covered
covered/total bins:	4	4	-
missing/total bins:	0	4	-
% Hit:	100.00%	100	-
Cross SS_n_MOSI_cc	100.00%	100	- Covered
covered/total bins:	2	2	-
missing/total bins:	0	2	-
% Hit:	100.00%	100	-

```

-----
Covergroup instance \SPI_slave_coverage_pkg::SPI_slave_coverage::cg
-----  

    covered/total bins:          100.00%   100      -  Covered  

    missing/total bins:          12        12      -  

    % Hit:                      100.00%   100      -  

Coverpoint rx_data_cp
    covered/total bins:          100.00%   100      -  Covered
    missing/total bins:          4         4      -  

    % Hit:                      100.00%   100      -  

        bin auto[0]              5581      1      -  Covered
        bin auto[1]              1517      1      -  Covered
        bin auto[2]              1475      1      -  Covered
        bin auto[3]              2427      1      -  Covered
Coverpoint SS_n_cp
    covered/total bins:          100.00%   100      -  Covered
    missing/total bins:          2         2      -  

    % Hit:                      100.00%   100      -  

        bin transition13         550       1      -  Covered
        bin transition23         110       1      -  Covered
Coverpoint MOSI_addr_cp
    covered/total bins:          100.00%   100      -  Covered
    missing/total bins:          4         4      -  

    % Hit:                      100.00%   100      -  

        bin write_address        4345      1      -  Covered
        bin write_data           1188      1      -  Covered
        bin read_address         951       1      -  Covered
        bin read_data            815       1      -  Covered
Cross SS_n_MOSI_cc
    covered/total bins:          100.00%   100      -  Covered
    missing/total bins:          2         2      -  

    % Hit:                      100.00%   100      -  

Auto, Default and User Defined Bins:
    bin <transition23,write_address> 110       1      -  Covered
    bin <transition13,write_address> 270       1      -  Covered
Illegal and Ignore Bins:
    ignore_bin illegal9          0        -  ZERO
    ignore_bin legal8            0        -  ZERO
    ignore_bin illegal5          0        -  ZERO
    ignore_bin illegal7          0        -  ZERO
    ignore_bin illegal14          0        -  ZERO
    ignore_bin illegal2           0        -  ZERO

```

TOTAL COVERGROUP COVERAGE: 100.00% COVERGROUP TYPES: 2

## Assertion Summary

Feature	Assertion
<b>Whenever reset is asserted, the outputs (MISO, rx_valid, and rx_data) are all inactive.</b>	<pre>@(posedge if_wrapper.clk)     (~if_wrapper.rst_n  =&gt; (if_wrapper.MISO == 'b0))</pre>
<b>The MISO remains with a stable value as long as it is not a read data operation</b>	<pre>@(posedge if_wrapper.clk)     disable iff (!if_wrapper.rst_n) // disable when reset active or slave not selected     (if_wrapper.SS_n ## 1 !if_wrapper.SS_n ## 1     (if_wrapper.MOSI ## 1 if_wrapper.MOSI ## 1     if_wrapper.MOSI))  =&gt; \$stable(if_wrapper.MISO)     throughout (!if_wrapper.SS_n);</pre>

# Assertion Coverage

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
lvm_pk:uvvm_req_map:do_write#pubk:#215181159#1731/lmmed_1735	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert(@cast(seq_o))	x
lvm_pk:uvvm_req_map:do_read#pubk:#215181159#1731/lmmed_1775	Immediate	SVA	on	0	0	-	-	-	-	-	off	assert(@cast(seq_o))	x
lvm_wrapper_read_write_sequence_pkp:Wrapper_read_only_sequence::body#anonblk#...	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert(randomize(...))	✓
lvm_wrapper_read_only_sequence_pkp:Wrapper_read_only_sequence::body#anonblk#...	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert(randomize(...))	✓
lvm_wrapper_write_only_sequence_pkp:Wrapper_write_only_sequence::body#anonblk#...	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert(randomize(...))	✓
lvm_wrapper_write_only_sequence_pkp:Wrapper_write_only_sequence::body#anonblk#...	Immediate	SVA	on	0	1	-	-	-	-	-	off	assert(randomize(...))	✓
lvm_wrapper_instSLAVE_instance/assert_p_state_operation_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_instSLAVE_instance/assert_p_state_operation_2	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_instSLAVE_instance/assert_p_state_operation_3	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_instSLAVE_instance/assert_p_state_operation_4	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_instSLAVE_instance/assert_p_state_operation_5	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_assert_p_reset	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_wrapper,dk) ...	✓
lvm_wrapper_assert_p_MISO_stable_when_not_read	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge DUT_if,dk) {~o...}	✓
lvm_wrapper_RAM/assert_l_rst_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_wrapper,dk) ...	✓
lvm_wrapper_RAM/assert_tx_val_low_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge DUT_if,dk) dis...	✓
lvm_wrapper_RAM/assert_tx_val_high_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge DUT_if,dk) dis...	✓
lvm_wrapper_RAM/assert_write_addr_data_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge DUT_if,dk) dis...	✓
lvm_wrapper_RAM/assert_tx_val_low_data_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_wrapper,dk) ...	✓
lvm_wrapper_SPL_slave/assert_p_rx	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_SPL_slave/assert_p_rx_valid_SS_n_operation_write_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_SPL_slave/assert_p_rx_valid_SS_n_operation_write_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_SPL_slave/assert_p_rx_valid_SS_n_operation_read_add	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓
lvm_wrapper_SPL_slave/assert_p_rx_valid_SS_n_operation_read_data	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@posedge if_c,dk) disable...	✓

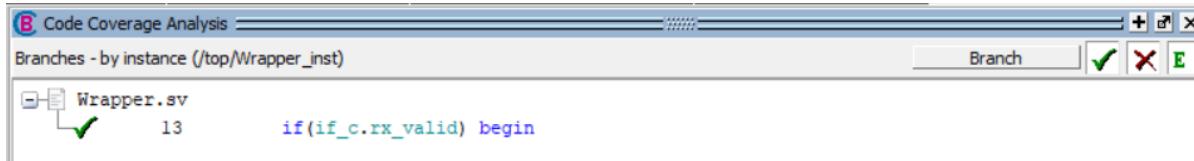
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmpt %	Cmpt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/lvWrapper_inst/SLAVE_instance/cover_p_state_operation_5...	SVA	✓	Off	2453	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lvWrapper_inst/SLAVE_instance/cover_p_state_operation_4...	SVA	✓	Off	2014	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lvWrapper_inst/SLAVE_instance/cover_p_state_operation_3...	SVA	✓	Off	4842	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lvWrapper_inst/SLAVE_instance/cover_p_state_operation_2...	SVA	✓	Off	721	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lvWrapper_inst/SLAVE_instance/cover_p_state_operation_1...	SVA	✓	Off	730	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/assert_p_reset	SVA	✓	Off	122	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/cover_l_rst	SVA	✓	Off	120	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/RAM/cover_p_rx	SVA	✓	Off	1680	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/RAM/cover_p_rx_val_low_check	SVA	✓	Off	3105	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/RAM/cover_p_rx_val_high_check	SVA	✓	Off	84	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/RAM/cover_tx_val_low_check	SVA	✓	Off	7655	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/RAM/cover_tx_val_high_check	SVA	✓	Off	120	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/SPL_slave/cover_p_rx_val_SS_n_operation_read_d...	SVA	✓	Off	97	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/SPL_slave/cover_p_rx_val_SS_n_operation_read_a...	SVA	✓	Off	74	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/SPL_slave/cover_p_rx_val_SS_n_operation_write_d...	SVA	✓	Off	91	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/SPL_slave/cover_p_rx_val_SS_n_operation_write_a...	SVA	✓	Off	183	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0
/top/lv_wrapper/SPL_slave/cover_p_rx_val_SS_n_operation_read_ss...	SVA	✓	Off	120	1	Unl...	1	100%	██████████	✓	0	0	0 ns	0

ASSERTION RESULTS:				
Name	File(Line)	Failure Count	Pass Count	
/top/Wrapper_inst/SLAVE_instance/assert__p_state_operation_5	SPI_slave.sv(172)	0	1	
/top/Wrapper_inst/SLAVE_instance/assert__p_state_operation_4	SPI_slave.sv(165)	0	1	
/top/Wrapper_inst/SLAVE_instance/assert__p_state_operation_3	SPI_slave.sv(158)	0	1	
/top/Wrapper_inst/SLAVE_instance/assert__p_state_operation_2	SPI_slave.sv(150)	0	1	
/top/Wrapper_inst/SLAVE_instance/assert__p_state_operation_1	SPI_slave.sv(141)	0	1	
/top/sva_Wrapper/assert__MISO_stable_when_not_read	Wrapper_sva.sv(16)	0	1	
/top/sva_Wrapper/assert__p_reset	Wrapper_sva.sv(7)	0	1	
/top/sva_RAM/assert__read_addr_data_check	RAM_sva.sv(35)	0	1	
/top/sva_RAM/assert__write_addr_data_check	RAM_sva.sv(28)	0	1	
/top/sva_RAM/assert__tx_valid_high_check	RAM_sva.sv(21)	0	1	
/top/sva_RAM/assert__tx_valid_low_check	RAM_sva.sv(14)	0	1	
/top/sva_RAM/assert__rst_check	RAM_sva.sv(7)	0	1	
/top/sva_SPI_slave/assert__p_rx_valid_SS_n_operation_read_data	SPI_slave_sva.sv(42)	0	1	
/top/sva_SPI_slave/assert__p_rx_valid_SS_n_operation_read_add	SPI_slave_sva.sv(35)	0	1	
/top/sva_SPI_slave/assert__p_rx_valid_SS_n_operation_write_data	SPI_slave_sva.sv(28)	0	1	
/top/sva_SPI_slave/assert__p_rx_valid_SS_n_operation_write_add	SPI_slave_sva.sv(21)	0	1	
/top/sva_SPI_slave/assert__p_reset	SPI_slave_sva.sv(14)	0	1	
/Wrapper_read_write_sequence_pkg/Wrapper_read_write_sequence/body/#anonblk#235939735#14#/#ublk#235939735#14/immed_16	D:/Summer2025/Digital_Verification/Saeed Nabawy_Project2/Wrapper/read_write_sequence.sv(16)	0	1	
/Wrapper_read_only_sequence_pkg/Wrapper_read_only_sequence/body/#anonblk#757463#15#/#ublk#757463#15/immed_17	D:/Summer2025/Digital_Verification/Saeed Nabawy_Project2/Wrapper/read_only_sequence.sv(17)	0	1	
/Wrapper_write_only_sequence_pkg/Wrapper_write_only_sequence/body/#anonblk#63355255#14#/#ublk#63355255#14/immed_16	D:/Summer2025/Digital_Verification/Saeed Nabawy_Project2/Wrapper/write_only_sequence.sv(16)	0	1	

DIRECTIVE COVERAGE:					
Name	Design Unit	Design UnitType	Lang File(Line)	Hits	Status
/top/Wrapper_inst/SLAVE_instance/cover__p_state_operation_5	SPI_slave	Verilog	SVA SPI_slave.sv(173)	2261	Covered
/top/Wrapper_inst/SLAVE_instance/cover__p_state_operation_4	SPI_slave	Verilog	SVA SPI_slave.sv(166)	1692	Covered
/top/Wrapper_inst/SLAVE_instance/cover__p_state_operation_3	SPI_slave	Verilog	SVA SPI_slave.sv(159)	5415	Covered
/top/Wrapper_inst/SLAVE_instance/cover__p_state_operation_2	SPI_slave	Verilog	SVA SPI_slave.sv(151)	725	Covered
/top/Wrapper_inst/SLAVE_instance/cover__p_state_operation_1	SPI_slave	Verilog	SVA SPI_slave.sv(142)	728	Covered
/top/sva_Wrapper/cover__MISO_stable_when_not_read	Wrapper_sva	Verilog	SVA Wrapper_sva.sv(17)	108	Covered
/top/sva_Wrapper/cover__p_reset	Wrapper_sva	Verilog	SVA Wrapper_sva.sv(8)	90	Covered
/top/sva_RAM/cover__read_addr_data_check	RAM_sva	Verilog	SVA RAM_sva.sv(36)	1421	Covered
/top/sva_RAM/cover__write_addr_data_check	RAM_sva	Verilog	SVA RAM_sva.sv(29)	3691	Covered
/top/sva_RAM/cover__tx_valid_high_check	RAM_sva	Verilog	SVA RAM_sva.sv(22)	74	Covered
/top/sva_RAM/cover__tx_valid_low_check	RAM_sva	Verilog	SVA RAM_sva.sv(15)	8006	Covered
/top/sva_RAM/cover__rst_check	RAM_sva	Verilog	SVA RAM_sva.sv(8)	90	Covered
/top/sva_SPI_slave/cover__p_rx_valid_SS_n_operation_read_data	SPI_slave_sva	Verilog	SVA SPI_slave_sva.sv(43)	94	Covered
/top/sva_SPI_slave/cover__p_rx_valid_SS_n_operation_read_add	SPI_slave_sva	Verilog	SVA SPI_slave_sva.sv(36)	81	Covered
/top/sva_SPI_slave/cover__p_rx_valid_SS_n_operation_write_data	SPI_slave_sva	Verilog	SVA SPI_slave_sva.sv(29)	113	Covered
/top/sva_SPI_slave/cover__p_rx_valid_SS_n_operation_write_add	SPI_slave_sva	Verilog	SVA SPI_slave_sva.sv(22)	236	Covered
/top/sva_SPI_slave/cover__p_reset	SPI_slave_sva	Verilog	SVA SPI_slave_sva.sv(15)	90	Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 17

# Code Coverage



```
=====
== Instance: /top/if_c
== Design Unit: work.SPI_slave_inf
=====

Toggle Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      -----
  Toggles          50      50        0  100.00%

=====Toggle Details=====

Toggle Coverage for instance /top/if_c --
                                         Node      1H->0L      0L->1H  "Coverage"
                                         -----
                                         MISO      1          1  100.00
                                         MOSI      1          1  100.00
                                         SS_n      1          1  100.00
                                         clk       1          1  100.00
                                         rst_n     1          1  100.00
                                         rx_data[9-0] 1          1  100.00
                                         rx_valid   1          1  100.00
                                         tx_data[7-0] 1          1  100.00
                                         tx_valid   1          1  100.00

Total Node Count      =      25
Toggled Node Count   =      25
Untoggled Node Count =      0

Toggle Coverage      =  100.00% (50 of 50 bins)
```

```
=====
== Instance: /top/if_ram
== Design Unit: work.RAM_if
=====
Toggle Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      ----   -----  -----
  Toggles          44      44        0  100.00%
=====
=====Toggle Details=====
Toggle Coverage for instance /top/if_ram --

```

Node	1H->0L	0L->1H	"Coverage"
clk	1	1	100.00
din[9-0]	1	1	100.00
dout[7-0]	1	1	100.00
rst_n	1	1	100.00
rx_valid	1	1	100.00
tx_valid	1	1	100.00

```
Total Node Count      =      22
Toggled Node Count   =      22
Untoggled Node Count =      0
Toggle Coverage      = 100.00% (44 of 44 bins)
```

```
=====
== Instance: /top/if_wrapper
== Design Unit: work.Wrapper_inf
=====
Toggle Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      ----   -----  -----
  Toggles          10      10        0  100.00%
=====
=====Toggle Details=====
Toggle Coverage for instance /top/if_wrapper --

```

Node	1H->0L	0L->1H	"Coverage"
MISO	1	1	100.00
MOSI	1	1	100.00
SS_n	1	1	100.00
clk	1	1	100.00
rst_n	1	1	100.00

```
Total Node Count      =      5
Toggled Node Count   =      5
Untoggled Node Count =      0
Toggle Coverage      = 100.00% (10 of 10 bins)
```

```

=====
== Instance: /top/Wrapper_inst/RAM_instance
== Design Unit: work.RAM
=====
Branch Coverage:
  Enabled Coverage      Bins    Hits    Misses   Coverage
  -----      ----  -----  -----  -----
  Branches          8       8       0     100%
=====
=====Branch Details=====
Branch Coverage for instance /top/Wrapper_inst/RAM_instance
  Line      Item            Count      Source
  ---      ---            ----  -----
  File RAM.sv
  -----IF Branch-----
  7           1           11000  Count coming in to IF
  7           1             90      if (~DUT_if.rst_n) begin
  13          1           10910  else begin
Branch totals: 2 hits of 2 branches = 100.00%
  -----IF Branch-----
  14          1           10910  Count coming in to IF
  14          1             1785  if (DUT_if.rx_valid) begin
  14          1             9125  All False Count
Branch totals: 2 hits of 2 branches = 100.00%
  -----CASE Branch-----
  15          1           1785  Count coming in to CASE
  16          1             623  2'b00 : Wr_Addr <= DUT_if.din[7:0];
  17          1             261  2'b01 : MEM[Wr_Addr] <= DUT_if.din[7:0];
  18          1             214  2'b10 : Rd_Addr <= DUT_if.din[7:0];
  19          1             687  2'b11 : DUT_if.dout <= MEM[Rd_Addr];
Branch totals: 4 hits of 4 branches = 100.00%

```

```

FSM Coverage:
  Enabled Coverage      Bins    Hits    Misses   Coverage
  -----      ----  -----  -----  -----
  FSM States          5       5       0     100.00%
  FSM Transitions     8       8       0     100.00%
=====
=====FSM Details=====
FSM Coverage for instance /top/Wrapper_inst/SLAVE_instance --
  FSM_ID: cs
    Current State Object : cs
    -----
    State Value MapInfo :
    -----
    Line      State Name        Value
    ---      -----
    28       IDLE             0
    34       CHK_CMD          1
    60       READ_DATA        4
    54       READ_ADD          3
    48       WRITE             2
    Covered States :
    -----
    Line      State        Hit_count
    ---      -----
    28       IDLE           743
    34       CHK_CMD         737
    60       READ_DATA       354
    54       READ_ADD         398
    48       WRITE            1419
    Covered Transitions :
    -----
    Line      Trans_ID      Hit_count      Transition
    ---      -----
    32       0              737      IDLE -> CHK_CMD
    44       1              119      CHK_CMD -> READ_DATA
    42       2              133      CHK_CMD -> READ_ADD
    39       3              476      CHK_CMD -> WRITE
    36       4                9      CHK_CMD -> IDLE
    62       5              119      READ_DATA -> IDLE
    56       6              132      READ_ADD -> IDLE
    50       7              476      WRITE -> IDLE
    Summary      Bins    Hits    Misses   Coverage
    -----      ----  -----  -----  -----
    FSM States          5       5       0     100.00%
    FSM Transitions     8       8       0     100.00%

```

```

Statement Coverage:
Enabled Coverage      Bins   Hits   Misses  Coverage
-----  -----  -----  -----  -----
Statements          43     43      0  100.00%
=====
Statement Details
=====
Statement Coverage for instance /top/Wrapper_inst/SLAVE_Instance --
Line    Item           Count  Source
---  ---
File SPI_slave.sv
2      module SPI_slave(SPI_slave_inf.dut if_c);
3
4          localparam IDLE      = 3'b000;
5          localparam CHK_CMD   = 3'b001;
6          localparam WRITE     = 3'b010;
7          localparam READ_ADD  = 3'b011;
8          localparam READ_DATA = 3'b100;
9
10         reg [3:0] counter;
11         reg      received_address;
12
13         reg [2:0] cs, ns;
14
15     1           3651  always @(posedge if_c.clk) begin
16             if (~if_c.rst_n) begin
17                 cs <= IDLE;
18             end
19             else begin
20                 cs <= ns;
21             end
22             cs_shared <= cs; // shared variable
23             ns_shared <= ns; // shared variable
24         end
25
26     1           7596  always @(*) begin

```

```

Statement Coverage:
Enabled Coverage      Bins   Hits   Misses  Coverage
-----  -----  -----  -----  -----
Statements          11     11      0  100.00%
=====
Statement Details
=====
Statement Coverage for instance /top/Wrapper_inst --
Line    Item           Count  Source
---  ---
File Wrapper.sv
1      module Wrapper (Wrapper_inf.dut if_wrapper , SPI_slave_inf.dut if_c , RAM_if.dut if_ram);
2
3          RAM  RAM_instance  (if_ram);
4          SPI_slave SLAVE_instance (if_c);
5
6     1           10626  always @(*) begin
7         1           10626  if_c.MOSI = if_wrapper.MOSI;
8         1           10626  if_c.SS_n = if_wrapper.SS_n;
9         1           10626  if_c.rst_n = if_wrapper.rst_n;
10        1           10626  if_wrapper.MISO = if_c.NISO;
11        1           10626  if_c.rst_n = if_wrapper.rst_n;
12
13            if(if_c.rx_valid) begin
14                1           2481   if_ram.din = if_c.rx_data;
15            end
16                1           10626  if_ram.rx_valid = if_c.rx_valid;
17                1           10626  if_c.tx_data = if_ram.dout;
18                1           10626  if_c.tx_valid = if_ram.tx_valid;
19                1           10626  if_ram.rst_n = if_wrapper.rst_n;

```

```

Statement Coverage:
Enabled Coverage      Bins    Hits    Misses Coverage
-----      ----   ----   ----- -----
Statements          10      10      0     100%
=====Statement Details=====
Statement Coverage for instance /top/Wrapper_inst/RAM_instance --
Line      Item      Count    Source
----  -----
File RAM.sv
1           module RAM (RAM_if.dut DUT_if);
2
3           logic[7:0] MEM [255:0];
4           logic [7:0] Rd_Addr, Wr_Addr;
5
6           1           11000  always @(posedge DUT_if.clk) begin
7                           if (~DUT_if.rst_n) begin
8                               DUT_if.dout <= 0;
9                               DUT_if.tx_valid <= 0;
10                          Rd_Addr <= 0;
11                          Wr_Addr <= 0;
12
13                           end
14                           else begin
15                               if (DUT_if.rx_valid) begin
16                                   case (DUT_if.din[9:8])
17                                       2'b00 : Wr_Addr <= DUT_if.din[7:0];
18                                       2'b01 : MEM[Wr_Addr] <= DUT_if.din[7:0];
19                                       2'b10 : Rd_Addr <= DUT_if.din[7:0];
20                                       2'b11 : DUT_if.dout <= MEM[Rd_Addr];
21
22                           endcase
23
24                           end
25           1           10910  DUT_if.tx_valid <= (DUT_if.din[9] && DUT_if.din[8])? 1'b1 : 1'b0;

```

```

Branch Coverage:
Enabled Coverage      Bins    Hits    Misses Coverage
-----      ----   ----   ----- -----
Branches          39      39      0     100.00%
=====Branch Details=====
Branch Coverage for instance /top/Wrapper_inst/SLAVE_instance
Line      Item      Count    Source
----  -----
File SPI_slave.sv
-----IF Branch-----
16           3651  Count coming in to IF
16           90      if (~if_c.rst_n) begin
19           3561  else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----CASE Branch-----
27           7596  Count coming in to CASE
28           1475  IDLE : begin
34           1001  CHK_CMD : begin
48           3180  WRITE : begin
54           1066  READ_ADD : begin
60           873   READ_DATA : begin
Branch totals: 6 hits of 6 branches = 100.00%
-----IF Branch-----
29           1475  Count coming in to IF
29           738   if (if_c.SS_n)
31           737   else
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
35           1001  Count coming in to IF
35           9      if (if_c.SS_n)
37           992   else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
38           992   Count coming in to IF
38           723   if (~if_c.MOSI)
40           269   else begin
Branch totals: 2 hits of 2 branches = 100.00%
-----IF Branch-----
41           269   Count coming in to IF
41           150   if (/received addrcs) //fix

```

```
=====
== Instance: /top/Wrapper_inst
== Design Unit: work.Wrapper
=====
Branch Coverage:
  Enabled Coverage      Bins    Hits    Misses Coverage
  -----      -----  -----  -----  -----
  Branches          2       2       0   100.00%
=====
=====Branch Details=====
Branch Coverage for instance /top/Wrapper_inst
  Line      Item      Count      Source
  ----      ---      -----      -----
  File Wrapper.sv
  -----IF Branch-----
  13           10626      Count coming in to IF
  13           2481      if(if_c.rx_valid) begin
  8145      All False Count
Branch totals: 2 hits of 2 branches = 100.00%
```